SUBROUTINES LIST
for ITWOM 3.0 41 subroutines

alphabetical by name

name	line number
abq_alos	255
adiff	423
adiff2	480
ahd	236
aknfe	155
alos	874
alos2	913
area	2772
ascat	757
avar	1505
curve	1490
d1thx	2029
d2thx2	2084
deg2rad	2319
fht	166
FORTRAN_DIM	144
h0f	198
hzns	1712
hzns2	1765
ITMAreadBLoss	2844
ITWOMVersion	2862
lrprop	1056
lrprop2	1236
int mymax	120
double mymax	136
int mymin	112
double mymin	128
point_to_point	2427
point_to_point_ITM	2329
point_to_pointDH	2673
point_to_pointMDH_two	2568
qerf	2004
qerfi	830
qlra	1013
qlrpfl	2139
qlrpfl2	2211
qlrps	851
qtile	1935
saalos	260
z1sql	1854
z1sql2	1893

by line number:

line number	name
112	int mymin
120	int mymax
128	double mymin
136	double mymax
144	FORTRAN_DIM
155	aknfe
166	fht
198	h0f
236	ahd
255	abq_alos
260	saalos
423	adiff
480	adiff2
757	ascat
830	qerfi
851	qlrps
874	alos
913	alos2
1013	qlra
1056	lrprop
1236	lrprop2
1490	curve
1505	avar
1712	hzns
1765	hzns2
1854	z1sql
1893	z1sql2
1935	qtile
2004	qerf
2029	d1thx
2084	d2thx2
2139	qlrpfl
2211	qlrpfl2
2319	deg2rad
2329	point_to_point_ITM
2427	point_to_point
2568	point_to_pointMDH_two
2673	point_to_pointDH
2772	area
2844	ITMAreadBLoss
2862	ITWOMVersion

SUBROUTINE ADIFF: A functional explanation, by Sid Shumate.

Last Revised: March 9, 2008.

Attenuation from Diffraction for ITWOM subroutine.

Note: Used with both point-to-point and area modes. Called by ***lrprop.***

Descriptions derived from the Irregular Terrain Model description by George Hufford, 2002, (ITMD), compared to the ITM.cpp prepared by J. D. McDonald and John Magliacane for compilation on unix and linux systems. "Line" numbers refer to the ITM.cpp as line numbered by Bloodshed Software's DevC++ print function. "Alg" numbers refer to the algorithm formula in "The ITS Irregular Terrain Model, version 1.22, the Algorithm" by G. A. Hufford, 1995 (the Algorithm). "ITS67" numbers refer to the algorithm formulas in "ESSA Technical Report ERL 79-ITS 67, Prediction of Tropospheric Radio Transmission Over Irregular Terrain, A Computer Method – 1968" by A.G.Longley and P.L.Rice.

For additional background on this function, see section 3-2, page 3-8 to 3-13 of ESSA Technical Report ERL 79-ITS 67 (ITS-67).

From ITMD Section 10, 11, 12:

The function ***adiff*** finds the "diffraction attenuation" at the distance *d*. It uses a convex combination of smooth earth diffraction and double knife-edge diffraction. A call with *d* = 0 is necessary to set up initial constants.

Call inputs:

d distance from transmit site at which diffraction attenuation is to be determined.

Prop_type

&prop array *prop* with array elements:

propa_type

&propa array *propa* with array elements:

defines private, or local, arguments:

prop_zgnd an array containing values of the *zgnd* surface transfer impedance, with elements:

prop.zgndreal, the real, (resistive) component of the surface transfer impedance;

prop.zgndimag; the imaginary, (reactive) component.

wd1	1/wd, the inverse of the weighting factor wd
xd1	dla added to a curvature adjustment equal to tha/gme.
afo,	attenuation from absorption and scattering from oxygen, water vapor, precipitation and terrain clutter.
qk	the product of the effective height of the transmit receive terminal multiplied by the receive terminal effective height, less the product of the transmit receive terminal height AGL multiplied by the receive terminal height AGL; later set to be equal to 1/|zgnd|.
aht,	coefficient; see text below.
xht;	coefficient; see text below.
a	preset to be equal to $dl^2/2he$ *(*then to *ds/th)* for each terminal site
q	utility variable; used for temporary holding of a value
pk	absolute value of K; stated as |K| in [Alg. 6.6]
ds	the difference, in meters, between prop.dist, the total path distance, and dla, the sum of the horizon distances. It equals zero for one obstruction; positive for multiple obstructions; negative for line-of-sight.
th	the sum of the transmit and receive take-off angles to the peaks of the highest visible obstructions.
wa	equal to $(a * wn)^{1/3}$; calculated twice, once for each terminal.
ar	attenuation due to rounded edge diffraction
wd	the weighting factor for knife edge versus rounded edge
adiffv	attenuation due to diffraction, the output value

This subroutine:

Uses *d*, prop_type, propa_type, and other information in arrays prop and propa in order to calculate the attenuation due to diffraction over an obstacle at a distance d using a convex combination of smooth earth diffraction and double knife-edge diffraction.

1. In steps 1-3, wd1, the inverse of the weighting factor wd, is calculated. An **if** statement is initiated; it operates from lines 225 to 254. If *d* is not equal to zero, go to the **else** statement in step 13 below, on line 256. If *d* is equal to zero, coefficients wd1, xd1, afo, ql, aht and xΔt will be determined:

   a. *q* is set to be equal to prop.hg[0] times prop.hg[1], the product of the transmit antenna height above ground level, multipled by the receive antenna height above ground level; units in meters, so output is in square meters.
   b. *qk* is set to be equal to the product of the effective height of the transmit antenna, prop.he[0], multipled by the effective height of the receive

antenna, prop.he[1], less the value of *q* from step a.; output is in square meters.

Line 225: if (d = = 0)
q=prop.hg[0]*prop.hg[1];
qk=prop.he[0]*prop.he[1]-q;

2. A second **if** statement is initiated within the first; if prop.mdp, the mode of the propagation model, is less than zero, indicating operation in the point-to-point mode, the value of q is increased by 10.

Line 230: if (prop.mdp<0.0)
q+=10.0;

3. *wd1* , the inverse of the weighting factor wd used to weigh between knife edge and rounded edge diffraction, is set to be equal to the square root of (1 + qk/q).

Line 233: wd1=sqrt(1.0+qk/q);

4. *xd1* is set to be equal to propa.dla + propa.tha/prop.gme.

Line 234: xd1=propa.dla+propa.tha/prop.gme;

5. In steps 5 and 6, the Δh(s), the terrain irregularity factor Δh, determined at a specified distance s, and σn(s), the standard deviation of Δh(s), is determined. The value held by *q* is reset to be equal to the terrain irregularity parameter, dh (a.k.a. delta h or Δh), multiplied by the distance compensation term (1.0-0.8*exp(-propa.dlsa/50,000)), using a formula derived from Alg. (3.9). See subroutine ***qlrps*** step 23, for the derivation. At this point, the value of *q* represents Δh(s).

Line 235: q=(1.0-0.8*exp(-propa.dlsa/50e3))*prop.dh;

6. *q* is then further modified by setting it to be equal to the value of *q* obtained on line 235 in step 5 above, multiplied by 0.78*exp(-pow(q/16.0,0.25). At this point, the value of q represents the σn(s).

This step utilizes the formula:

$$\sigma_h(s) = 0.78\, \Delta h(s) \exp\left[-\left(\Delta h(s)/H\right)^{1/4}\right] \quad \text{with } H = 16 \text{ meters.''} \qquad \text{[Alg. 3.10]}$$

This formula, is found in the Algorithm, shows the relationship between Δh and the terrain roughness factor σ_h used in Tech Note 101. Here it is used to convert the value stored in *q* from the value for Δh(s) to the value for σ_h(s).

Line: 236: q*=0.78*exp(-pow(q/16.0,0.25));

7. In this step, we determine the attenuation from absorption and scattering due to oxygen, water vapor, precipitation, and terrain clutter. The value of *afo*, Attenuation From Other, is set to be equal to the lesser of:

a. 15
b. (2.171*log(1.0+4.77e-4*prop.hg[0]*prop.hg[1]*prop.wn*q)); [Alg. 4.10]
   Where:
   hg[0] is the transmit antenna height above ground in meters;
   hg[1] is the receive antenna height above ground in meters;
   prop.wn is the wave number, (equal to freq. in MHz/47.7)
   q is currently equal to the value of $\sigma_h(s)$, the terrain roughness factor at a specified distance "s", (i.e., with distance correction).

8. The value of *qk* is reset to be equal to 1/(absolute value of *prop_zgnd*).

prop_zgnd is a complex double, representing the earth's surface transfer impedance with two elements; a real element, the resistance value, and an "imaginary" value, the reactance, which describes the phase mismatch between the voltage and the current, in terms of a capacitive value (current peak leads voltage peak) or a inductive value (current peak lags behind voltage peak).

Line 238: qk=1.0/abs(prop_zgnd);

9. The value of *aht* is set to be equal to 20.0.

Line 239: aht=20.0; [Alg. 6.7]

10. The value of x*ht* is set to be equal to 0.0.

Line 240: xht=0.0;

11. A **for** statement is initiated with two loops, j=0 and j=1. The **for** loop starts with j=0:

Line 242:
```
for (int j=0; j<2; ++j)
{
```

a. On the first for loop, the value of *a* is set to be equal to:
   $0.5*(prop.dl[0])^2/prop.he[0]$; [Alg. 4.15]
   Where:
   prop.dl[0] is the distance from the transmit site to the horizon
   prop.he[0] is the effective height of the transmit site

Line 245: a=0.5*(prop.dl[j]*prop.dl[j])/prop.he[j];

b. The value of *wa* is set to be equal to $(a*prop.wn)^{1/3}$ [Alg. 4.16]
   Where:

a was determined in step 11(a.).
prop.wn is the wave number, = (frequency in MHz/47.7)

Line 246: wa=pow(a*prop.wn,THIRD);

c. The value of *pk* is set to be equal to *qk/wa*. [Alg. 4.17]
Where:
qk was determined in step 8.
wa was determined in the last step, 11(b).

Line 247: pk=qk/wa;

d. The value of *q* is again reset, this time to be equal to:
((1.607-pk)*151.0*wa*prop.dl[0]/*a*; [Alg. 4.18 and 6.2]

Line 248: q=(1.607-pk)*151.0*wa*prop.dl[j]/a;

e. The value of *xht* is increase by adding the value of *q*.[Alg. 4.19 –height -gain]

Line 249: xht+=q;

f. Subroutine ***fht*** is called with inputs *(q,pk).* Subroutine ***fht*** then returns *fhtv*, the height-gain over a smooth spherical earth for use with the three-radii method. The value of *aht* is increased by adding the value returned by ***fht***. [Alg. 4.20]

Line 250: aht+=fht(q,pk);
}

The **for** loop then repeats, with j=1:

g. The value of *a* is set to be equal to: $0.5*(\text{prop.dl[1]})^2/\text{prop.he[1]}$; [Alg. 4.15]
Where:
prop.dl[0] is the distance from the receive site to the horizon
prop.he[0] is the effective height of the receive site

h. The value of *wa* is set to be equal to $(a*prop.wn)^{1/3}$ [Alg. 4.16]
Where:
a was determined in step 11(g.).
prop.wn is the wave number, = (frequency in MHz/47.7)

i. The value of *pk* is set to be equal to *qk/wa*. [Alg. 4.17]
Where:
qk was determined in step 8.
wa was determined in the last step, 11(h).

j. The value of *q* is again reset, this time to be equal to: [Alg. 4.18 and 6.2]
((1.607-pk)*151.0*wa*prop.dl[1]/*a*;

k. The value of *xht* is increase by adding the value of *q*. [Alg. 4.19]

l. Subroutine ***fht*** is called with inputs *(q,pk).* Subroutine ***fht*** then returns *fhtv*, the height-gain over a smooth spherical earth for use with the three-radii method. The value of *aht* is increased by adding the value returned by ***fht.***.

The **for** loop completes. The initial diffraction constants have been calculated, and the subroutine proceeds to report out diffraction attenuation = 0.0, indicating a coefficient setup run has completed:

12. *adiffv* is then set equal to zero.

Line 253: adiffv=0.0;
}

13. The **if** statement on line 225 has a matching **else** statement on line 256. Therefore, if the input value *d* is not equal to zero, indicating a second execution, following the required first execution with d=0 to set the coefficients, then:

Line 256: else
{

a. *th* is set to be equal to *propa.tha* + *d***prop.gme*; [Alg. 4.12]
Where:
propa.tha, the total bending angle, set in ***lrprop;*** in radians (after correction).
d is the distance at which the attenuation is to be calculated.
gme is the earth's effective radius.

Line 258: th=propa.tha+d*prop.gme;

b. *ds* is set to be equal to *d* − *propa.dla*;
Where:
d is the distance at which the attenuation is to be calculated.
propa.dla is the sum of the two horizon distances, all in meters.

NOTE 1: ds is the difference, in meters, between prop.dist, the total path distance, and dla, the sum of the horizon distances. It equals zero for one obstruction; positive for multiple obstructions; negative for line-of-sight.

NOTE 2: The following calculations produce absurd and non-a-number or infinity results if ds = 0.0; this is why the original ITM version of this subroutine cannot compute diffraction over a single obstacle, only multiple obstacles.

Line 259: ds=d-propa.dla;

c. *q* is reset to be equal to 0.0795775*prop.wn*ds*th*th; At this point, the value of *q* represents Δh(s).

Line 261: q=0.0795775*prop.wn*ds*th*th;

d. subroutine ***aknfe*** is called twice to calculate the diffraction loss due to a knife edge at two adjacent peaks;
the first time with input: (q*prop.dl[0]/(ds+prop.dl[0])),
and the second time with input: (q*prop.dl[1]/(ds+prop.dl[1]));
in each case, ***aknfe*** reports out *a*, the attenuation due to a single knife edge diffraction; the Fresnel integral (in decibels) as a function of the input, v^2.
Adiffv is then temporarily set to equal the sum of the two outputs from ***aknfe***, the knife edge diffraction from two knife edges. [Alg. 4.14]

Line 262:
adiffv=aknfe(q*prop.dl[0]/(ds+prop.dl[0]))+aknfe(q*prop.dl[1]/(ds+prop.dl[1]));

Next, the attenuation due to rounded earth is calculated in steps e. to h.

e. *a* is set to equal *ds/th***,** in meters/radian. Note that if ds =0.0, (see above NOTE 2.) this calculation returns "inf", indicating that it attempted to compute infinity.

Line 263: a=ds/th;

f. *wa* is set to be equal to $(a*prop.wn)^{1/3}$ [Alg. 4.16]

Line 264: wa=pow(a*prop.wn,THIRD);

g. *pk* is set to equal the value of *qk*/*wa*.

Line 265: pk=qk/wa; [Alg 4.17]

q is reset to be equal to (1.607-*pk*) * 151.0 * *wa* * *th* + *xht* [Alg. 4.18 and 6.2]

Line 266: q=(1.607-pk)*151.0*wa*th+xht;

h. *ar* is the rounded earth attenuation, A_r, calculated as equal to:

$$0.05751 * q - 4.343 * \log(q) - aht \qquad \text{[Alg. 4.20]}$$

which can be better stated using the specified $\log_{10}$, as:

$$A_r = 0.05751 * q - 10 * \log10(q) - aht$$

Line 267: ar=0.05751*q-10*log10(q)-aht;

Next, the process of calculating wd, the weighting factor, wd, for knife edge vs rounded edge, is completed using wd1 and xd1.

i. *q* is reset to be equal to:

(*wd1*+*xd1*/*d*)*mymin(((1.0-0.8*exp(-*d*/50e3))**prop.dh***prop.wn*),6283.2) [Alg. 4.9]

Line 268: q=(wd1+xd1/d)*mymin(((1.0-0.8*exp(-d/50e3))*prop.dh*prop.wn),6283.2);

j. wd, the edge weighting factor, is set to be equal to:
(25.1/(25.1+sqrt(*q*))) [Alg. 4.9]

Line 269: wd=25.1/(25.1+sqrt(q));

k. *adiffv* , here representing the total diffraction attenuation, is set to be equal to: *ar* * *wd* + (1.0-*wd*) * *adiffv* + *afo* [Alg. 4.11]

where;
ar is the rounded earth attenuation
afo is the attenuation from all other sources.

Line 270: adiffv=ar*wd+(1.0-wd)*adiffv+afo;
}

14. The subroutine then returns the value of *adiffv*, the “diffraction attenuation” at the distance *d*.

return adiffv;

SUBROUTINE ADIFF2: A description for ITWOM, by Sid Shumate.

Revised Sept. 28, 2010 from straight line to 2nd scattered-diffracted path obstruction loss.
Revised Sept. 27, 2010. to change obstruction loss to a scattered signal less diffracted signal cancellation. Use with itwom2.0s.
Revised Sept. 25 to add effective earth curvature to v2 distance calculations.
For use with itwom2.0r.
Revised Sept. 23, 2010.
Revised: Aug 29, 2010. For itwom2.0e Being revised to add obstruction top foliage scatter adjustment.

For itwom2.0c. Aug. 11 Revised to include knife edge diffraction when receiver antenna is above clutter canopy, but still behind obstruction with grazing angle between 0.2 and 1.22 radians.

Aug. 8; Discussion of Knife Edge functions added. Rounded edge from single knife edge completed. Modified to add clutter loss to all secondary (post-obstruction) scenarios.

Attenuation from Diffraction for ITWOM subroutine.

Note: Used with both point-to-point and area modes. Called by ***lrprop2.***

Descriptions derived from the Irregular Terrain Model description by George Hufford, 2002, (ITMD), compared to the ITM.cpp prepared by J. D. McDonald and John Magliacane for compilation on unix and linux systems. "Line" numbers refer to the ITM.cpp as line numbered by Bloodshed Software's DevC++ print function. "Alg" numbers refer to the algorithm formula in "The ITS Irregular Terrain Model, version 1.22, the Algorithm" by G. A. Hufford, 1995 (the Algorithm). "ITS67" numbers refer to the algorithm formulas in "ESSA Technical Report ERL 79-ITS 67, Prediction of Tropospheric Radio Transmission Over Irregular Terrain, A Computer Method – 1968" by A.G.Longley and P.L.Rice.

Additional references may be made to: "Tropospheric Radio Wave Propagation over Irregular Terrain; the Computation of Field Strength for UHF Broadcasting", Research Department Report No. 1971/43, Nov. 1971, (BBC Report) Engineering Division, British Broadcasting Corporation.

For additional background on this function, see section 3-2, page 3-8 to 3-13 of ESSA Technical Report ERL 79-ITS 67 (ITS-67), and Sections 5,7 and 8 of Tech Note 101, Volume 1.

From ITMD Section 10, 11, 12:

The function ***adiff2*** finds the "diffraction attenuation" at the distance *d*. It uses a convex combination of smooth earth diffraction and double knife-edge diffraction for a two-obstacle scenario, and knife-edge diffraction for a single obstruction. Clutter loss is calculated and added for all paths. A minimum of two runs are required; the first with *d* = 0.0, is necessary to set up initial constants. The second run calculates the attenuation at a distance d.

Calculation of Attenuation Loss for An Obstacle with a Knife-Edge Peak:

The subroutine ***adiff2*** prepares the coefficients, then calls subroutine ***aknfe*** with input v^2, where v is the internal wedge angle of an obstruction, to determine the attenuation for an obstruction with a sharp peak; whose peak approximates the shape of a knife edge perpendicular to the radio signal path.

The argument v represents the "internal wedge phase angle", and contains both an angle component and a frequency component. It is equal to:

$$v = 2*(\Delta r/\lambda)^{1/2} \qquad \text{[TN101 7.1a]}$$

Δr is defined as: Δr = the sum of the distance from the transmit antenna to the knife edge + distance from the knife edge to the receive antenna + distance equivalent of the phase reversal at the reflection point, less the length of a direct ray signal between the transmitter and the receiver antennas, even if it passes through the earth or an obstruction.

There is a phase reversal at the reflection point that is always there for horizontal polarity, but is only found for vertical polarity at low grazing angles below the pseudo-Brewster angle. This is considered in the phasing adjustments coefficient K.

Δr is often approximated as: $\Delta r = dl_0 + dl_1 - d = \theta^2(dl_0 * dl_1/2d)$, but these approximations use several assumptions; the latter is particularly troublesome in that it includes assumptions that the knife edge is at a distance from both the transmitter and receiver locations, and approximately centered between the two. This is often not the case when computing a radial, as the receive point is moved up, over, and down the other side of each obstruction in the path, so it is better to use a more rigorous geometric calculation of Δr.

The original ITM uses the latter approximation, which can be approximately stated in terms of frequency in MHz:

$$v = +\,2.583\ \theta(f_{MHz}\ dl_0 * dl_1/d)^{1/2}, \qquad \text{[TN101 7.1b]}$$

in that dl_0 and dl_1 are approximately equal to the path lengths from the terminals to the peak of the obstruction, except when a terminal is very close to a tall obstruction. Converting frequency to wave number, and squaring v to effectively create the modulus, as used in the source code:

$$v = \pm 123.21\theta(wn*dl_0*dl_1/d)^{1/2},$$

$$v^2 = 15180.48*\theta^2(wn*dl_0*dl_1/d)$$

Where λ is the wavelength. Note that both Δr and λ have to be stated in the same units of length (usually meters or kilometers) for v to be correctly defined, and this can only be used for an obstruction that breaks the line between the transmitter and receiver, as the sign would otherwise have to be transferred when v goes negative.

The term Δr is the difference, stated as a length between the sum of: the length of the signal path ray distance from the transmit antenna to the top of the obstruction and the signal path ray from the top of the obstruction to the receive antenna, less the direct line distance between the transmitter and the receiver. Does this look familiar from the previous discussion of two-ray Multipath cancellation in the line of sight range? When this difference is divided by the wavelength, the resulting term $\Delta r/\lambda$ represents the difference in length between the path distance of the direct path and the diffracted path, specified in number of wavelengths. From this, we can see that the diffraction effects of an obstruction below the direct path, as it approaches the direct path, are related to the effects of a two-ray reflection occurring off of the ground or clutter canopy; both can be explained in terms of the effects of the wavelet interaction that produce the Cornu spiral. Take twice the square root of this difference, and you have the coefficient v.

The coefficient *v* is then used to compute the diffraction approximation formula for a single knife edge, in the subroutine ***aknfe***, which uses:

$$\text{For } v > 3,\ A(v,o) = 12.953 + 20\log_{10}(v) \text{ in dB.} \qquad \text{[TN101 7.2]}$$

The input *q* as used as an input to ***aknfe*** represents the square of *v*. The argument *v* can be negative, if the knife edge is approaching but not breaking the line between the transmitter and the receiver. Once the knife edge cuts the direct path, v becomes positive. The above equation is modified in ***aknfe*** for use with v^2 as:

For v2<5.7, a=6.02+9.11*sqrt(v2)-1.27*v2

For v2≥5.7, a =12.953+10log10(v2).

Calculation of Attenuation Loss for Two Obstacles with Knife-Edge Peaks:

There is a significant difference between the dual knife edge method and the rounded earth method. The dual knife edge method uses a version of a Vogler "three radii" method, the three radii being:

1. The arc between the transmitter and the highest visible obstruction from the transmitter,
2. The arc between the receiver and the highest visible obstruction from the receiver, and
3. The arc between the two obstacle peaks.

However, the methodology used in the ITM for knife edge diffraction has been modified from the methodology that L. E. Vogler developed for use in the ITS-67 software. Vogler used two separate calculations of knife edge-diffraction for each obstruction peak and averaged the results of the two for each peak; the ITM methodology was changed to a Epstein-Peterson methodology, which uses a single calculation of diffraction for each peak. The ITM considers the path from the transmitter to the receiver obstacle peak as one knife-edge diffraction path, then considers the path from the transmitter obstacle peak to the receiver as the second knife-edge diffraction path, then sums the results of the two calculations.

Rounded Obstacle Attenuation for Two Obstacles

For rounded earth, a much different, "four radii" method, described in Tech Note 101 section 8, developed by Vogler in 1964, is used. The rounded earth methodology considers four radii;

1. The arc between the transmitter and the highest visible obstruction from the transmitter, with radius a_1;
2. The arc between the receiver and the highest visible obstruction from the receiver, with radius a_2;
3. The arc between the transmitter obstruction peak and a theoretical knife edge halfway between the two obstacle peaks, with radius a_t;
4. The arc between the receiver obstruction peak and a theoretical knife edge halfway between the two obstacle peaks, with radius a_r.

These radii are shown graphically in Figure 8.7 of Tech Note 101. Note that the radii a_t and a_r end at the circled points where they cross the vertical centerline between the transmitter and the receiver; these lines are extended to show how radii a_1 and a_2 end at a junction with the extensions of a_t and a_{tr}.

Computing the rounded earth diffraction (treated as two low obstructions, and also used for multiple obstructions) the first run of the subroutine, with q =0.0, prepares the coefficients for an area mode calculation, or for subsequent point-to-point mode calls. In the Algorithm, George Hufford stated that the calculation for A_r, the rounded earth attenuation, consisted of summing four functions:

$$A_r = G(x_0) - F(x_1, K_1) - F(x_2, K_2) - C(K_0) \qquad \text{[TN101 8.1],[Alg. 4.20]}$$

Note that the two F functions, and the C function, are subtracted from the G function.

This is computed as the attenuation from the theoretical knife edge in four functions, with each function computing for coefficients related to combinations of these four paths;

a. a path from the transmitter to the peak of the transmitter horizon obstacle,
b. a path from the transmitter obstacle peak to a theoretical peak between the transmitter and receiver obstacles,
c. a path from the theoretical peak to the peak of the receiver horizon obstacle, and
d. a path from the peak of the receiver horizon obstacle on to the receiver computed as the "G" function, and from which the "F" functions, derived from the #1 and #2 arcs above, respectively, and the "C" function, are subtracted.

And that these four consisted of:

$$G(x) = 20\log(x^{-1/2}e^{x/A}) \qquad \text{[Alg. 4.23]}$$

where A is a dimensionless constant equal to 151.03.

G(x) can be visualized as the diffraction loss over a theoretical knife edge at the center of the rounded obstruction area between the two obstructions, (which may be smooth or nearly smooth earth horizons or peaked obstacles). It starts as a computation of signal cancellation due to the difference between the path length of a direct signal between the transmitter and the receiver (even as passes through the theoretical knife edge or the earth), and a path from the transmitter antenna to the top of the (theoretical) knife edge and to the receiver site, as per standard Fresnel-Kirchhoff optical diffraction theory.

The term x in the G(x) equation is a combination of advanced mathematical functions: Airy Integrals. Note that it includes the terms x_1 and x_2, which have been summed in the argument *xht*, and added to x, which is represented by *q* at that point in the subroutine.

$$F(x_1, K_1) = 20\log|(\pi/(2^{1/3}AB))^{1/2}Wi(to - (x_1/(2^{1/3}AB))^2)| \qquad \text{[Alg. 4.24]}$$

The first F function represents attenuation between the transmitter site and the transmitter obstruction computed as a signal cancellation due to a two-ray effect; the difference between the path length of a direct signal between the transmitter site and the transmitter obstacle peak, and a path from the transmitter site to a reflection point between the transmitter and the obstacle peak, and then to the top of the transmitter side obstacle peak.

$$F(x_2, K_2) = 20\log|(\pi/(2^{1/3}AB))^{1/2}Wi(to - (x_2/(2^{1/3}AB))^2)| \qquad \text{[Alg. 4.24]}$$

The second F function represents attenuation between the receiver site and the receive obstruction computed as a signal cancellation due to a two-ray effect; the difference between the path length of a direct signal between the receive site and the receive obstacle peak, and a path from the receive site to a reflection point between the receiver and the obstacle peak, and then to the top of the transmitter side obstacle peak.

and:

$$C(K) = 20\log|1/2(\pi/(2^{1/3}AB))^{1/2}(2^{2/3}K^2t_0 - 1)Wi'(t_0)^2| \qquad \text{[Alg. 4.25]}$$

C(K) represents the non-phase-cancellation field strength loss, E_d/E_o, of the amplitude of the signal due to diffraction over a knife edge.

The K function is a sum of phase considerations coefficient. It varies with the reflection coefficient, and with the polarity of the signal.

The $G(x_0)$ function is also stated as:

$$G(x_{3,4}) = 0.05751\ x_{3,4} - 10\log10\ x_{3,4} \quad \text{dB} \qquad \text{[ITS-67 3.37]}$$

The two $F(x_1, K_1)$ attenuation functions are driven by the effective heights of the transmitter and receiver, and each is computed during the first run, using a call to subroutine ***fht*** to compute the functions, then summed with the value preset in argument *aht*. An if statement decides whether to compute the F functions for a single obstruction, where ds = zero, or for two separated obstructions.

The distances x_1 and x_2 used in the two obstruction F function, are defined in the Algorithm as:

$$x_{1,2} = A*B(K_{1,2})*\alpha_{1,2}*\gamma_{1,2}*d_{L1,2} \qquad \text{[Alg. 4.18]}$$

where: A is the dimensionless constant 151.0.

B is the B function of K, approximated as equal to (1607-pk), and limited to 1607 as the phase coefficient factor, K, here represented by the argument *pk*, approaches zero.

$\alpha_{1,2}$ is represented by the argument *wa*, which represents a term equal to: $(a_{1,2}*wn)^{1/3}$.

$a_{1,2}$ here represents the radius of the arc between the terminal and the horizon obstacle peak, 1=transmit, 2=receive. The argrument $a_{1,2}$ is computed as:

$$a_1 = d_{L1}^2/(2*h_{e1}),\ a_2 = d_{L2}^2/(2*h_{e2}), \text{ all units in meters.} \qquad \text{[Alg. 4.15]}$$

$\gamma_{1,2}$ is the terminal to horizon curvature, equal to $1/a_{1,2}$. [ITS-67 3.29a]

wn is the wave number, equal to the frequency in MHz divided by 47.7.

d_{L1}, d_{L2} are the horizon distances, in meters.

K, the phase coefficient factor, is represented by the argument *pk*. The argument *pk* is equal to *qk*/*wa,* where qk is the inverse of the absolute value of the ground impedance, zgnd, a complex number. So:

$$K = pk = qk/wa = 1/(\,|zgnd\backslash{*}wa) = 1/[\,|zgnd\backslash{*}(a_{1,2}{*}wn)^{1/3}] \quad \text{(xxx)}$$

The Algorithm notes that "for those values of K in which we are interested it is a good approximation to say $C_1(K) = 20$ dB." The preset value of *aht*, equal to 20.0, is therefore an approximation for C(K), and:

$$Ar = 0.05751 * q - 10*\log10(q) - aht \quad \text{[Alg. 4.20, 4.23,4.24, and 4.25]}$$

Where q, at that point, holds the value for $x_{1,2}$. ITS-67 states that x for rounded earth is defined by [ITS-67 3.9a] to be $x = 18000\sigma/f$, where the frequency f is in MHz and the ground constant σ for average ground is 0.005 mho/m. (Table 2, p.8, ITS-67].

And we can now apply the theory, methodology and equations shown above, into the source code below.

For a two-obstacle scenario, where the receive site is beyond the peak of the 2nd obstacle, the Irregular Terrain Models compute both the knife edge and rounded edge attenuation values, and combine the results in a weighted manner to provide a weighted average value of attenuation. The weighting system is based on the roughness of the terrain, as quantified in the delta-h ($d\Delta$) coefficient, as determined by the extended version of the terrain irregularity factor computation employed in the ITM and the corrected version of the terrain irregularity factor computation employed in the ITWOM. The calculation of the terrain irregularity factor delta-h, as described in the FCC rules and regulations is limited to line-of-sight use, and therefore to a maximum distance of 50 km; the ITM version, which is specifically designed for use with a 30-arc-second database only, extends into the diffraction range, and calculates out to 25 terrain database intervals, or up to 800 kilometers. The ITWOM calculates out to 800 kilometers, irrespective of the database used. (See chapter on subroutine ***d1th1*** or ***d1th2***) This weighing system was developed by Longley and Rice for the ESSA ITS-67 computer implementation, and is not discussed in Tech Note 101.

When two obstructions are found, but the receive antenna is located at the peak of the 2nd obstruction, the field strength arriving at the second peak is calculated using knife-edge methodology only (see the section below on "Rounded Obstacle Attenuation for a Single Obstacle"). To this, an additional knife edge-peak diffraction loss of 5.8 dB is added, and the second peak is treated as a secondary transmission site. The path attenuation from the second peak to the receive site is calculated in a similar manner to that stated above for a path just past a single obstruction, until the receive site grazing angle drops below 0.2 radians.

When the receive site grazing angle past the second obstruction peak is less than 0.2 radians, the original ITM methodology developed by Vogler, with a correction for the

error that crept into the ITM when Vogler's two-obstruction, four-calculation method was simplified to a two-calculation method, is used to calculate the attenuation to the receive site.

Past the peak of a first obstruction, we take a cue from the Epstein-Petersen methodology. As the receive site moves past the diffraction point at the peak of the obstruction, if the receive site grazing angle exceeds 0.2 radians (11.5 degrees), the field strength diffracted across the peak of the obstruction is treated as a secondary transmitted signal, and the path from the obstruction peak to the receiver is treated as a line-of-sight transmission from the obstruction peak to the receiver site. The "back of the obstruction" path is processed in one of three methods.

If the receive site grazing angle exceeds 1.22 radians (70 degrees), the path beyond the obstruction peak is heavily shadowed; the obstruction as viewed from the receiver is either a cliff face or tall building, and an empirically derived 20 dB of loss, similar to the C(K) approximation above, is added.

If the receive site grazing angle is between 0.6 radians (33 degrees) and 1.22 radians, the path is treated as a signal transmitted at the peak of the first obstruction, at the height of the clutter canopy, to a receive site that is also at or below the clutter canopy line, as a barely line-of-sight cluttered path down the side of a hill or mountain. The Shumate's approximations for cluttered path attenuation are applied in the same manner as they were for the line-of-sight path.

If the receive site grazing angle is between 0.2 and 0.6 radians, the path is treated as a signal transmitted above the height of the clutter canopy, to a receive site that is also at or below the clutter canopy line, but far enough beyond the receive-side base of the obstruction so that a significant portion of the path is through clear air. The Shumate's approximations are applied as they were for the line-of-sight path.

When the receive site is far enough away from the obstruction that the receive site grazing angle is less than 0.2 radians, the original Tech Note 1 Section 7 methodology is referenced. The TN101 methodology was modified to match the weighted knife-edge vs. rounded edge methodology developed by Lewis E. Vogler from Fresnel-Kirchoff diffraction theory, for the ITS-67 implementation. The knife-edge methodology was further simplified from a two-calculation per obstruction method to a Epstein-Peterson single calculation-per-obstruction method for use in the ITM, and is used to calculate the attenuation to the receive site. In the ITWOM, a correction is added for the mathematical error that crept into the ITM when Vogler's two-obstruction, four-calculation method was simplified to a two-calculation method. Then Shumate's approximations are used to calculate and add the clutter loss to the total attenuation.

We also note that at a distance, the terrain irregularity factor (dh) for a path extending over tall obstructions tends to rise to a value in the hundreds of meters, so the weighting of the results causes the knife-edge results to be relied on more than the rounded-edge results.

Attenuation for a Single Obstacle

Since the ITM does not consider a single, isolated obstacle, for the ITWOM it is necessary to derive the computer implementation equations for the ITWOM from the theory provided in Section 7 of Tech Note 101.

In the ITWOM, when the receive site is atop the peak of the first obstruction, the signal is treated as a line-of-sight signal with 5.8 dB of standard knife edge diffraction loss (for *v2*=0.0) added.

For the single-obstruction knife-edge calculation, the equations used for the Epstein-Peterson calculation for the transmitter site to 2nd obstruction peak are applicable, with the 2nd obstruction peak location replaced with the receive site. Subroutine ***aknfe*** is called to calculate the single peak knife edge, this time with the variable q representing the entire value of v^2. The knife edge diffraction attenuation for a single obstacle, A(v,0), is then stored temporarily in variable *adiffv2*.

For a three-arc-second database, the terrain points are approximately 100 meters apart, and as more detailed databases are employed, the terrain interval, the distance between terrain points in the database, is progressively less. In the source code, the distance between the peak of the highest obstacle "visible" from the transmit site, the transmit horizon obstacle, and the peak of the highest obstacle "visible" from the receive site, the receive horizon obstacle, is held by argument *ds*. The ITWOM only performs a single obstruction calculation when *ds* is equal to zero. This only happens when the two horizon obstacle peaks are at the same terrain point (in the source code, when *ds* ==0.0), a common occurrence in a normal radial calculation run. At this point, the obstacle peak is as close to a true "knife-edge" as the program can determine, given the size of the intervals in the terrain database. To also calculate and add in a weighted manner the rounding of a single obstacle would make little or no difference in the result. Therefore, the ITWOM computes a single obstacle as a simple knife-edge. If the two "horizon obstacles" are as little as one terrain database interval apart, (in the source code, when *ds*>0.0) the two-obstacle calculations are utilized, and consideration of rounding of the obstacles is included.

In the ITWOM, when the receive site is atop the peak of the first obstruction, the signal is treated as a line of sight signal with 5.8 dB of standard Fresnel-Kirchhoff knife edge diffraction loss (for *v2*=0.0) added.

Past the peak of a first obstruction, a cue is taken from the Epstein-Petersen methodology. As the receive site moves past the diffraction point at the peak of the obstruction, the field strength diffracted across the peak of the obstruction is treated as a secondary transmitted signal, and the path from the obstruction peak to the receiver is treated as a line-of-sight transmission from the obstruction peak to the receiver site. The "back of the obstruction" path is processed in one of three methods.

If the receive site grazing angle exceeds 1.22 radians (70 degrees), the path beyond the obstruction peak is heavily shadowed; the obstruction as viewed from the receiver is either a cliff face or tall building, and an empirically derived minimum average attenuation of 20 dB, is added to the standard peak diffraction loss of 5.8 dB.

If the receive site grazing angle is between 0.6 radians (33 degrees) and 1.22 radians, the path is treated as a secondary signal, weakened by a standard peak diffraction loss of 5.8 dB, transmitted at the peak of the first obstruction, at the height of the clutter canopy, to a receive site that is also at or below the clutter canopy line, as a barely line-of-sight cluttered path down the side of a hill or mountain. The Shumate's approximations for cluttered path attenuation are then calculated and added in the same manner as they are for a primary line-of-sight path.

If the receive site grazing angle is between 0.6 and 0.2 radians, the path is treated as a secondary signal, weakened by a standard peak diffraction loss of 5.8 dB, transmitted at the peak of the first obstruction, above the height of the clutter canopy, to a receive site far enough beyond the receive-side base of the obstruction so that a significant portion of the path is through clear air. The Shumate's approximations for clutter loss near the receive site, are applied as they were for the line-of-sight path.

If the receive site grazing angle is below 0.2 radians, the path is treated as a signal transmitted above the height of the clutter canopy, with its strength determined by weakening the original transmitter strength by knife edge loss for a single obstacle, computed according to Fresnel-Kirchhoff theory, to a receive site that is also at or below the clutter canopy line, but far enough beyond the receive-side base of the obstruction so that a significant portion of the path is through clear air. The Shumate's approximations for clutter loss near the receive site, are applied as they were for the line-of-sight path.

In this subroutine:

Call inputs:

d distance from transmit site at which diffraction attenuation is to be determined.

Prop_type

&prop array *prop* with array elements:

propa_type

&propa array *propa* with array elements:

defines private, or local, arguments:

prop_zgnd an array containing values of the *zgnd* surface transfer impedance, with elements:

prop.zgndreal, the real, (resistive) component of the surface transfer impedance; *prop.zgndimag*; the imaginary, (reactive) component.

wd1	1/wd, the inverse of the weighting factor wd
xd1	dla added to a curvature adjustment equal to tha/gme.
qk	the product of the effective height of the transmit receive terminal multiplied by the receive terminal effective height, less the product of the transmit receive terminal height AGL multiplied by the receive terminal height AGL; later set to be equal to 1/|zgnd|.
aht	represents the attenuation from both F(x) functions and the C(x) function included in the rounded earth attenuation equation for *ar*.
xht	sum of the values for x_1 and x_2
$x_{1,2,3,4}$	approximations of Airy functions; used in calculating G(x) and F(x) rounded edge functions.
a	the effective earth's radii $a_{0,1}$ for the terrain between the antennas and their horizons, preset to be equal to $dl^2/2he$ (then to *ds/th)* for each terminal site; also defined as $a_{3,4}$ for the effective earth's radius for the terrain between horizons.
q	utility variable; used for temporary holding of a value.
pk	absolute value of K; stated as |K| in [Alg. 6.6].
ds	the difference, in meters, between prop.dist, the total path distance, and dla, the sum of the horizon distances. It is calculated twice; the first time it is limited to be 0.0 or greater to avoid calculation errors in the geometric distance calculations. The second time it is calculated, it equals zero for one obstruction, is a positive value for multiple obstructions, and is a negative value for line-of-sight.
th	the sum of the transmit and receive take-off angles to the peaks of the highest visible obstructions.
wa	equal to $(a * wn)^{1/3}$; calculated twice, once for each terminal.
ar	attenuation due to rounded edge diffraction
wd	the weighting factor for knife edge versus rounded edge
adiffv2	attenuation due to diffraction, the output value
toh	height of transmit obstruction above t-r line, in meters AMSL
toho	height of transmit obstruction above t-rcvr obstruction line, in meters
roh	height of receive obstruction above t-r line, in meters AMSL
roho	height of receive obstruction above tx obstruction-r line, in meters
dto	distance from transmitter to obstruction peak for one obstruction, in m.
dto1	distance from transmitter to 1st obstruction peak for 2 obstructions, m.
dro	distance from receiver to obstruction peak for one obstruction, in m.
dro2	distance from receiver to 2nd obstruction peak for 2 obstructions, m.
dtro	distance from transmitter to receive obstruction top, in meters
drto	distance from receiver to transmitter obstruction top, in meters
dtr	distance from transmitter to receiver, in meters

dhh	distance from obstruction peak to obstruction peak, in meters, case 1
dhh1	distance from obstruction peak to obstruction peak, in meters, case 2
dhh2	distance from obstruction peak to obstruction peak, in meters, case 3
rd	ground distance from receive site to obstruction peak, = prop.dl[1]; meters
dhec	dh extended constant for dh beyond the line-of-sight range, equal to 2.73.
dfdh	delta h (dh) for diffraction range. This can be preset for test purposes by an input value to point to point set at a value above 1.

This subroutine:

Uses *d*, prop_type, propa_type, and other information in arrays prop and propa in order to calculate the attenuation due to diffraction over one or two obstacles at a distance d using a convex combination of smooth earth diffraction and double knife-edge diffraction.

b. First checks if the optional diffraction range delta-h (terrain height irregularity coefficient) is set equal to or greater than 1 in the point_to_point call input coefficients. If set to greater than zero, the optional value replaces the existing calculated value of delta-h stored in prop.dhd.

Line (new):

```
dfdh=prop.dh;
if (((int) (prop.dhd))>0)
dfdh=prop.dhd;
```

In the first run, with d = zero, coefficients for a two-peak, four radii rounded earth computation are prepared.

c. An **if** statement is initiated; if *d* is equal to zero, coefficients *wd1, xd1, ql, aht* and *xht* will be determined.

I. *q* is set to be equal to prop.hg[0] times prop.hg[1], the product of the transmit antenna height above ground level, multipled by the receive antenna height above ground level; units in meters, so output is in meters squared.

II. *qk* is set to be equal to the product of the effective height of the transmit antenna, prop.rch[0], multipled by the effective height of the receive antenna, prop.rch[1], less the value of *q* from step a.; output is in square meters.

III. *dhec* , the terrain height, extended, constant, is preset to be equal to 2.73.

Line 225:

```
if (d = = 0)
        q=prop.hg[0]*prop.hg[1];
        qk=prop.rch[0]*prop.rch[1]-q;
        dhec=2.73;
```

d. A second **if** statement is initiated within the first; if prop.mdp, the mode of the propagation model, is less than zero, indicating operation in the point-to-point mode, the value of q is increased by 10.

Line 230:
```
if (prop.mdp<0.0)
  q+=10.0;
```

e. The argument *wd1*, the inverse of the weighting factor *wd* used to weigh between knife edge and rounded edge diffraction, is set to be equal to the square root of (1 + qk/q).

Line 233:
```
wd1=sqrt(1.0+qk/q);
```

f. *xd1* is set to be equal to propa.dla + propa.tha/prop.gme.

Line 234:
```
xd1=propa.dla+propa.tha/prop.gme;
```

g. In steps 5 and 6, the Δh(s), the terrain irregularity factor Δh, determined at a specified distance s, and σn(s), the standard deviation of Δh(s), is determined. The value held by *q* is reset to be equal to the terrain irregularity parameter, dh (a.k.a. delta h or Δh), multiplied by the distance compensation term (1.0-0.8*exp(-propa.dlsa/50,000)), using a formula derived from Alg. (3.9). See subroutine ***qlrps*** step 23, for the derivation. At this point, the value of *q* represents Δh(s).

Line 235:
```
q=(1.0-0.8*exp(-propa.dlsa/50e3))*prop.dh;
```

h. *q* is then further modified by setting it to be equal to the value of *q* obtained on line 235 in step 5 above, multiplied by 0.78*exp(-pow(q/16.0,0.25). At this point, the value of *q* represents the standard deviation of the terrain irregularity factor at a distance "s"; $\sigma_h(s)$.

This step utilizes the formula:

$$\sigma_h(s) = 0.78\ \Delta h(s) \exp\left[-\left(\Delta h(s)/H\right)^{1/4}\right] \quad \text{with } H = 16 \text{ meters.''} \qquad \text{[Alg. 3.10]}$$

This formula, is found in the Algorithm, shows the relationship between Δh and the terrain roughness factor σ_h used in Tech Note 101. Here it is used to convert the value stored in *q* from the value for Δh(s) to the value for $\sigma_h(s)$.

Line: 236:
```
q*=0.78*exp(-pow(q/16.0,0.25));
```

i. In this step, ITS-67 and "The Algorithm" both describe Afo as a "clutter factor". In ITS-67 the equation is found in the form:

$$A_{fo} = 5\log_{10}[\ 1+h_{g0}h_{g1}f_{MHz}\sigma_h(d_{Ls})10^{-5}]\ \text{dB or 15 dB, whichever is smaller.}$$

[ITS-67 3.38

The value of *afo*, Attenuation From Other, is set to be equal to the lesser of:

I. 15 dB

II. (2.171*log(1.0+4.77e-4*prop.hg[0]*prop.hg[1]*prop.wn*q));

[Alg. 4.10] and [ITS-67 3.38c]

Note: In ITS-67, the log is identified as a common logarithm, i.e. $5*\log_{10}$. In the ITWOM, the log10 function is used to replace the original substitution of .4343*log (.4343ln) for log10.

Where:

hg[0] is the transmit antenna height above ground in meters;
hg[1] is the receive antenna height above ground in meters;
prop.wn is the wave number, (equal to freq. in MHz/47.7)
q is currently equal to the value of $\sigma_h(s)$, the terrain roughness factor at a specified distance "s", (i.e., with distance correction).

So in the ITM we find:

afo=mymin(15.0,5.0*log10(1.0+4.77e-4*prop.hg[0]*prop.hg[1]*prop.wn*q));

But this has been removed from the ITWOM. This empirical formula:

I. Has no theoretical basis described in Tech Note 101, nor in ITS-67 where it first appears.

II. Utilizes the same inputs used by Shumate's approximations.

On this basis, we suspect that it was an early, empirical derivation that had somewhat similar results to Shumate's clutter loss approximations, and is replaced by Shumate's deterministic-based clutter loss approximations in ***adiff2***. Therefore, it has been removed from consideration in this subroutine.

j. The value of *qk* is reset to be equal to 1/(absolute value of *prop_zgnd*).

prop_zgnd is a complex double, representing the earth's surface transfer impedance with two elements; a real element, the resistance value, and an "imaginary" value, the reactance, which describes the phase mismatch between the voltage and the current, in terms of a capacitive value (current peak leads voltage peak) or a inductive value (current peak lags behind voltage peak). *Note that while the Algorithm notes that zgnd is a complex (a +ib) double, the absolute function, used on a complex number, is used to create a "real" only value for qk.*

Line 238: qk=1.0/abs(prop_zgnd);

k. The value of *aht ,* rounded edge attenuation height functions, is set to be equal to 20.0 to incorporate the default approximation value of 20 dB for the function $C(K_o)$ in the rounded obstruction attenuation equation.

Line 239: aht=20.0; [Alg. 6.7]

l. The value of x*ht* is preset to be equal to 0.0. This argument will later hold the sum of the values for x_1 and x_2.

Line 240: xht=0.0;

m. Here the original alos subroutine's ***for*** loop always calculates both of the two F(ht) functions, the height-related rounding functions added to each side of the knife edge to approximate a rounded edge. The math used to implement this function will cause an error if it attempts to calculate the receive side rounding loss when the receive point is at the peak of the obstruction. With dl[1] is zero or approaches zero, a computes to be zero, wa computes to be zero, and pk reports out an absurdly large number, or "inf" when it attempts to divide by wa=0.0; this causes the computation of q to fail or report out zero, causing wild results for xht, fht and aht. However, the rounded earth function may still need the results of this computation. So we have split the for loop into two separate runs, in order to sort the various results separately to the later functions as needed.

n. First, the coefficients for the rounding function on the transmitter side of the knife edge are computed.

   a. The value of *a* is set to be equal to:

$$0.5*(\text{prop.dl}[0])^2/\text{prop.he}[0]; \qquad \text{[Alg. 4.15]}$$

where:

prop.dl[0] is the distance from the transmit site to the horizon
prop.he[0] is the effective transmitter height, in meters.

Because subroutine qlrpfl significantly changes the values of the effective heights when the path extends into the diffraction range, in other locations we now use prop.rch[] , the true radiation center height with respect to mean sea level, instead of prop.he[].

prop.rch[0] is the transmitter antenna height, RC-AMSL, in meters.

The argument *a* represents the effective earth's radii $a_{0,1}$ for the terrain between the antennas and their horizons. Here, in the initial coefficient-setting run, (d=0), it is set to be equal to $dl^2/2he$ for each terminal site. First it represents a_0 for dl[0], the transmitter site to horizon (or highest obstruction) distance.

Line 245: a=0.5*(prop.dl[0]*prop.dl[0])/prop.he[0];

   b. The value of *wa* is set to be equal to $(a*prop.wn)^{1/3}$ [Alg. 4.16]

Where:

prop.wn is the wave number, = (frequency in MHz/47.7)

Line 246: wa=pow(a*prop.wn,THIRD);

II. The value of *pk* is set to be equal to *qk/wa*. [Alg. 4.17]

Where:

qk was determined in step 8.

wa was determined in the last step.

Line 247: pk=qk/wa;

1. The value of *q* is again reset, this time to be equal to: ((1.607-pk)*151.0*wa*prop.dl[0]/*a*; [Alg. 4.18 and 6.2]

Line 248: q=(1.607-pk)*151.0*wa*prop.dl[0]/a;

2. The value of *xht* is increased by adding the value of *q*.[Alg. 4.19 –height - gain]

Line 249: xht+=q;

3. Subroutine ***fht*** is called with inputs *(q,pk).* Subroutine ***fht*** then returns *fhtv,* the height-gain over a smooth spherical earth, i.e. a coefficient of the rounding loss, for use with the three-radii method. The value of *aht* is increased by adding the value returned by ***fht.*** [Alg. 4.20]

Line 250: aht+=fht(q,pk);

o. In the ITWOM, the for loop found in the ITM has been removed; the coefficient functions remain; a new ***if*** statement is initiated, so that if prop.dl[1] is zero, indicating that the receive point is atop an obstruction, or if the grazing angle, prop.the[1] of the receive site is greater than 0.2 radians, then xht will be doubled, and aht will have the value of fht calculated for the transmitter side doubled. The receive side then is calculated as if it were a mirror image of the transmitter side, and a later ***if*** statement cuts the resulting attenuation in half.

Line (new):
```
If ((int(prop.dl[1])==0.0)  || (prop.the[1]>0.2))
{
        xht+=xht;
        aht+=(aht-20.0);
}
```

p. An else statement then provides an alternative path to the if statement above; if the receive grazing angle is below 0.2 radians, and the receive site is not atop the obstruction, then the receive site coefficients are computed the same way the transmitter coefficients were calculated above.

I. The value of *a* is set to be equal to:

$$a = (\text{prop.dl}[1])^2/(2*\text{prop.he}[1]) \qquad \text{[Alg. 4.15]}$$

Where:

prop.dl[1] is the distance from the receive site to the receive horizon (obstruction peak).

prop.he[1] is the effective receiver height, in meters.

In other areas, we find that *prop.he[1]* is replaced by:

prop.rch[1], the receive antenna height RC-AMSL in meters.

For the receive side calculation, *a* represents the effective earth's radii for the terrain between the receive antenna and its horizon, a_1 for dl[1].

II. The value of *wa* is set to be equal to $(a*prop.wn)^{1/3}$ [Alg. 4.16]

Where:

a was determined above.

prop.wn is the wave number, = (frequency in MHz/47.7)

III. The value of *pk* is set to be equal to *qk/wa.* [Alg. 4.17]

Where:

qk was determined in step 8.

wa was determined in the last step, 11(h).

IV. The value of *q*, here to represent $x_{1,2}$, is reset to be equal to:

q =(1.607-pk)*151.0*wa*prop.dl[1]/*a*; [Alg. 4.18 and 6.2]

V. The value of *xht*, which exists to hold the sum of x_1 and x_2, is increased by adding the value of *q.* [Alg. 4.19]

VI. Subroutine ***fht*** is called with inputs *(q,pk).* Subroutine ***fht*** then returns *fhtv,* the height-gain over a smooth spherical earth for use with the three-radii method. The value of *aht* is increased by adding the value returned by ***fht.***

Line(new):

```
a=0.5*(prop.dl[1]*prop.dl[1])/prop.he[1];
wa=pow(a*prop.wn,THIRD);
pk=qk/wa;
q=(1.607-pk)*151.0*wa*prop.dl[1]/a;
xht=q;
aht+=fht(q,pk);
```

The ***if*** or ***else*** statement completes.

q. The initial diffraction constants for a two obstruction, four radii rounded earth computation have been calculated, and the subroutine proceeds to report out diffraction attenuation = 0.0, indicating a coefficient setup run has completed. The argument *adiffv2* is set equal to zero.

Line 253: adiffv2=0.0;
}

r. The **if** statement on line 225 has a matching **else** statement on line 256. Therefore, if the input value *d* is not equal to zero, it indicates a second or later run is in progress, following the required first run with d=0 to set the coefficients.

Line 256: else
{

s. *th* is set to be equal to *propa.tha* + *d*prop.gme*; [Alg. 4.12]

Where:

propa.tha, the total bending angle, set in ***lrprop;*** in radians (after correction).

d is the distance at which the attenuation is to be calculated.

gme is the earth's effective radius.

Line 258: th=propa.tha+d*prop.gme;

Here, in the ITM, *ds* is set to be equal to *d* − *propa.dla*;

Where:

d is the distance at which the attenuation is to be calculated.

propa.dla is the sum of the two horizon distances, all in meters.

NOTE: The argument *ds* is the difference, in meters, between prop.dist, the total path distance, and dla, the sum of the horizon distances. It is calculated twice; the first time, it is limited to be a positive number only to avoid calculation errors in the geometric distance calculations below. After the second calculation, it equals zero for one obstruction; positive for multiple obstructions; negative for line-of-sight. In the original ITM, the calculations that follow produce absurd and non-a-number or infinity results if ds = 0.0; this, combined with the use of two other geometrical approximations for *a* that cause failure as the receive point approaches an obstacle peak, explains why the original ITM version of this subroutine cannot compute diffraction over a single obstacle, only multiple obstacles, and not when the receive site is near an obstacle. The changes for the ITWOM include:

Line 259: ds=mymax (d-propa.dla,0.0);

38. Next, the attenuation due to rounded earth for two obstructions is calculated.
    a. *a* is set to equal ***ds/th,*** in meters/radian.

Here, the argument *a* represents the effective earth's radii $a_{3,4}$ for the terrain between the two horizons and/or highest visible obstructions from the transmit and receive sites. In this main computation run (d >0), *a* is set to be equal to *ds/th* to represent $a_{3,4}$, the effective earth's radii between the obstructions. The computational difficulty comes in where *ds* =0, which happens when there is only one mutual horizon or obstruction. If ds =0.0, (see above NOTE 2.) this calculation returns "inf", indicating that it attempted to compute infinity, causing failure in both the ar and adiffv2=aknife computations. This problem has been addressed by separating out the computations for a single obstruction, for when ds=0, from those for two obstructions.

Line 263: a=ds/th;

39. *wa* is set to be equal to $(a*prop.wn)^{1/3}$ [Alg. 4.16]

Line 264: wa=pow(a*prop.wn,THIRD);

40. *pk* is set to equal the value of *qk*/*wa.*

Line 265: pk=qk/wa; [Alg 4.17]

The computations for ***adiff***2 have been treated differently than in the other heavily updated subroutines identified with a 2 after the original subroutine name; here the ***adiff2*** subroutine has been modified to work better with both the old ITM and new ITWOM implementations.

NOTE: In the original implementation of the ITM, here is found one of the biggest Achilles heels in the Longley-Rice implementation. The problem is caused by the substitution of a shortcut formula for determining two-ray and diffraction path phase difference lengths instead of more rigorously computing the accurate geometrical coefficients. This geometrical math approximation causes the diffraction equations to work well only at a distance from an obstruction, contributing to the already extant limitation that the diffraction computations were only good for receive look-up angles of 0.2 radians or less. This bad approximation is sanctioned in Tech Note 101 starting in section 5.

Therefore, the ITWOM uses a more rigorous geometrical computation of the phase path length difference v. Tech Note 101, section 5.2 paragraph 1, states that the equation for path length difference between a reflected path and a direct path is:

$$\Delta r \equiv r_1 + r_2 - r_0$$

Where $r_1 + r_2$ are the lengths of the two reflected path legs, and r_0 is the length of the direct path between the transmitter and the receiver. This is simple, straightforward, and correct.

But then in 7.1, where the same equation is used, it states that:

$$\Delta r = r_1 + r_2 - r_0 = \theta^2 d_1 d_2 / 2d$$

Where: θ is th, the total bending angle

d_1 is the ground distance between the transmit site and the peak of the knife edge

d_2 is the ground distance between the receive site and the peak of the knife edge

d is the total path distance.

The second version, $\theta^2 d_1 d_2/2d$, saves time in both manual and computer implementation; but it is a bad approximation to use. To start with, it assumes that the knife edge is approximately half way between the transmitter and receiver, which allows the two take-off angles to be summed as angle θ, (theta, or th). This alone means that the computation is incorrect when d_1 is not approximately equal to d_2, which means as the ITM computes a terrain path, when the location of the receive site is just past an obstruction, the computation of Δr has significant error. This is one of the reasons why the E3 or kwx=3 error code is generated in this case; causing a significant error, and problem in the implementations, including with the computation of DTV reception in the FCC SHIVA and SHIVERA implementations.

The second major problem is that the implementation uses the argument ds to compute the d_2 in $\theta^2 d_1 d_2/2d$, in a two knife-edge solution. The argument ds is set to be equal to the path distance *d* less *dla*, where *dla* is the sum of the two horizon distances, and represents the ground distance between two obstruction peaks:

ds= d - prop.dla, where dla = prop.dl[0] + prop.dl[1]

But when there is only one obstruction, dla is equal to d. The argument ds then computes to be equal to zero, and the Δr argument, and subsequent *v2* argument, computes to be infinity, which the program reports as "inf", and fails. Where the receive horizon distance is short, i.e. when the receiver is near the obstruction peak, ds approaches infinity, causing the computation to overload with a too-big impossible result, forcing a "nan" output.

Therefore, the ITM implementation cannot compute the diffraction for a single obstruction, or near an obstruction. The same problem occurs in the computation of the rounded edge. The subroutine ***adiff*** then fails to compute for a single obstruction, reporting out "inf" for a single obstruction, and "nan", or not-a-number, for receiver horizon distances near an obstruction.

In ***adiff2***, the computations have had this approximation excised; the distances r_1, r_2, and r_0 are computed using straightforward plane geometry and trigonometry.

41. In the ITWOM, it is necessary to compute the single knife edge geometrical coefficients. The first to determine is the obstruction height above a line between the transmitter antenna and the receiver antenna for a single obstacle. This working height is determined by taking the height AMSL of a point where a direct line between the transmitter and receiver antennas crosses the single obstacle centerline, at distance dl[0], and subtracting this height from the obstruction height, prop.hht, to obtain the correct effective obstruction heights, *toh*, and *roh*, for the v computation. Similar computations are required for the two-obstruction calculations. The term [(prop.rch[0]-prop.dl[0]*((prop.rch[0]-prop.rch[1])/prop.dist)] is a statement of a straight line formula of the form y= a + mx, solved for a height at the obstruction location, where the transmitter height he[0] is the zero-crossing constant a, the term (prop.rch[1]-prop.rch[0])/propa.dist) is the slope m, and prop.dl[0] is the distance x.

For two obstructions, each obstruction will have its attenuation computed as if it were a stand-along obstruction, with the receiver obstruction first treated as a receive site for a single obstacle loss calculation, and then the first obstacle treated as a secondary transmitting site for a second "single obstacle" calculation. The two losses are then added together. The geometrical distances from the transmitter antenna to the second obstruction peak, *dtro*, and from the first obstruction peak to the receiver, *drto*, must be calculated, as well as *toho*, the transmitter obstruction height above a line from the transmitter to the second obstruction peak, and *roho*, the receive obstruction height above a line from the first obstruction peak to the receiver.

An approximation is used, in that the obstruction heights above the lines are computed for vertical lines between the obstruction peak and the direct line, not lines perpendicular to the direct line. This can be addressed later, in the fine-tuning of the ITWOM for databases with intervals smaller than 3-arc-seconds. In most cases, the grazing angles involved will be less than 3 degrees, and the resulting error is insignificant.

42. In the ITWOM, toh, toho, roh, roho, dto, dto1, dro, dro2, dtro, drto, dtr, dhh, dhh1, dhh2, and dhec are declared as static double.

43. The argument toh, the height of the transmitter horizon obstruction above a relatively horizontal line drawn from the transmitter antenna to the receive antenna is determined. It is computed using simple geometry and a straight line formula (this is an approximation; a higher precision computation would use a line perpendicular to the direct line):

    Prop.rch[1]-prop.rch[0]/prop.dist is the difference between the receive site rc-amsl and transmitter site rc-amsl, divided by the distance between them. It is

the slope of the direct line between the transmitter and receiver in a flat-earth scenario. Multiplying this slope by the distance between the transmitter to the obstacle, prop.dl[0], and adding to it the transmit antenna RC-AMSL height, gives us the height on the obstacle's vertical centerline, at which a line from the transmit antenna to the receive antenna would cross. This is then subtracted from the obstacle height AMSL to obtain toh, the height of the obstacle between a direct line from the transmitter antenna to the receive antenna, over flat earth. This would apply to either a single-obstacle or two-obstacle scenario.

Line(new): toh= prop.hht-(prop.rch[0]-prop.dl[0]*((prop.rch[1]-prop.rch[0])/prop.dist))

44. Then compute roh, the receive obstruction height above a direct line between the transmitter and the receiver for a two-obstacle scenario, in a similar manner.

Line(new): roh= prop.hhr-(prop.rch[0]-(prop.dist-prop.dl[1])*(prop.rch[1]-prop.rch[0])/prop.dist));

45. Then compute toho, the transmitter obstruction height above a line between the transmitter and the receive obstruction peak.

Line(new): toho= prop.hht-(prop.rch[0]-(prop.dl[0]+ds)*((prop.hhr-prop.rch[0])/(prop.dist-prop.dl[1])))

46. Then compute roho, the receive obstruction height above a direct line between the transmitter obstruction and the receiver.

Line(new): roho= prop.hhr-(prop.hht-ds*(prop.hhr-prop.rch[0])/(prop.dist-prop.dl[0])));

47. Now calculate *dto*, the path distance from the transmitter to the transmitter obstruction peak for a single obstacle, *dto1*, the path distance from the transmitter to the transmitter obstruction peak for the first of two obstacles, *dtro*, the path distance from the transmitter to the receiver obstruction peak, *drto*, the path distance from the receiver to the transmitter obstruction peak, *dro*, the path distance from the receiver to the receiver obstruction peak for a single obstacle, and *dro2*, the path distance from the receiver to the receiver obstruction peak for the second of two obstacles, using basic geometry:

Line(new): dto=sqrt(prop.dl[0]*prop.dl[0]+toh*toh);
Line(new): dto1=sqrt(prop.dl[0]*prop.dl[0]+toho*toho);
Line(new): dtro=sqrt((prop.dl[0]+ds)*(prop.dl[0]+ds)+prop.hhr*prop.hhr);
Line(new): drto=sqrt((prop.dl[1]+ds)*(prop.dl[1]+ds)+prop.hht*prop.hht);
Line(new): dro=sqrt(prop.dl[1]*prop.dl[1]+roh*roh);
Line(new): dro2=sqrt(prop.dl[1]*prop.dl[1]+roho*roho);

Note that the height difference, as well as the ground path distance, between the points, is considered in the above calculations.

48. Next we determine the distances for the same paths, less the foliage height prop.cch, atop the obstructions.

Line(new): dtof=sqrt(prop.dl[0]*prop.dl[0]+(toh-prop.cch)*(toh-prop.cch));
Line(new): dto1f=sqrt(prop.dl[0]*prop.dl[0]+(toho-prop.cch)*(toho-prop.cch));
Line(new): drof=sqrt(prop.dl[1]*prop.dl[1]+(roh-prop.cch)*(roh-prop.cch));
Line(new): dro2f=sqrt(prop.dl[1]*prop.dl[1]+roho-prop.cch*roho-prop.cch));

49. For each of these distances, a second line is then added that adds the additional distance between the points occurring due to the effective curvature of the earth by adding prop.gme*(related ground distance). An additional fine tuning improvement in the professional version will add compensation for the average height (AMSL) of the two points, by adding the average height AMSL of the two points to prop.gme, before multiplying by the related ground distance.

50. Calculate the distance of the line from the transmitter antenna to the receive antenna, taking into consideration the height difference between the transmitter and the receiver:

Line(new): dtr=sqrt(prop.dist*prop.dist+(prop.rch[0]-prop.rch[1])*(prop.rch[0]-prop.rch[1]));

51. Calculate the actual path distance, for two scenarios, between the obstruction peaks. This has to reference against (1) for dhh1, the vertical height of the 1st obstacle with reference to a line between the transmit antenna and the 2nd obstacle peak, and then (2) for dhh2, against the vertical height of the 2nd obstacle with reference to a line between the 1st obstacle peak and the receive antenna. This is a Pythagorean triangle calculation, $x = (y^2 + z^2)^{0.5}$ where x represents the actual path distance, y is the ground path distance between the obstacles, and z is the height difference between the two obstacles. The ground path distance is the total path length, prop.dist, less the sum of the transmit and receive horizon distances, prop.dla. These are used for the knife edge computations in the ITWOM.

52. Calculate the actual path distance from the 1st obstruction peak to the 2nd obstacle peak, dhh1, using the horizontal path distance from the 1st obstruction peak to the 2nd obstacle peak, (the total path distance, prop.dist, less the sum of the horizon distances, prop.dla), and the height difference between the 1st obstruction peak and a line between the transmitter and the 2nd obstruction

peak. Toho is the height of the transmit horizon obstruction above a line between the transmit antenna and the 2nd obstacle peak line. Use for calculating knife-edge diffraction over a transmitter horizon obstacle or single-obstruction scenario.

53. Calculate the actual path distance from the 1st obstacle peak to the 2nd obstacle peak, dhh2, using the horizontal path distance from the 1st obstruction peak to the 2nd obstacle peak, and the height difference between the 2nd obstruction peak and a line between the 1st obstruction peak and the receive antenna. Roho is the height of the receive horizon obstruction above a line between the 1st obstruction to the receive antenna. Use for calculating knife-edge diffraction over a receive horizon obstacle.

Line(new): dhh1=sqrt((prop.dist-propa.dla)*(prop.dist-propa.dla)+toho*toho);
Line(new): dhh2=sqrt((prop.dist-propa.dla)*(prop.dist-propa.dla)+roho*roho);

54. Here, *ds* is reset to be equal to $d - propa.dla$;
    Where:
    d is the distance at which the attenuation is to be calculated.
    propa.dla is the sum of the two horizon distances, all in meters.

NOTE: The argument *ds* is the difference, in meters, between prop.dist, the total path distance, and dla, the sum of the horizon distances. It is calculated twice; the first time, it is limited to be a positive number only to avoid calculation errors in the geometric distance calculations below. Here, after the second calculation, it equals zero for one obstruction; positive for multiple obstructions; negative for line-of-sight. In the original ITM, the calculations that follow produce absurd and non-a-number or infinity results if ds = 0.0; this, combined with the use of two other geometrical approximations for *a* that cause failure as the receive point approaches an obstacle peak, explains why the original ITM version of this subroutine cannot compute diffraction over a single obstacle, only multiple obstacles, and not when the receive site is near an obstacle.

Line 259: ds=d-propa.dla;

Now we have the coefficients to compute v accurately, even near obstructions.

55. Coefficients are then set up for use in subroutine ***saalos***, to calculate clutter loss past the peak of the receive horizon obstacle peak:
    a. prop.tgh, the transmitter ground height, is set to prop.cch +1.0 meter; so subroutine ***saalos*** can treat the signal diffracting over the knife edge peak as if it is an effective secondary transmitter site atop the obstacle with respect to the receiver. This is done by adding a height 1 meter above the clutter canopy height.

b. The argument taag is set to be equal to the height of the secondary transmitter site, the receive horizon obstacle peak height, prop.hhr, which for a single obstacle will also be the same as the transmitter horizon obstacle peak height.
c. distance *rd*, here to be the distance from the mutual horizon obstacle peak to the receive antenna, is set to be equal to the receiver horizon distance, prop.dl[1].

Line (new)

```
prop.tgh=prop.cch+1.0;
prop.taag=hht;
rd=prop.dl[1];
```

The next steps implement the new ITWOM obstacle computations, starting with two peak scenarios, and with a corrected, true point-to-point version of the original Vogler two-obstacle computations as modified to an Epstein-Peterson methodology for the ITM. These are followed by the new ITWOM calculations for areas where the receive site grazing angle is greater than 0.2 radians, and then for single obstacle scenarios.

56. The argument ds can now be used to tell the difference between one and more obstructions, by using the integer value, the value rounded down to the nearest integer value, of ds. If the integer value of ds, the difference between the path distance and the sum of the two horizon distances, is zero, one obstruction has been found. If ds is greater than zero, two obstructions have been identified. An if statement is implemented to select for the case when two obstructions have been found.

Line (new):

```
If ((int(ds)>0.0)
{
```

57. A second, embedded if function is initiated to select for the case where the receive site is at least one interval past the peak of the second obstacle. If the receive antenna to receive horizon obstacle path is greater than zero, then:

Line (new)

```
If(int(prop.dl[1])>0.0)
{
```

58. In the ITM, for two obstacles, *q*, a utility variable, is reset to be equal to:

$$q = (1.607-pk) * 151.0 * wa * th + xht \qquad \text{[Alg. 4.18 and 6.2]}$$

and represents the distances x_3 and x_4 found in [ITS-67 3.37]. The distances x_3 and x_4 are defined in ITS-67 as:

$$x_3 = (B_3(a_3)^{-2/3}(ds)/1000)+x_1+x_2 \text{ km , where ds is in meters.} \qquad \text{[ITS-67 3.31a]}$$

$x_4 = (B_4(a_4)^{-2/3}(ds)/1000)+x_1+x_2$ km , where ds is in meters. [ITS-67 3.31b]

where:

$x_3 = (B_1(a_1)^{-2/3}(dl[0]\ /1000)$ km, where dl[0] is in meters. [ITS-67 3.30]

$x_4 = (B_2(a_2)^{-2/3}(dl[0]\ /1000)$ km, where dl[0] is in meters. [ITS-67 3.30]

and $B_{1,2,3,4}$ is defined as: $416.4f^{1/3}[1.607 - K_{h,v}(a_{1,2,3,4})]$. [ITS-67 3.32]

and the parameters $K_{h(a)}$ (K horizontal for *a*) and $K_{v(a)}$ (K vertical for *a*) are defined as:

$$K_{h(a)} = 0.36278(a*f)^{-1/3}[\varepsilon - 1]^2 + x^2]^{-1/4} \qquad \text{[ITS-67 3.33a]}$$

Or using wn instead of f: $K_{h,(a)} = 0.100032(a*wn)^{-1/3}[\varepsilon - 1]^2 + x^2]^{-1/4}$

$$K_{v(a)} = K_{h,(a)}[\varepsilon^2 + x^2]^{1/2} \qquad \text{[ITS-67 3.33b]}$$

Where this x is defined by [ITS-67 3.9a] as $x=377.36\ \sigma/wn$, $(18{,}000\ \sigma\ /f)$ and the ground constants σ and ε are provided in the input. The computer implementation uses σ and ε to compute the complex ground impedance, zgnd, which is used here instead of σ and ε.

So we can see that *wa* represents the $(a*wn)^{1/3}$ component, and that a combined $K_{h,v(a)}$ is represented by *pk*.

Line zzz: q=(1.607-pk)*151.0*wa*th+xht;

59. *ar* is the rounded earth attenuation, A_r, calculated as equal to:

$$0.05751 * q - 4.343 * \log(q) - aht \qquad \text{[Alg. 4.20, 4.23,4.24, and 4.25]}$$

which can be better stated using the specified $\log_{10}$, as:

$$A_r = 0.05751*q - 10*\log10(q) - aht$$

In the Algorithm, the author, George Hufford, stated that the calculation for A_r, the rounded edge attenuation, consisted of adding up four functions:

$$A_r = G(x_0) - F(x_1, K_1) - F(x_2, K_2) - C(K_0) \qquad \text{[Alg. 4.20]}$$

And that these four consisted of:

$$G(x) = 20\log(x^{-1/2}e^{x/A}) \qquad \text{[Alg. 4.23]}$$

where A is a dimensionless constant equal to 151.03;

$$F(x, K) = 20\log|(\pi/(2^{1/3}AB))^{1/2}Wi(to - (x/(2^{1/3}AB))^2)| \qquad \text{[Alg. 4.24]}$$

and:

$$C(K) = 20\log|1/2(\pi/(2^{1/3}AB))^{1/2}(2^{2/3}K^2t_0 - 1)Wi'(t_0)^2| \qquad \text{[Alg. 4.25]}$$

Which at first glance, does not seem to match up with the formula in the c++ code. It also does not make it clear whether the log functions are natural or common, nor does it state the units. But wait! There's more. If we look at ITS-67, we find that the formula for the G(x) function is:

$$G(x_{3,4}) = 0.05751\ x_{3,4} - 10\log10\ x_{3,4} \quad \text{dB} \qquad \text{[ITS-67 3.37]}$$

Aha! This looks more familiar! It states the units, in dB! And it clearly states the log function is log10, a common logarithm! And to those experience with knife-edge computations, it can be seen to be a modified knife-edge approximation equation.

To compare the two, it helps to know logarithm math. When working in logarithms, a multiplied function can be split into two, added, functions. So Hufford's $G(x) = 20\log(x^{-1/2}e^{x/A})$ becomes $G(x) = 20\log_{10}(x^{-1/2}) + 20\log_{10}(e^{x/A})$. The –1/2 power can cross over the common logarithm term, so $20\log_{10}(x^{-1/2})$ converts to $-(20/2)\log_{10}(x)$, or, simply, $-10\log_{10}(x)$. A natural logarithm result can be converted to a common logarithm result by multiplying the natural logarithm result by .4343, so $20\log_{10}(e^{x/A})$ converts to $(20*.4343)\ln(e^{x/A})$. An ln function is the inverse of an e function, so they cancel out, leaving (20*.4343)(x/A). A=151.03, so the result is: .4343(20x/151.03) = 0.05751x. So despite George's obfuscation and missing details, we see that:

$$G(x) = 20\log(x^{-1/2}e^{x/A}) = 0.05751(x) - 10\log_{10}(x) \qquad \text{[Alg. 4.23]}$$

And it is obvious that q in the code here represents $x_{1,2}$ in the formulas. Which except for the argument aht, is most of the formula used in the ITM. ITS-67 states that x is defined by [ITS 3.9a] to be $x = 18000\sigma/f$, where the frequency f is in MHz and the ground constant σ for average ground is 0.005 mho/m. (Table 2, p.8, ITS-67].

Next, we check aht. Either only a part of the full equation is used, or aht is tasked with providing all of the result of $-F(x_1, K_1) - F(x_2, K_2) - C(K_0)$. The argument aht was calculated during the first coefficient calculation run, where (d==0); it was first set to be equal to 20. The ITS-67 equivalent formula to [Alg. 4.20] is:

$$A_{r3,4} = G(x_{3,4}) - F(x_1) - F(x_2) - 20 \quad \text{dB} \qquad \text{[ITS-67 3.28]}$$

And therefore replaces $C(K_0)$ outright with 20. The Algorithm notes that "for those values of K in which we are interested it is a good approximation to say $C_1(K) = 20$ dB." So $C(K_0)$ is represented in aht, when aht is preset to 20.0.

As to $F(x_1)$ and $F(x_2)$, they are computed in two calls to subroutine ***fht*** with inputs (q, pk), also during the first coefficient setup run, and the two results are summed with the 20.0 $C(K_0)$ value preset in aht. So aht does contain the results of both F functions, and the C function. Also note that if the receive site look up angle is greater than 0.2 radians, the two peak calculation is treated as a knife-edge diffraction over two peaks, with attenuation on the receive side of an obstruction is then determined with Radiative Transfer Function approximations using the ***saalos*** subroutine.

Line 267: ar=0.05751*q-10*log10(q)-aht;

60. Next, the process of calculating wd, the weighting factor, wd, for knife edge vs rounded edge, is completed using wd1 and xd1.

    b. *q* is reset to be equal to:

    (*wd1*+*xd1*/*d*)*mymin(((1.0-0.8*exp(-*d*/50e3))**prop.dh***prop.wn*),6283.2) [Alg. 4.9]

Line 268: q=(wd1+xd1/d)*mymin(((1.0-0.8*exp(-d/50e3))*prop.dh*prop.wn),6283.2);

61. wd, the edge weighting factor, is set to be equal to:
    (25.1/(25.1+sqrt(*q*))) [Alg. 4.9]

Line 269: wd=25.1/(25.1+sqrt(q));

In the original implementation of the ITM, only a two-obstacle scenario is considered. From ITS-67, section 3.2, " In this application the knife-edge attenuation is computed as though the radio path crossed two sharp, isolated ridges." So in the ITM:

62. *q* is reset to be equal to 0.0795775*prop.wn*ds*th*th; At this point, the value of *q* is a partial term that represents most of v^2, the internal wedge angle of a knife edge diffraction calculation.

To show this calculation, in ITS-67, we find v1,3 specified in 3.26a (with notation changed for c++ for clarity) as:

$$v_{1,3} = 1.2915*\theta_3*(f_{MHz}* dl[0]*(ds)/(d - dl[1]))^{1/2} \qquad \text{[ITS-67 3.26a-d]}$$

The frequency is stated in megahertz, and the distances in kilometers.

First we substitute to use the wave number, 47.71*wn= f_{MHz} :

$$v_{1,3} = 1.2915*\theta_3*(47.7*prop.wn*\ dl[0]*(ds)/(d - dl[1]))^{1/2}$$

Squaring both sides of the equation, we get:

$v^2_{1,3}$ = 1.2915*1.2915*47.7*th*th*prop.wn*prop.dl[0,1]*ds /(prop.dist-prop.dl[1,0]);

$v^2_{1,3}$ = 79.5775*prop.wn*th*th*ds*prop.dl[0,1]/(prop.dist-prop.dl[1,0]);

In the original equations, Δr and λ are in kilometers, as noted in TN101 7.1. To accommodate data entered in meters instead of kilometers, the constant is divided by 1000, resulting in:

$v^2_{1,3}$ = 0.0795775*prop.wn*th*th*ds*prop.dl[0,1]/(prop.dist-prop.dl[1,0]);

So q represents most of v^2. The rest was added in the next step, in the input calls to ***aknfe***, and the ITM reset the value of the utility variable q to be:

q=0.0795775*prop.wn*th*th*ds.

NOTE: Here we find a discrepancy. On this issue the Tech Note 101 methodology and the ITS-67 methodology and FORTRAN source code are in agreement. The original ITS-67 methodology, a four calculation (two calculations per obstruction) method, was changed and simplified for the ITM to be a two calculation methodology, but an error occurred that remains in the ITM code. The documentation in "The Algorithm" appears to be written as if George Hufford was aware of the discrepancy, and was avoiding any mention of it.

In the ITM FORTRAN source code in the Guide, and in the ITMDLL.cpp source code, we find q. The utility argument q at that point in the code represents v^2 with distances and heights in meters, and wn=f_{MHz}/47.7. The argument q is set to be equal to:

Q= 0.0795775*WN*DS*TH**2

in FORTRAN, and in the ITMDLL.cpp:

Q= 0.0795775*prop.wn*ds*pow(th,2.0);

The question is about the derivation of the constant 0.0795775. To derive the correct constant, we start with:

$$v = \pm\, 2*(\Delta r/\lambda)^{1/2} \qquad \text{[TN101 7.1a]}$$

Where Δr and λ are in kilometers.

Using the approximation stated on page 7-1 of TN 101 of:

$$\Delta r = \theta^2 * d_1 * d_2/2d$$

Then: $v = \pm 2*(\Delta r/\lambda)^{1/2} = \pm 2*((\theta^2 * d_1 * d_2/2d)*1/\lambda)^{1/2}$

The wavelength, λ, in meters, is equal to speed of light in free space, C, or 2.799792e10 meters/sec, divided by the frequency, f in hertz, or f_{Hz}:

$$\lambda = C/f_{Hz} = 2.799792e10/f_{Hz}$$

The inverse of λ is: $= f_{Hz}/2.799792e10 = f_{MHz}/2.799792e2, = f_{MHz}/279.9792$ in meters.

Or: $1/\lambda = f_{Hz}/C = f_{MHz}/.2799792$, in units of 1/kilometer, which can be approximated as:

$$1/\lambda = f_{MHz}/.3 \text{ in units of 1/km.}$$

In terms of frequency in MHz and distances d_1, d_2 and d in kilometers, v can be approximated as:

$$v = \pm 2*((\theta^2 * d_1 * d_2/2d)*1/\lambda)^{1/2} = v = \pm 2*((\theta^2 * d_1 * d_2/2d)(f/.3))^{1/2}$$

$$v = \pm 2.583*\theta*(f*d_1*d_2/d)^{1/2} \qquad \text{[TN101 7.1b]}$$

Note that the 1/(2*.3) has been removed from the denominator inside the square root term brackets and included in the 2.583 constant, as $2.583 = 2*(1/.6)^{1/2}$.

Converting to the use of wn, where frequency in MHz = wn*47.7, it is restated as:

$$v = \pm 2.583*\theta*(47.7*wn*d_1*d_2/d)^{1/2}$$

And converting from the use of distances in kilometers to distances in meters:

$$v = \pm 2.583*\theta*(47.7*wn*d_1*d_2*1000*1000/1000*d)^{1/2}$$

$$v = \pm 2.583*(47.7/1000)^{1/2}*\theta*(wn*d_1*d_2/d)^{1/2}$$

$$v = \pm 2.583*(47.7/1000)^{1/2}*\theta*(wn*d_1*d_2/d)^{1/2}$$

$$v = \pm .564136*\theta*(wn*d_1*d_2/d)^{1/2}$$

The square of v, represented by q as should be used in the ITM, is then:

$$q = v^2 = \pm .3182491*\theta^2*wn*d_1*d_2/d \qquad \text{(xxx)}$$

But in the ITM, we find 0.0795775 instead of 0.3182491. 0.0795775 is one-fourth of 0.3182491, but since v is squared, it would have only an effect of the square root of 4, or a factor of 2, reducing the correct value of v by one-half. As a result, it would appear that ITM computations do not provide full value for knife edge attenuation, by a factor of $\log_{10}(2)$, or 0.3 of the correct value added to 12.953.

To solve the mystery, note that in the derivation from the ITS-67 source code and methodology stated above at step 29, we obtained a value for v of:

$$v_{1,3} = 1.2915*\theta_3*(f_{MHz}* dl[0]*(ds)/(d - dl[1]))^{1/2} \qquad \text{[ITS-67 3.26a-d]}$$

And note that the constant is 1.2915 instead of 2.583. 1.2915 is half of 2.583. In ITS-67 section 3-2, we find that the loss for each obstacle is calculated in two sections, creating a total of four loss values, two for each obstacle, that are added together to get the total diffraction attenuation over the two obstacles. Since each obstacle is calculated twice, the average of the two values for each obstacle is obtained by dividing each calculation by 2. So 1.2915 replaces 2.583 in [ITS-67 3.26a-d], and each of the four calculations provides half of the diffraction attenuation value for one of the two obstacles.

In the ITM, however, the calculation methodology is changed so that there are only two calculations, a single call to ***aknfe*** for each obstacle. This changes Vogler's original ITS-67 knife-edge methodology to be equal to the Epstein-Peterson methodology, for use in the ITM. When the modification occurred between the ITS-67 implementation (1968) and the ITM (early 1980's), the modifier forgot to change the constant 1.2915, used for two calculations per obstruction, to 2.583 for a single calculation per obstruction.

As verification, going back to the Fresnel-Kirchhoff theory on which Vogler's methodology is based, note that the square of v, i.e. argument v2, is equal to $4*\Delta r/\lambda$, as evident from [TN101 7.1]. Converting this equation to use the wave number instead of the wavelength, where the wave number, wn, for an electric field is: wn = $2\pi/\lambda$, so $1/\lambda = wn/2\pi$;

$$v2 = 4*\Delta r/\lambda = \Delta r(4/\lambda) = \Delta r(4*wn/2\pi) = \Delta r(2wn/\pi) = \Delta r(0.6365*wn)$$

The argument q, as shown above, then represents the partial $4/\lambda$ term of v2, and for use in the ITM, q should be equal to: 0.6365*prop.wn*(1/2) = 0.31825*prop.wn for a single diffraction edge calculation per obstruction, after including the ½ constant from the angle approximation calculation of Δr.

Note that while in the ITM the value of ds/2 is included in q, in the ITWOM, the full value of Δr *is included in the following call to aknfe, so in the ITWOM, where the* Δr *calculation has been changed to a more rigorous and accurate Pythagorean geometric calculation, q will be equal to: 0.6365*prop.wn.*

Line (changed): q=0.6365*prop.wn;

63. A tertiary embedded if statement is initiated; for the path from the second obstruction to the receive site, if the receive site grazing angle prop.the[1] (a.k.a. look-up angle) is greater than 0.2 radians, then a full two obstacle scenario using the ITM methodology, corrected and modified for true point-to-point calculation (eliminating the averaging line system), is calculated:

```
Line (new)          If(prop.the[1]>0.2)
                    {
```

64. Subroutine ***aknfe*** is called twice to calculate the diffraction loss due to a knife-edge at two separate peaks. In the ITM, the first time with input: (q*prop.dl[0]/(ds+prop.dl[0])), and the second time with input: (q*prop.dl[1]/(ds+prop.dl[1])).

    Note that the additions in the input complete the inputs to aknfe to represent v_1^2 and v_3^2. In each case, ***aknfe*** reports out *a*, the attenuation due to a single knife edge diffraction; the Fresnel integral (in decibels) as a function of the input, v^2.
    Adiffv2 is then temporarily set to equal the sum of the two outputs from ***aknfe***, the knife edge diffraction from two knife edges. [Alg. 4.14]

    In the ITM, this was accomplished using:

```
adiffv=aknfe(q*prop.dl[0]/(ds+prop.dl[0]))+aknfe(q*prop.dl[1]/(ds+prop.dl[1]));
```

But now we use:

65. The original ITM methodology for two obstructions, modified to remove the averaging line system and calculated using the actual transmitter and receiver horizon obstacle peaks, is invoked:

    a. Subroutine ***aknfe*** is called twice to calculate the diffraction loss due to a knife- edge peak on two separate obstacles, using a simplified version of the Vogler multiple-knife-edge methodology. The first call calculates the diffraction loss over the transmitter obstruction, using a path from the transmitter to the 2nd obstruction peak. The second call calculates the diffraction loss over the receive obstruction, using a path from the transmitter obstruction peak to the receiver;

    a. the first time with input: (2+q*abs(dto1+dhh1-dtro)),
    b. and the second time with input: (2+q*abs(dro2+dhh2-drto));

    Note that the additions in the input complete the inputs to ***aknfe*** to represent v_1^2 and v_3^2. In each case, ***aknfe*** reports out the attenuation due

to a single knife edge diffraction; the Fresnel integral (in decibels) as a function of the input, v^2.
Adiffv2 is then temporarily set to equal the sum of the two outputs from ***aknfe***, the knife edge diffraction from two knife edges. [Alg. 4.14]

Line 262: else
{
adiffv2=aknfe(2+q*abs(dto1+dhh1-dtro)) +aknfe(2+q*abs(dro2+dhh2-drto));

66. The calculation for *adiffv2* , here representing the total diffraction attenuation, is then completed; by being set to be equal to:
ar * *wd* + (1.0-*wd*) * *adiffv2* [Alg. 4.11] with afo removed.
where;
adiffv2, on the right side of the = sign, holds the value of the knife edge attenuation.
ar is the rounded earth attenuation

Line 270: adiffv2=ar*wd+(1.0-wd)*adiffv2;
}

From this full-length two-obstacle diffraction calculation scenario, we then work back toward the transmitter site, with the other diffraction calculation scenarios.

The next scenario considered is when there are two obstructions, but the receive site grazing angle, the look-up angle from the receiver to the receive site horizon obstacle (the tallest obstacle "visible" from the receive site, exceeds the theoretical limit of 0.2 radians stated in Tech Note 101, and which is programmed in the ITM to cause a kwx=3, a.k.a. EC3, level alarm, indicating that the value calculated cannot be trusted to be correct. Put another way, when the receive site is in the obstacle's "shadow area", too close to the obstacle for Fresnel-Kirchhof knife-edge theory to work correctly.

67. An ***else*** statement follows; if the receive site is too close to an obstruction, i.e. if the receive site grazing angle is equal to or greater than 0.2 radians, then:

68. Subroutine ***aknfe*** is called once to calculate the diffraction loss due to a knife-edge peak at the transmitter horizon that is followed by a separate receive site obstruction. The call calculates the diffraction loss over the transmitter obstruction, using a path from the transmitter to the 2nd obstruction peak (the receive site), with input: (q*abs(dto1+dhh1-dtro)).

Note that the additions in the input, within the absolute value statement, calculate Δr, the path length difference between the diffracted and a theoretical direct ray, and therefore allows the input to ***aknfe*** to represent v_1^2. Subroutine ***aknfe*** reports out the attenuation due to a single knife edge diffraction; the Fresnel integral (in

decibels) as a function of the input, v^2. *Adiffv2* then temporarily represents the sum of the output from ***aknfe*** for the transmitter horizon obstacle. [Alg. 4.14 modified]

Line 262: adiffv2=aknfe(q*abs(dto1+dhh1-dtro));

69. A fourth-level embedded if statement is initiated, so that if the receive site grazing angle is greater than 0.6 radians, (34.4 degrees) then the value of prop.tgh is reset to be equal to prop.cch-1.0. This allows ***saalos*** to treat the diffracted signal at the second peak as if it were a radiating from a secondary transmitting antenna at the clutter canopy height (set to 1 meter below) with the signal passing through the clutter on the side of the obstacle on its way down the side of the obstacle to the receiver.

```
Line    :       if(prop.the[1]>0.6)
                {
                        prop.tgh =prop.cch-1.0;
                }
```

70. An else statement follows the above if statement, so if the receive site grazing angle is between 0.2 and 0.6 radians, (11.5 to 34.4 degrees) then the transmitter ground height, prop.tgh is set to be equal to: the clutter height, prop.cch, plus the height of the receive horizon obstacle above mean sea level, prop.hhr, less the effective height of the receive antenna, plus the actual height of the ground at the receive site. This will allow subroutine saalos to treat the signal diffracted over the 2nd obstacle as a secondary transmit site, transmitting a signal from above the clutter canopy line through what starts as a clear air path toward a receive antenna in the valley below, a secondary line-of-sight clutter loss calculation.

```
Line: (new):    else
                {
                        prop.tgh=prop.cch+1.0;
                }
```

71. A following fourth-level embedded if statement is initiated; if the receive site grazing angle is less than 1.22 radians (69 degrees), distance *rd* is set to be equal to the receiver horizon distance. If the receiver antenna is at or below the clutter canopy, subroutine ***saalos*** is called with inputs (*rd,prop, propa*); to calculate the clutter loss. The output attenuation of ***saalos*** is added to adiffv2, which already holds the weighted diffraction attenuation to and including the 2nd obstruction peak. If the receive antenna is above the clutter canopy, standard knife edge diffraction loss is applied.
72. An else statement follows; if the receive site grazing angle exceeds 1.22 radians, the receive site is assumed to be deeply shadowed; directly behind a

tall building or steep cliff, and an empirically derived 20 dB of attenuation is assigned.

Line (new):

```
if(prop.the[1]<1.22)
{
        rd=prop.dl[1];
        if(prop.hg[1]<=prop.cch)
        {
                adiffv2+=saalos(rd, prop, propa);
        }
        else
        {
                q=0.6365*prop.wn;
                adiffv2=aknfe(q*abs(dto1+dhh1-dtro))
        +aknfe(q*abs(dro2+dhh2-drto));
        }
}
else
{
        adiffv2+=20.0;
}
}
}
}
```

The case of the receive site being on the peak of the second, receive horizon obstruction, is considered as a single obstacle computation ending at a receive site on a diffracted peak.

73. An else statement follows; so when there are more than 2 obstacles, and if the receive path length is not greater than zero, then the receive site is atop the 2nd obstacle.

74. The value of q is reset to be equal to the value of v2, the Fresnel diffraction internal wedge angle, for a single knife edge: = 0.6365*prop.wn∗abs(dto+dro-dtr);

Line(new): q=0.6365*prop.wn∗abs(dto+dro-dtr);

75. By working definition, if only one terrain point (for a database with 3 arc second or smaller terrain detail) is both the highest obstacle "visible" from the transmitter site and the receive site, then it is a knife-edge, equal to or less than one terrain database interval wide. If the transmit and receive terminal horizon obstacle peak terrain points have the same value, and are separated by as little as one terrain interval, then ds>0.0 and the two points are treated as separate obstacles, and the two-obstacle knife-edge vs. rounding calculation is

applied. So for this and the following single-obstacle scenarios, the first obstruction is considered to be a knife-edge obstruction, simplifying the calculation to consider only knife-edge attenuation for the first obstruction.

76. *adiffv2* is set to equal the attenuation from knife edge diffraction from a single obstacle peak to a receive site atop a 2nd obstacle peak. A call to ***aknfe*** with input *q*, computes the diffraction attenuation for the first obstacle. To this is added 5.8 dB of knife-edge peak diffraction attenuation for the 2nd obstacle, where v=0, taken from Figure 7.1, Tech Note 101, for a receive site atop a peak.

Line(new): adiffv2=aknfe(q)+5.8;
}

At this point, the calculation of diffraction over two obstacles, if two horizon obstacles were found in the path, has been completed. A new method involving clutter losses is used for when the receive grazing angle is greater than 0.2 radians, and a non-averaged direct modification of the original ITM methodology is used when the grazing angle is less than 0.2 radians.

The next ITWOM section deals with what Tech Note gives instructions for, but that the ITM could not do; directly address and compute the loss due to a single, large and/or knife-edge type obstacle.

An else statement provides an alternative path to the if statement, If (int(ds)>0), above. If ds is less than or equal to 0.0, then there is one obstacle or a line-of-sight path. Since this is the diffraction subroutine, it indicates a single obstruction. What follows is a new ITWOM path to compute the diffraction attenuation for a single obstacle, something that the ITM has never been able to do.

77. An if statement follows; for a single obstruction scenario where the receiver is at the peak, indicated as where the integer value of ds, is zero, then the attenuation is set to be the standard diffraction loss value for a peak; 5.8 dB, plus an arbitrary reduced value, one-quarter of the calculated clutter loss, saalos, as the peak of an obstruction suffers less clutter loss. By definition, the clutter canopy depth is less than the average, to none, at an obstacle peak, depending upon the nature of the obstacle. As more detailed terrain databases become available, this could be improved upon; when a building is recognized, the clutter loss value could be made to be zero, leaving only the diffraction loss.
78. An else statement follows, so when there is a single obstruction scenario and the receiver is past the peak, then:

79. An embedded if statement is initialized within the else statement. If the receiver grazing angle is greater than 0.2, then:

Line(new): if(prop.the[1]>0.2)

```
{
```

Argument ***adiffv2*** is set to be equal to the 5.8 dB of diffraction attenuation at an obstacle peak where v=0, taken from Figure 7.1, Tech Note 101.

Line 262: adiffv2=5.8;

80. A fourth-level embedded if statement is initiated, so that if the receive site grazing angle is greater than 0.6 radians, then the value of prop.tgh is lowered by 1.0 meter, resetting it to be equal to the clutter height. This allows ***saalos*** to treat the diffracted signal at the second peak as if it were a radiating from a secondary transmitting antenna at the clutter canopy height, with the signal passing through the clutter on the side of the obstacle on its way down the side of the obstacle to the receiver.

Line : if(prop.the[1]>0.6)

```
{
        prop.tgh–=1.0;
}
```

81. A following fourth-level embedded if statement is initiated; if the receive site grazing angle is more than 0.2 radians, and less than 1.22 radians:
    b. If the receiver antenna is at or below the clutter canopy height, a call to ***saalos*** calculates the clutter loss, which is added to the diffraction loss already stored in *adiffv2*.
    c. An else statement follows; if the receiver antenna is above the clutter canopy height, a standard knife edge calculation is used.
    d. An else statement follows; if *prop.the* is equal to or greater than 1.22 radians, then a deep shadow loss of 20 db is used.

Note that if the grazing angle is more than 1.22 radians, the receive point is treated as a bare cliff or tall building, and a 20 db deep shadow loss is added to *adiffv*.

Line (new): if(prop.the[1]<1.22)

```
{
        rd=prop.dl[1];
        if(prop.hg[1]<=prop.cch)
        {
                adiffv2+=saalos(rd, prop, propa);
        }
        else
        {
                adiffv2=aknfe(0.6365*prop.wn*abs(dto+dro-dtr));
        }
}
```

```
else
{
        adiffv2+=20.0;
}
```

Here we utilize an updated version of the "original" Tech Note 1 diffraction methodology for a single obstacle, based on the description in section 7 of the TN101, for a bare-top obstacle (mountain), which is not used in the ITM. For obstacles (hills and mountains) that have foliated tops (are not above the tree line) a completely new methodology is used.

82. An **else** statement then provides an alternative path to the **if** (prop.the[1]> 0.2) statement, so for a single obstruction between a transmitter site and a receive site, where the receive site grazing angle is equal to or less than 0.2 degrees, then:

Line(new):
```
else
{
```

83. One problem lies within the approximation used to compute *v2* in the ITM; it becomes less accurate when the receive terminal is near an obstruction, and fails entirely where there is a single obstruction, where dl[0]+dl[1]=d. Code has been added to ***hzns*** to output the obstruction heights to prop.hht and prop.hhr; with these, we can most accurately compute *dto* and *dro*, the path distance from each terminal to the transmitter and receiver obstruction peaks using the instructions in section 7 of TN101.

For a single knife edge, the "internal wedge angle" coefficient, v, is computed from:

$$v = 2(|\Delta r|/\lambda)^{1/2}$$

$$v2 = v^2 = 4|\Delta r|/\lambda$$

where : λ is the wavelength, and
Δr is the path length difference, which can be computed as:

$$\Delta r = r1+r2-r0 = dto+dro-dtr \qquad \text{[Tech Note 101, Section 7.1]}$$

Since we are using the wave number instead of wavelength, we must convert our formula:

$\lambda = c_{km/sec}/(1000*f_{MHz})$, and $wn*47.7=f_{MHz}$, so:

$\lambda = c_{km/sec}/(wn*47{,}700)$, and $1/\lambda = (wn*47{,}000)/c_{km/sec}$

where c is the speed of light, 299,792 kilometers/sec;

$1/\lambda = (wn*47{,}700)/\ 299{,}792$ km/sec = 0.1591wn

and since q will now represent v2, aka v^2, temporarily, for a single knife-edge obstacle:

$q=2^2*(|\Delta r|/\lambda)=(4*(0.1591*wn*abs(dto+dhh-dtr)=.6365*prop.wn*abs(dto+dhh-dtr)$

A more in-depth version of this page's information was discussed in the previous two-obstacle section.

Line(new):q=0.6365*prop.wn∗abs(dto+dhh-dtr);

84. An *if* statement is initiated. If the height of the obstacle is below the average tree line (based on the average tree line height of the Rocky Mountains), then:
85.

Line : if (prop.hht < 3400)
{

86. A completely new methodology is used; based on the primary signal being the signal scattered through the thin layer of foliage atop an obstruction (with a 0.6 absolute phase delay) against multipath created by the signal diffracting over the clutter canopy. (A variation on two-ray path).
87. The cancellation distance to the receiver, cdr, is calculated from the difference, in number of wavelengths, between the diffracted signal across the top of the obstacle, which includes the foliage off which the signal was reflected, and the foliated scatter path across the top of the obstacle, calculated using the obstacle height from SRTM database less clutter height.

    The equation is: cdr= scatter phase delay + $\Delta r/\lambda$.

    Converting this equation to use the wave number instead of the wavelength, where the wave number, wn, for an electric field is: $wn = 2\pi/\lambda$, $1/\lambda = wn/2\pi$;

    cdr = scatter phase delay + $\Delta r(wn/2\pi)$

    The scatter phase delay has been empirically determined to be an average of 0.6 = scatter phase delay, so then:

    cdr = 0.6 + 0.159155∗wn*Δr

    a. For a single obstacle, Δr = abs(dto+dro-dtof-drof), then:
       cdr = 0.6 + 0.159155∗wn* abs(dto+dro-dtof-drof).
    b. We then determine the cancellation percentage. The decimal places, the value to the right of the decimal point, of the cdr value, is the phase difference between the two signals in hertz. We strip off the decimal places by subtracting the integer value of the cdr from the full value,

leaving only the decimal places value. This is multiplied by 2 and subtracted from 1, so that the value goes from 1.0 to 0.0 for the first half cycle, representing in-phase to out-of phase, and then, from 0.0 to –1 as the signal passes through the second half cycle, back from out-of-phase to in-phase. The value of 6.0 represents the additive situation, where both signals are in-phase, as $\log_{10}(1) = 0.0$. The maximum out-of phase situation is limited to 0.03, to establish a maximum cancellation value equal to -6-20*(-1.52)=24.45 dB, an empirical practical value determined from the author's experience in calibration and adjustment of deep-nulling antenna arrays on TV translators used to protect the NRAO Greenbank installation. This limiting also prevents a $\log_{10}(0)$ calculation, which fails, as log10(0) is not a computable value. The range of arp is then 1.0 for exactly in-phase, down to 0.03 for completely out-of-phase, and -20*log10(arp) then ranges from 0 for in-phase to 30.4 for out-of phase, resulting in an adiffv loss range of –6 dB (a gain) for in-phase, up to a 24.45 dB limited maximum loss for out-of-phase.

Line:

```
arp=abs(1.0 –2.0*(cdr-(int(cdr))));
arp=mymax(arp,0.03);
adiffv=-6.0-20.0*log10(arp);
```

88. An else statement provides an alternate path to the "if" statement above. If the obstruction is above the tree line, indicating a clean, knife edge diffraction scenario:
89. *Adiffv2* is then set to equal the sum of:
    a. the attenuation from knife edge diffraction from a single obstacle peak to a receive site beyond the peak, with a call to ***aknfe*** with input *q*, to compute the diffraction attenuation for the first obstacle,
    b. a call to saalos to add the clutter losses on the path to the receiver antenna.

Line(new):

```
else
{
        adiffv2=aknfe(q);
}
        adiffv2+=saalos(rd, prop, propa);
```

90. The if and else statements have completed. The subroutine returns adiffv2.

Line (new):

```
return adiffv2;
```

SUBROUTINE AHD: A functional explanation, by Sid Shumate.
Last Revised July 15, 2007.

Approximate tHeta D function for scatter fields; subroutine: ***ahd***.

Note: Used with both point-to-point mode and area mode. Called by ***ascat.***

Descriptions derived from the Irregular Terrain Model description by George Hufford, 2002, (ITMD), compared to the ITM.cpp prepared by J. D. McDonald and John Magliacane for compilation on unix and linux systems. "Line" numbers refer to the ITM.cpp as line numbered by Bloodshed Software's DevC++ print function. "Alg" numbers refer to the algorithm formula in "The ITS Irregular Terrain Model, version 1.22, the Algorithm" by G. A. Hufford, 1995.

From ITMD Section 26:

This is the $F(\theta d)$ function for scatter fields.

As defined in the Algorithm, Section 6, "Addenda – numerical approximations." This section starts by mentioning:

> "Part of the algorithm for the ITM consists in approximations for the standard functions that have been used. In these approximations, computational simplicity has often taken greater priority than accuracy."

The Algorithm later states:

> "we have the two functions, $F(\theta d)$ and H_0 , used for tropospheric scatter. First,
>
> $$F(D, N_s) = F_0(D) - 0.1\,(N_s - 301)e^{-D/Do} \qquad (6.8)$$
>
> where
>
> $$D_0 = 40 \text{ km}$$
>
> And (when D_0 is given in meters)
>
> $$F_0(D) = 133.4 + 0.332 * 10^{-3} * D - 10 * \log D \quad \text{for } 0 < D \leq 10 \text{ km, or}$$
>
> $$F_0(D) = 104.6 + 0.212 * 10^{-3} * D - 2.5 * \log D \quad \text{for } 10 < D \leq 70 \text{ km, or}$$
>
> $$F_0(D) = 71.8 + 0.157 * 10^{-3} * D + 5 * \log D \quad \text{otherwise "} \qquad (6.9)$$

This can also be found in Annex III, Section III.5 "Forward Scatter", of Tech Note 101; from equations [TN101 III.46, 47, and 48] on page III-24.

Call inputs for subroutine ***ahd***:

Td *D*, distance in meters

Declares private, or local, arguments:
i;
a[3]={ 133.4, 104.6, 71.8};
b[3]={ 0.332e–3, 0.212e–3, 0.157e–3};
c*[3]*= { –4.343, – 1.086, 2.171 };

In this subroutine:

1. Initiates an **if** statement. If *td* is less than or equal to 10,000 meters, then the value of *i* is set to be equal to zero.

```
Line 207:     if (td<=10e3)
                    i=0;
```

2. An ***else if*** statement follows; if *td*, at line 207, was more than 10,000 meters, and less than or equal to 70,000 meters, then the value of *i* is set to be equal to 1.

```
Line 178:     else if (td<=70e3)
                    i=1;
```

3. An ***else*** statement follows; so if *td*, at line 207, was more than 70,000 meters, then:

   i is set to be equal to 2.

```
Line  213:    else
                    i=2;
```

4. The subroutine then calculates and returns the value of F_0 (D), using the appropriate formula from (6.9):

```
Line 216:     return a[i] + b[i] *td +c[i] * log(td);
```

SUBROUTINE AKNFE: A functional explanation, by Sid Shumate.

Last Revised: Feb. 14, 2009 to clarify description.
previous last correction March 22, 2008.

Attenuation from Knife Edge Diffraction subroutine.

Note: Used with both point-to-point and area modes. Called by ***adiff.***

Descriptions derived from the Irregular Terrain Model description by George Hufford, 2002, (ITMD), compared to the ITM.cpp prepared by J. D. McDonald and John Magliacane for compilation on unix and linux systems. "Line" numbers refer to the ITM.cpp source code supplied for use with SPLAT! version 1.2.0b, the Linux-based open-source Longley-Rice program, as line numbered by Bloodshed Software's DevC++ print function. "Alg" numbers refer to the algorithm formula in "The ITS Irregular Terrain Model, version 1.22, the Algorithm" by G. A. Hufford, 1995 (the Algorithm). "ITS67" numbers refer to the algorithm formulas in "ESSA Technical Report ERL 79-ITS 67, Prediction of Tropospheric Radio Transmission Over Irregular Terrain, A Computer Method – 1968" (ITS-67) by A.G.Longley and P.L.Rice. Reference may also be made to "Ultra-Short-Wave Propagation, (USWP), by Schelleng, Burrows, and Ferrell, Bell Telephone Laboratories, as published in the Proceedings of the IRE, March, 1933.

From ITMD Section 13:

The function ***aknfe*** computes the attenuation due to a single knife edge – using the Fresnel integral (in decibels) as a function of v^2, the square of the internal wedge angle. The approximation is that given in [Alg. 6.1].

Call inputs: & v2

defines private, or local, arguments:

a attenuation due to a single knife edge

This subroutine:

1. The argument v2, represents the square of v; v is the difference, measured in wavelengths, between:
    a. the direct signal from a transmitter to a receiver over a line of sight, versus the delayed path (including the phase reversal for horizontal or low angle vertical polarity at the reflection point) from a reflection off of a sharp-peaked obstruction that is at or below, but near, the line of sight path, or:

b. the diffracted and delayed signal bending over an obstruction by diffraction, versus the direct wavelet front passing over the top of the knife edge (see external discussions of the theory of a Cornu spiral)

An **if** statement is initiated; if v2 is less than 5.76 (i.e., if v is less than 2.4 wavelengths), then:

a is set to be equal to: 6.02 + 9.11 * sqrt(v2) –1.27 * v2

Line 122: if (v2<5.76)

```
a=6.02+9.11*sqrt(v2)-1.27*v2;
```

2. The following ***else*** statement provides that if v2 is equal to or more than 5.76:

a is set to be equal to: 12.953 + 10*$\log_{10}$(v2).

Note: the ITM, versions 1.2.2 to 7.0, continues to use an early, outdated form of log10, as in the original language used in ITS-67, FORTRAN 66 (a.k.a. Fortran IV), there is no common logarithm function, and the ALOG function was, in fact, the natural log, or ln, function. To obtain the result of a common logarithm (log to the base 10, or $\log_{10}$), the natural log function ALOG was used and the result was multiplied by .4343 to obtain the result that would have been obtained from a common logarithm.

In c++ source code, the FORTRAN ALOG function becomes the **log** function, multiplied by .4343, still duplicating the original form of the source code in order to duplicate the results of the newer **log10**, or common logarithm (log to the base 10) function. Therefore, 4.343*log can be replaced by the more modern 10*log10, changing:

Line 124: else

```
a=12.953+4.343*log(v2);
```

to:

Line 124: else

```
a=12.953+10*log10(v2);
```

3. Subroutine ***aknfe*** returns the value of *a*, the attenuation due to a single knife edge.

This is a very crude approximation derived from Fresnel integral expressions for a, the real component, and b, the phase, or imaginary component.

Line 126: return a;

SUBROUTINE ALOS: A functional explanation, by Sid Shumate.

Last Revised: July 15, 2007.

Attenuation for Line of Sight subroutine; ***alos***.

Note: Used with both point-to-point and area modes. Called by ***lrprop.*** Calls ***abq_alos, mymin,*** and ***mymax..***

Descriptions derived from the Irregular Terrain Model description by George Hufford, 2002, (ITMD), compared to the ITM.cpp prepared by J. D. McDonald and John Magliacane for compilation on unix and linux systems. "Line" numbers refer to the ITM.cpp as line numbered by Bloodshed Software's DevC++ print function. "Alg" numbers refer to the algorithm formula in "The ITS Irregular Terrain Model, version 1.22, the Algorithm" by G. A. Hufford, 1995 (the Algorithm). "ITS67" numbers refer to the algorithm formulas in "ESSA Technical Report ERL 79-ITS 67, Prediction of Tropospheric Radio Transmission Over Irregular Terrain, A Computer Method – 1968" by A.G.Longley and P.L.Rice.

From ITMD Sections 17, 18, 19:

The function ***alos*** computes the line-of-sight attenuation for a distance d. It uses a convex combination of plane earth fields and diffracted fields. A call with d = 0 sets up initial constants.

Call inputs:
double d
prop_type
& prop array with constants
propa_type
& propa array with constants

defines private, or local, arguments:

complex<**double**> prop_zgnd (prop_zgndreal,prop.zgndimag);
static double wls
complex<**double**> r
double s
double sps
double q
double alosv

In this subroutine:

1. An **if** statement is initiated; **if** d is equal to zero, then:

    a. *wls* is set to be equal to: 0.021, divided by (0.021+*prop.wn***prop.dh*), and all of which is divided by the greater of 10,000 or *propa.dlsa*;

    where
    prop.wn is the wave number, = frequency/47.7 MHz *meters
    prop.dh is delta h, or Δh, the terrain irregularity parameter
    propa.dlsa is the sum of the two smooth earth horizon distances; from the terminals to the horizon over smooth earth

    b. alosv is set to be equal to zero.

Line 405:

```
if (d= = 0.0)

{
        wls=0.021/(0.021+prop.wn*prop.dh/mymax(10e3,propa.dlsa));
        alosv=0.0;
}
```

2. An **else** statement follows, so if d is not equal to zero, then:

    a. q is set to be equal to $(1.0 - 0.8^{(-d/50,000)})$**prop.dh*;

    where
    prop.dh is delta h, or Δh, the terrain irregularity parameter

    b. *s* is set to be equal to 0.78 * *q* *10^(-(*q*/16.0)$^{0.25}$));

    c. q is set to be equal to the sum of *prop.he[0]+prop.he[1]*;
    where
    prop.he[0] is the effective height of the transmit antenna
    prop.he[1] is the effective height of the receive antenna

    d. sps is set to be equal to q/sqrt(d*d+q*q);

    e. r is set to be equal to:
    (sps-prop_zgnd)/(sps+prop_zgnd)*exp(-mymin(10.0,prop.wn*s*sps));

f. The subroutine ***abq_alos*** is called with input (r).

The subroutine ***abq_alos*** **,** in its entirety, consists of:

```
double abq_alos (complex<double> r)
{
return r.real()*r.real()+r.imag()*r.imag();
}
```

The subroutine ***abq_alos*** returns r.real()*r.real()+r.imag()*r.imag(), and q is set to be equal to r.real()*r.real()+r.imag()*r.imag();

Line 411:
```
else
{
    q=(1.0-0.8*exp(-d/50e3))*prop.dh;
    s=0.78*q*exp(-pow(q/16.0,0.25));
    q=prop.he[0]+prop.he[1];
    sps=q/sqrt(d*d+q*q);
    r=(sps-prop_zgnd)/(sps+prop_zgnd)*exp(-mymin(10.0,prop.wn*s*sps));
    q=abq_alos(r);
```

3. An **if** statement is initiated; if *q* is less than 0.25 or if *q* is less than *sps*, then:
   r is set to be equal to $r*(sps/q)^{1/2}$;

Line 420:
```
if (q<0.25 || q<sps)
    r=r*sqrt(sps/q);
```

4. alosv is then set to be equal to [*propa.emd* * *d* + *propa.aed*];
   where:
   - *propa.emd* has been set equal to (a4-a3)/(d4-d3) in ***lrprop***
   - *d* is the path distance
   - *propa.aed* has been set equal to *a3–propa.emd*d3*

   And *q* is reset to be equal to: *prop.wn*prop.he[0]*prop.he[1]**2.0/*d*;
   where:
   - *prop.wn* is the wave number
   - *prop.he[0]* is the effective height of the transmit antenna
   - *prop.he[1]* is the effective height of the transmit antenna
   - *d* is the path distance

Line 423:
```
alosv=propa.emd*d+propa.aed;
q=prop.wn*prop.he[0]*prop.he[1]*2.0/d;
```

An **if** statement is initiated; if *q* is greater than 1.57, then *q* is reset to be equal to: 3.14- (2.4649/*q*).

```
Line 426:    if (q>1.57)
                 q=3.14-2.4649/q;
```

5. The subroutine ***abq_alos*** is called with input (complex<double>(cos(q),-sin(q))+r)).

The subroutine ***abq_alos*** **,** in its entirety, consists of:

```
double abq_alos (complex<double> r)
{
return r.real()*r.real()+r.imag()*r.imag();
}
```

The subroutine ***abq_alos*** returns r.real()*r.real()+r.imag()*r.imag().

alosv is then reset to be equal to:

(-4.343*log((return from ***abq_alos***: r.real()*r.real()+r.imag()*r.imag() - alosv)*wls+alosv;

```
Line 429:    alosv=(-4.343*log(abq_alos(complex<double>(cos(q),-sin(q))+r))-
             alosv)*wls+alosv;
             }
```

NOTE THE USE HERE OF THE CONSTANT 4.343, INSTEAD OF 10 AS USED IN LONGLEY RICE.

6. The subroutine ***alos*** returns *alosv*, the value of the line of sight attenuation;

```
Line 432:    return alosv;
             }
```

SUBROUTINE ALOS2: A functional explanation, by Sid Shumate.

Revised Oct 18, 2010 to utilize reflection point determination in ***hzns2***.
Last Revised: Sept. 19, 2008.

Attenuation for Line of Sight, version 2, for use as part of ITWOM; ***alos2***.

Note: To be used with point-to-point mode and area mode. Called by ***lrprop.*** Calls ***abq_alos, mymin,*** and ***mymax.***

Descriptions derived from the Irregular Terrain Model description by George Hufford, 2002, (ITMD), compared to the ITM.cpp, the ITMDLL.cpp, minimally modified by J. D. McDonald and John Magliacane for compilation on unix and linux systems. "Line" numbers refer to the ITM.cpp as line numbered by Bloodshed Software's DevC++ print function. "Alg" numbers refer to the algorithm formula in "The ITS Irregular Terrain Model, version 1.22, the Algorithm" by G. A. Hufford, 1995 (the Algorithm). "ITS67" numbers refer to the algorithm formulas in "ESSA Technical Report ERL 79-ITS 67, Prediction of Tropospheric Radio Transmission Over Irregular Terrain, A Computer Method – 1968" by A.G.Longley and P.L.Rice.

Background: Place earth field, a.k.a. ray-tracing, two-ray, and multipath calculations.

Plane earth fields calculations, often referred to as two-ray, or, when expanded in scope, as multiple ray tracing, multipath, wavefront, or wavelet calculations, start out with the basic concept of a direct, main signal ray, r_0, also described as the "incident plane wave", traveling from the transmitting antenna to the receive antenna. A second set of rays, r_1 and r_2, are traveling in the same plane as r_0. The ray r_1 travels from the transmitter antenna to the earths' surface, where a reflected ray r_2 then proceeds to meet up with the direct ray at the receive antenna. The combination of these two rays is complex; the two signals can theoretically add together to double (a 6 dBμ increase) the RF field arriving at the receive antenna, but only if they arrive at the same time, (i.e. "in phase"); each with a path length equal to a precise multiple of the wavelength of the signal. If they arrive 180 degrees (or π radians) out of phase, they will cancel each other out, creating a null in the received signal, often experienced as a "stoplight fade" while listening to FM in a car.

The angle between ray r_1 and the ground is called the grazing angle, and is normally represented by the Greek symbol (Ψ). If the reflecting medium (the earth) is not a perfect reflector, the grazing angle between the ray leaving the reflection point, r_2 , and the earth, will be perpendicular (90 degrees off of) the angle of the signal absorbed into the ground. However, in the Longley-Rice equations, the earth's surface is close enough to a perfect reflector that the grazing angle of ray r_1 is often, for practical purposes, considered to be equal to the grazing angle of ray r_2.

Another factor in the path calculation is the phase of the reflected signal. In horizontal polarization, the phase of the reflected signal reverses (zero degrees becomes 180, or the effective length changes by ½ wavelength) at the reflection point. In vertical polarization, the reflected ray r_2 is in phase with the incoming ray r_1 at a high grazing angle approaching $\pi/2$ radians (90°), but as the path length increases, the grazing angle becomes smaller. At a particular grazing angle the strength of the vertically polarized reflection signal, r_2, drops into a wide, deep null to zero. For a perfectly reflecting medium, this grazing angle is referred to as the "Brewster" angle, and can be determined using the equation: $\Psi_B = \cot^{-1} (\varepsilon_r)^{1/2}$ where for two lossless mediums, $\varepsilon_r = \varepsilon_2 / \varepsilon_1$, and the product of the dielectric constants ε_2 and ε_1 of the two media is a real number.

Slight differences in the reflecting media cause this critical angle to change values for signal strength (the real number) and phase (the imaginary number); for any imperfect reflecting medium, this critical grazing angle is then referred to as the "Pseudo-Brewster angle" (PSB). For a PSB, the null in the vertically polarized reflection is not as deep as for a Brewster angle. Below the PSB, the vertically polarized reflection signal rapidly changes phase with the angle, from 90 degrees ($\pi/2$ radians) to 180 degrees ($\pi/2$ radians) out of phase. For the origin of the equations regarding the Pseudo-Brewster angle, we defer to "The Pseudo-Brewster Angle", by G. P. Ohman. (see bibliography).

Ground (or sea) irregularities at the reflection point, and clutter, at and near the reflection point and between the reflection point and a low receiver, can affect the strength, phase, and phase coherence of the received reflected signal.

Reflection coefficient, the PSB and low grazing angles:

"Grazing Behavior of Scatter and Propagation Above Any Rough Surface", was published in January of 1998 in the IEEE Trans. On Antennas and Propagation. The author is Dr. Donald Barrick, a radar propagation expert who served from 1972 to 1982 as Chief of the Sea State Studies Division of NOAA's Wave Propagation Laboratory in Boulder, CO. The conclusions include:

> "Our results show that backscattered power depends on grazing angle to the fourth power; the impedance and admittance are constant as grazing is approached. These relations hold true for both polarizations, for arbitrary surface materials (including perfect conductors), for all frequency/roughness scales, and for a single deterministic roughness profile as well as averages over surface ensembles. …we considered only backscatter rather than arbitrary bistatic scatter, …but the extension to bistatic scatter is obvious: as either the incidence or scattering angle alone approaches grazing, echo power decreases as grazing angle squared.
>
> Although our approach was primarily employed to establish general grazing-limit behavior, our simple angle-independent constants describing backscatter and propagation are useful in their own right; these expressions allow

> a single numerical evaluation to serve the entire near-grazing region up to the Brewster angle."

Bistatic radar systems, which include but are not limited to passive radar systems, are radar systems where the transmitter and receiver are widely separated. They operate in a like manner to FM and TV broadcasting systems with respect to plane wave considerations; therefore, we can extend Dr. Barrick's conclusions to the case of plane wave reflections. The theory, however, must now change in name from plane wave to wavefront, or wavelet, consideration, as Barrack's conclusions shift the analysis from a 2 dimensional plane-wave consideration of a reflection at a spot in a path, to a three-dimensional, Newtonian conservation-of-energy revelation, regarding the consideration of wavelet-based reflections from a large ground area.

Stated another way; we are not actually dealing with two rays; that is a two dimensional (plane) model useful for calculation of the strongest, central part of a wavefront. In a model closer to reality, we are dealing with a wavefront, and wavelet theory in four dimensions. Here Barrick found that a form of Newton's second law, a conservation of energy, exists in radio reflection. The reflection energy at small grazing angles is not entirely lost due to rough surface scattering; for rough but unobstructed surfaces at the reflection point for which there is no ground clutter absorption (such as a rough desert surface) the energy of a narrowband radio signal in the main reflection ray r_1 and r_2 that is lost due to roughness of the surface, in a two-dimensional single-ray consideration, is regained in additional reflections from the rough surface in a three-dimensional wavefront reflection area consideration, and therefore, for a ray reflecting off of a surface without significant obstructions or absorbing ground clutter such as vegetation, the strength of the reflected ray will essentially be the same as over a smooth surface. One can say that the energy that reflects off in other directions, and is lost due to a rough (but still highly reflective) surface, equals the energy reflected to the receive terminal from other points on the same surface that would have otherwise been directed, in plane waves, toward other receive terminals. The phase coherence, however, will be affected; there will be a variation in arrival time from these additional reflections that will create variations in phase and realizable signal strength across the bandwidth of wideband signals.

Barrick's equations allow us to ignore the previously considered scattering effect of roughness on plane wave calculations at low grazing angles over nearly smooth or very slightly irregular surfaces, treating these surfaces as smooth earth surfaces with respect to roughness. Absorption (refractive transmission of the signal into a lossy medium, rather than reflection, or passage through an absorptive medium, such as leaves), however, remains a major factor, rapidly damping out the two-ray or wavefront reflection losses to near zero as the terrain roughness factor, Δh (dh) increases beyond 4 meters. As Δh continues to increase, these absorptive losses take over as the primary cause of line-of-sight losses in addition to free space losses for the line of sight range.

The Absorptive Clutter Losses, which we will refer to with the designation “AB” may be referred to as “Clutter”, or a component of “Clutter” however, this is somewhat confusing, as it does not match the definition of “Clutter” used in ITU-R P.1546-2.

The complex relative impedance, Z_g, derived from the ground constants in subroutine ***qlrps***, is used to generate a Reflection coefficient, R_e, with a range between 1 for no reflection loss to zero for no reflection. In vertical polarity, this reflection coefficient also incorporates the effect of the Pseudo-Brewster angle (PBA), generating a reflection null at the PBA, and a 180-degree phase shift below the PBA. The effect of ground irregularities are quantified using $\Delta h(d)$, the terrain roughness factor estimated at the subject path distance d, and incorporated into the argument sigma (σ), or *s* in the code, and an exponent factor, based on factors *s*, *k*, and sin Ψ, is multiplied to the reflection coefficient R_e’, and represented in the code by argument *r*, for consideration of the effect of roughness, utilizing the dh factor, for reducing the depth of the nulls in the plane wave comb pattern at high grazing angles above Brewster’s angle.

To summarize; to calculate the additional loss due to the interaction of ground-reflected waves with the primary, direct plane wave in line of sight paths prior to an obstruction, we will have to, by some means:

1. Calculate the Brewster angle.

2. For horizontal polarity:

    a. Calculate the “comb pattern” gain and attenuation due to the combination of the main and reflected rays.
    b. Incorporate the 180-degree phase change at the reflection point by adding a ½ wavelength of length to the reflected path total length computation. Adjust the attenuation amount due to the two-ray resultant comb pattern and far field null-out “depth” using a Rayleigh Criterion based reflection coefficient adjustment to account for damping effect of ground clutter and phase coherence reduction effect of rough ground reflections.
    c. Calculate the clutter losses, based on a set of calculations empirically derived from ITU Recommendation P.1546-2 Figures 1, 9, and 17.

3. For vertical polarity, in addition to the steps above for horizontal polarity: Above the Pseudo-Brewster angle; calculate the “comb pattern” gain and attenuation for no phase change at reflection point. Also calculate the Pseudo-Brewster angle signal null and phase angle change below the PBA.

Many of these steps are incorporated, often in non-obvious ways, within the procedural steps below; we will point out how they are accomplished.

From ITMD Sections 17, 18, 19:

The function ***alos2*** computes the line-of-sight attenuation for a distance d. This is a parallel function to the ***alos*** subroutine originally found in the ITMDLL.cpp. It computes losses due to two-ray losses (plane earth field losses), and also considers 3-dimensional multipath from rough, unobstructed terrain, which, as per Barrick's equations, includes and incorporates any Ricean distributed scattered multipath. A call with d = 0 sets up initial constants for the area mode; not required for line-of-sight.

Improvements associated with ITWOM include replacing the older, discredited line-of-sight range diffraction computations with a determination of ground clutter layer-related losses by a call to subroutine ***saalos***, which uses the line-of-sight equations and methodology from Shumate's Approximations, a set of deterministic approximation equations for Radiative Transfer Engine functions derived from ITU-R 1546-2, Figures 1, 9 and 17.

Call inputs:
double d
double pfl[ ]
prop_type
& prop array with constants
propa_type
& propa array with constants

defines private, or local, arguments:

complex<**double**> prop_zgnd (prop_zgndreal,prop.zgndimag);
complex<**double**> r

double acd attenuation from clutter at path distance d
double cd direct signal clutter exposure distance, in meters
double cr reflected signal clutter exposure distance, in meters
double dr distance to the reflection point
double hr height of transmit antenna AMSL
double ht height of receive site antenna AMSL
double hrp terrain height at the 2-ray reflection point in meters AMSL.
double re square of the polar vector value of the reflectivity coefficient
double s sigma, representing the modified standard deviation of Δh (dh), the terrain irregularity factor
double sps sin Ψ, the sin of the (reception point) grazing angle
double q utility argument, used to store temporary values in the subroutine
double alosv attenuation in the line of sight range (in addition to free space loss).

int rp pfl array interval number at the reflection point

Here it is necessary to add an additional consideration not found in the original ITM version 1.2.2. In the two-ray calculations, as the ground clutter (represented as a part of

$\Delta h(d)$, for lack of a better quantification) increases, the cancellation effect of the reflected ray disappears in the early part of the signal path; the reflectivity, R, of the ground clutter canopy is low near the transmitter site, and the transmissivity is high, so that the signal passes through the clutter canopy, and through the clutter, in order to bounce off the ground. Then, if the receiver is above the clutter canopy, the signal must pass again through the clutter and clutter canopy to get to the receive point, being attenuated along the way by the clutter to the point that the multipath cancellation is negligible. If the receive point is below the clutter canopy, the clutter attenuation also reduces the cancellation significantly.

But when the grazing angle gets so small that the direct ray also passes through most of the same ground clutter as the reflected signal, the signal strength of the direct ray starts to approach the signal strength of the reflected ray, and the 2-ray cancellation effect again becomes noticeable. To quantify this effect, we will make the assumption that the absorbing clutter through which the direct and reflected rays pass is statistically homogenous; i.e. attenuates approximately equally with distance. We will also assume, on the average, that the reflection point is at the midpoint in height of a layer of clutter represented by the height of the terrain irregularity factor.

In addition, for reception points above the clutter canopy, from Snell's Law geometry and Transmissivity/Reflectivity computations, the reflectivity R of the clutter canopy/ground irregularities increases toward 1.0 as the grazing angle approaches zero, allowing the cancellation effect of 2-ray computations to regain effect, as a slowly increasing cancellation factor.

It will be necessary to determine the elevation height of the reflection point. For the point-to-point mode, this can be determined using the value of rp=int(floor(d/pfl[1])) to obtain the pfl[rp+1] value from the terrain elevation database array. For the area mode, we will not have the advantage of a terrain elevation database, and the exact reflection point height that is critical to this calculation is not available; therefore, we will not consider this additional attenuation in the area mode.

To determine the location of the reflection point in order to determine the elevation height of the reflection point, the first step is to declare new arguments, **int** *rp*, the interval number of the reflection point, **double** *dr*, the distance to the reflection point, and **double** *hrp*, the elevation height at the reflection point. We will also later need **double** *hr*, *ht*, *cd*, and *cr*.

We will utilize the equation used in a test 2-ray multipath spreadsheet to determine the distance to the reflection point, *dr*; however, we will simplify to eliminate the refraction adjustments in order to obtain the correct value for the distance to the reflection point in the elevation database. Therefore, calculating the distance from the transmitter to the reflection point, r_1, in meters:

$$d1 = dr = d/(1 + h_r / h_t) \qquad (33.2)$$

where to calculate d1, instead of: h_{er} = prop.he[0] and h_{et} = prop.he[1], which are calculated based on the height above the average terrain line, we will use the more exact:

> h_t = (transmitter antenna RCAGL) + (terrain height above ground level at the transmitter site) = prop.hg[0] + pfl[2]
> h_r = (receive antenna RCAGL) + (terrain height above ground level at the receive point) = prop.hg[1] + pfl[np+2].

These are already calculated in subroutine hzns, as za (ht) and zb (hr), and xi is also set to pfl[1]. To avoid having to load the entire pfl array into ***lrprop2***, and then into ***alos2***, we have calculated the reflection point location and height, and the interval width value, while in ***hzns*** and stored the values in the prop array.

1. The values calculated in subroutine ***hzns*** are retrieved:
    a. Transmitter height, AMSL, in meters: ht=prop.ght;
    b. Receiver height, AMSL, in meters; hr=prop.ghr;
    c. Reflection point, designated by interval number; rp=prop.rpl;
    d. Reflection point height, in meters; hrp=prop.rph;

    We will use these values to determine the distance through the clutter that the reflected signal traverses, and to recalculate the effective terminal heights to be used to determine the relative phase difference between the direct and reflected signals.

Line (new):
```
ht=prop.ght;
hr=prop.ghr;
rp=prop.rpl;
hrp=prop.rph
```

2. An **if** statement is initiated; **if** d is equal to zero, then:

    a. Argument *alosv* is set to be equal to zero.

Line (new):
```
if (d= = 0.0)

{
        alosv=0.0;
}
```

3. An **else** statement follows, so if d is not equal to zero, then:

    a. q is set to equal the sum of the effective transmitter and receiver heights, and then used to calculate sps, the sin of psi, or Ψ, the receive site grazing angle, by dividing q by the square root of the squares of the distance and the sum of the effective heights.

b. q is repurposed to be equal to $(1.0 - 0.8e^{(-d/50,000)})$**prop.dh*; This equation takes the determined overall value of Δh, the terrain irregularity parameter, and calculates the median estimate of Δh(d), the terrain irregularity parameter at a desired distance d; an equation empirically derived from a study of a large number of profiles. For example; a Δh of 90 meters, the resulting Δh(d), using an exponent to the base e, ranges from 19.4 meters at a distance of a kilometer, to 63.4 meters at 50 km. and on to 90 meters at 100 km . [ITS67 (3), p. 7], also see [ITS67 3.6a (modified)] or [Alg. 3.9]

where

prop.dh is delta h, or Δh, the terrain irregularity parameter
d is the path distance in meters

Line new:

```
else
{
q=prop.he[0]+prop.he[1];
sps=q/sqrt(d*d+q*q);
q=(1.0-0.8*exp(-d/50e3)*prop.dh;
```

The distance, dr, from the transmitter site to the reflection point, calculated in ***hzns2***, is retrieved from prop.rpd. The reflection point height is subtracted from the transmitter height AMSL (h_t , or *ht*) and the receive height AMSL, (h_r , or *hr*) to obtain the most accurate determination of the actual difference in wavelengths (not considering, yet, the 180° phase shift at the reflection point) between the direct and reflected rays:

Line (new):

```
if prop.mdp<0
{
dr=prop.rpd;
```

4. Now that the distance to the reflection point, dr, and the reflection point height, hrp, are known:

a. Calculate the direct signal Clutter Exposure, CEr_0, the distance that the direct signal r_0 traverses after it dips into the ground clutter, by multiplying the total path distance by the ratio of: the clutter height above the receiver, to the height difference between the transmitter and the receiver:

$$cd = CEr_0 = d*(prop.cch - hrg)/(h_t - h_r)$$

where:

d = distance from transmit terminal to receive terminal
prop.cch = the average height of the clutter AGL, in meters
hrg = receive terminal height AGL in meters

$ht = h_t$ = transmit terminal height AMSL in meters
$hr = h_r$ = receive terminal height AMSL in meters

b. Calculate the reflected signal Clutter Exposure, $CEr_{1,2}$, the distance that the reflected rays r_1 and r_2 traverse through the ground clutter:

$cr = CEr_{1,2} = dr(prop.cch/(h_t - h_{rp})) + (pd - dr)$
where:
$dr = d_1$, the ground distance traversed by reflected ray r1 between the transmit terminal and the reflection point
hrp=reflection point height AMSL in meters
The (d – dr) term is the ground distance traversed by reflected ray r1 between the reflection point and the receive site.

c. The ratio of: clutter exposure for the direct ray path, to clutter exposure for the reflected ray path, which will be used to determine the amount of 2-ray loss effect that will return, is then:

$$CER = CEr_0/CEr_{1,2} = cd/cr$$

The effect will return as this ratio approaches 1. But the clutter loss per meter traversed can be high; the CER ratio will have to be very close to 1.0 for the 2-ray cancellation to be noticeable. The CER will normally be a miniscule fractional value, and at its highest will approach 1. A logarithmic function is used to reduce the $\Delta h(d)$ to near zero, when the CER closely approaches 1.

Therefore, as a reasonable first approximation, when the value of sin Ψ drops below 0.2 radians, and as the CER approaches 1, we fade back in the 2-ray cancellation effect, by multiplying $\Delta h(d)$ by the term:

$$(\min(-20*\log_{10}(CER), 1.0) = (\min(-20*\log10(cd/cr), 1.0)$$

The log of a fraction will produce a negative value. The –20 is an arbitrarily chosen gain constant that can be adjusted to match measured results. The term will produce a result of 1.0 until as such time as the direct ray traverses 90% of the clutter distance that the reflected ray traverses; the term will then approach 0 as the CER approaches a 1:1 ratio. This term is multiplied to the $\Delta h(d)$ only when the grazing angle becomes small. This is accomplished by adding an ***if*** statement, as shown below the existing calculation of the $\Delta h(d)$, the value of which is temporarily held by q.

Line new:
```
if sps<0.1
  {
    if ((prop.he[1]  – prop.dh) < 0.0)
```

```
            {
             cd = mymax(0.01,pd*(prop.cch – hrg )/(ht – hr));
             cr = mymax(0.01,pd-dr+dr*(prop.cch/(ht – hrp));
             q=(1.0-0.8*exp(-d/50e3)*prop.dh*(mymin(-20*log10 (cd/cr),1.0)));
            }
        }
    }
```

5. *s*, (a.k.a. the factor $\sigma_h(d)$, originally was specified to be the root-mean-square (rms) deviation of modified terrain elevations, y_i, relative to the smooth earth curve defined by (TN101 5.16], within the limits of the first Fresnel zone in the horizontal reflecting plane. In the irregular terrain model, it has been changed to a empirically derived "representation" of the standard deviation of the terrain and terrain clutter within the limits of the first Fresnel zone in the dominant reflecting plane, at a distance d, derived from Δh(d). It is set to be equal to 0.78 * *q* *10^ (-(q/16.0)$^{0.25}$)), where q, at this point in the subroutine, represents the terrain irregularity parameter at the same desired distance d, or Δh(d). [ITS67 3.6a modified], or [Alg.3.10]

   For example, for a Δh(d) of 50 meters, using an exponent with base e, then *s* will range from 20 meters at d = kilometer to 93 meters at 50 km.

TN101 clearly states that, except where noted, all logarithmic functions are to the base 10. In the text, log refers to $\log_{10}$, or common logarithmic functions.

The ITMDLL.cpp source code uses the c++ exp function three times in this subroutine, which returns the exponential function of x, which is the e number, 2.718282, raised to the power x. This is the natural, not common, exponential function. In c++, to obtain 10^x requires use of the pow function. Since there was no specific mention of the use of base e, it was necessary to consider whether replacing the use of exp(x) with pow(10,x) was necessary to obtain the correct function.

We found that base 10, however, does not work for the Δh and σ calculations, which are separate, statistical empirical constructs based on curve matching to derived empirical results. They are, in fact, exponents to the base e. This statement for Δh can be verified in two ways; by comparing the results of the Δh(d) calculation on the spreadsheet, with Δh set to 650 meters, to Figure 2.8 of ITS-67, page 2-13, and noting in the Algorithm that the equation [Alg. 3.9] clearly states that it is a base e exponent. As to the σ calculation, we see by substituting an exponent to the base 10 that the answer becomes too large too soon and absurdly large at a great distance, verifying that the exponent is intended to be a natural exponent, i.e. to the base e. Similarly, the third use, found below, also appears to work correctly as a natural exponent.

a. q is repurposed by being reset to be equal to the sum of: *prop.he[0] + prop.he[1]*;
   where
   prop.he[0] is the effective height of the transmit antenna

prop.he[1] is the effective height of the receive antenna

b. *sps* (a.k.a. sin Ψ), the sine of the grazing angle, is set to be equal to *q*/sqrt(*d***d*+*q***q*), which is the sum of the transmitter effective height he[0] and receiver effective height he[1], divided by the length of the reflected rays r_1 and r_1, the two rays in the reflected path. The reflected path length is calculated as the square root of the sum of the path distance squared and the square of the sum of the effective heights.

NOTE: This shortcut formula for calculating *sps* works well, and provides the same answer as a more conventional and slower, rigorous trigonometric solution.

c. The magnitude of the theoretical plane earth reflection coefficient, *r* (replacing $R_{h,v}$ in the earlier documentation, including ITS67, with a redefined, dual-polarity R_e' derived from the relative transfer impedance Z_g), is set to be equal to:

$$R_e' = ((\sin\Psi - Z_g)/ (\sin\Psi+Z_g))\exp[-k\sigma_h(d) \sin\Psi],$$

NOTE: This is [Alg. 4.47], EXCEPT THAT (1) d has been substituted for the distance s for clarity; (2) a limitation on the maximum size of the exponent term, ($k\sigma_{h(d)} \sin\Psi$), of –(10), has been added without documentation in the Algorithm.

Note, for comparison to ITS67 3.8a, that since $k = 2\pi/\lambda$, then ($k\sigma_{h(d)} \sin\Psi$) = ($2\pi(\sigma_{h(d)} \sin\Psi)/\lambda$), thereby incorporating, without reference in the documentation, a Rayleigh criteria-derived determination of roughness.

R_e' = (sps – prop_zgnd)/(sps + prop_zgnd)*exp(– mymin(10.0,prop.wn*s*sps));
[in lieu of ITS67 3.9a to 3.12, and 3.5, or Alg. 4.47 modified]

To understand this equation, it must be split into two parts. The (sps – prop_zgnd)/(sps + prop_zgnd) part is the reflection coefficient, R_e, representing the percentage of power in ray r1, that is reflected as ray r2 over smooth earth. By changing the definition of zgnd, a.k.a. the ground transfer impedance, Z_g, depending upon the polarity of the signal, this part of the equation includes consideration of the Pseudo-Brewster angle for vertical polarity, including the vertical polarity PBA null, and the phase reversal from above to below the PBA. Above the PBA, the vertical polarity signal does not have a phase reversal at the reflection point. Below the PBA, at small grazing angles (toward the horizon), the phase of the vertically polarized signal matches the horizontally polarized signal, which always has a phase reversal (a 180 degree, π radians, or ½ wavelength change) at the point of reflection. The range of this part of the equation is from zero to one; zero indicating no reflection, and one indicating a no-loss reflection.

The exponential part of this equation moderates the two-ray effects based upon the Rayleigh criterion applied to the terrain roughness, and functions correctly as a base e exponent.

The Rayleigh Criterion, derived from optics as applied to electromagnetic wave theory, considers a reflection surface smooth if the path length differences due to terrain roughness are no more than a small fraction of a wavelength. Specifically, a surface is considered rough if the phase difference variation due to the roughness is less than 0.5π radians, or 90°. The phase shift equation used in ITS67 is:

$$\delta_{ITS67} = 2\pi\Delta r/\lambda$$

The path length equation for the reflected ray ($r_1 + r_2$) is:

$$r_1 + r_2 = 2(h_{e1}h_{e2})/d \qquad \text{[TM101 5.9],}$$

$$\Delta r = 2((h_{e1+}\Delta t)(h_{e2+}\Delta t)-(h_{e1}h_{e2}))/d$$

Since the wave number, k, or prop.wn, is equal to $2\pi/\lambda$, combining we get:

$$\delta = wn*\Delta r = (wn)*2((h_{e1+}\Delta t)(h_{e2+}\Delta t)-(h_{e1}h_{e2}))/d \qquad \text{from [Alg. 4.49, with d for s]}$$

$$\delta = (wn)*2((h_{e1}h_{e2}+\Delta th_{e1}+\Delta th_{e2}+\Delta t^2-h_{e1}h_{e2}))/d = (wn)*2((\Delta th_{e1}+\Delta th_{e2}+\Delta t^2))/d$$

$$\delta = (2wn\Delta t)(\Delta t + h_{e1}+h_{e2})/d$$

The ($\sin\Psi$) after the terrain height change is: $(\sin\Psi) = (\Delta t + h_{e1} + h_{e2})/d$, so by replacing $(\Delta t + h_{e1} + h_{e2})/d$ with ($\sin\Psi$):

$$\Delta\delta = wn*\Delta r = wn(\Delta t)(\sin\Psi)$$

So the terrain is considered rough if: $\Delta\delta = wn(\Delta t)(\sin\Psi) > 0.5$ radians

Solving for the roughness decision point of Δt:

$$\Delta t = 0.5 / wn*(\sin\Psi) \text{ in units of meters}$$

As we move along the path, the grazing angle Ψ will get smaller; and the ($\sin\Psi$) will get smaller. For a 300 m. effective transmitter height, ($\sin\Psi$) will change from approximately .99 at the first terrain interval from the base of the tower to 0.3 at 1 km, and to 0.006 at 50 km. Since ($\sin\Psi$) is in the denominator of the equation for Δt, the roughness decision point becomes larger as we go away from the transmitter site. Therefore, the same roughness of terrain, as considered by the reflecting rays, appears much less rough at a distance than when the ray is almost vertical, near the transmitting site.

When working with the Rayleigh criterion, the roughness is usually described in terms of the standard deviation σ, (here, *s*) of the terrain around the mean level. A constant, C is derived, assessing the roughness in terms of σ;

$$C = 4\pi\sigma(\sin\Psi)/\lambda = 2k\sigma(\sin\Psi)$$, or in c++, C = 2(*prop.wn*)(*s*)(*sps*)

However, this is based on the Rayleigh criterion's usual derivation of the phase difference as being:

$$\Delta\delta = 2(\Delta t)(\sin\Psi)$$

where:

the $(\sin\Psi)$ is derived from the height change divided by the horizontal distance of the incident, or incoming, ray.

In the derivation for our two-ray example, the change in phase is:

$$\Delta\delta = wn*\Delta r = wn*(\Delta t)(\sin\Psi)$$

So in this case, and by replacing $2\pi/\lambda$ with the wave number, k, a.k.a. prop.wn, we have:

$$C = 2\pi\sigma(\sin\Psi)/\lambda = k\sigma(\sin\Psi)$$, or in c++, C = (*prop.wn*s*sps*)

Which we recognize as the exponent of the right side of the equation for R_e'

The Rayleigh criterion here is that if $C < 0.1$, then we have a smooth surface. If $C > 10$ then the reflection is so diffuse that it can usually be neglected. So we can limit the equation at 10, and we can understand the limitation on the maximum size of the exponent term, $(k\sigma_{h(d)}\sin\Psi)$, of 10, that is found in the c++ code as mymax (10, *prop.wn*s*sps)*. This limitation is not documented in the Algorithm.

Checked for proper exponential function to the base e, the code reads:

```
r=(sps-prop_zgnd)/(sps+prop_zgnd)*exp(-mymin(10.0,prop.wn*s*sps));
```

What *r* represents at this point is the complex reflection coefficient R_e'.

d. The subroutine ***abq_alos*** is called with input (r).

The subroutine ***abq_alos*** **,** in its entirety, consists of:

```
double abq_alos (complex<double> r)
{
return (r.real()*r.real())+sqrt(r.imag()*r.imag());
}
```

The subroutine ***abq_alos*** returns the sum of the square of r.real (r.real()*r.real()) plus the square of (r.imag()*r.imag()), the absolute value of the real and phase components of the input argument. Note, however, that it does not include the step of taking the square root of the sum of the squares, which would provide the value of the vector r, using the Pythagorean theorem, $r = (a^2 + b^2)^{1/2}$.

q is then set to be equal to the sum of the squares of the real and imaginary components of $R_{h,v,}$ or R_e; r.real()*r.real() + r.imag()*r.imag(). So at this point, q represents the square of the value of: $R_{h,v} \exp[-(2\pi\ \sigma_h \sin\Psi)/\lambda]$ in [ITS67 3.8a], i.e., the square of the polar co-ordinate "r" vector value of the reflection coefficient.

In addition, in the test spreadsheet, we notice that minor rounding and truncation error causes the value of r to exceed its theoretical maximum of 1. So we add a corrective line that limits the value of r at a maximum of 1.0.

Line (new):

```
s=0.78*q*exp(-pow(q/16.0,0.25);
q=prop.he[0]+prop.he[1];
sps=q/sqrt(d*d+q*q);
q= exp(-mymin(10.0,prop.wn*s*sps));
r=((sps-prop_zgnd)/(sps+prop_zgnd))*q;
q=abq_alos(r);
q=mymin(q, 1.0);
```

6. An **if** statement is initiated; if *q*, now representing the square of the value of: $R_{h,v} \exp[-(2\pi\ \sigma_h \sin\Psi)/\lambda]$, is less than 0.25, and if *q* is less than *sps*, i.e. if the reflection coefficient is less that 0.25, and close to the transmitter site, where *sps* approaches the value of 1.0, R_e", is set to be equal to: $r*(sps/q)^{1/2}$, or $r*(\sin\Psi/abq\_alos(r))^{1/2}$, otherwise r, remains R_e".
[ITS67 3.8a, 3.8b modified] or [Alg. 4.48]

If the reflection coefficient is both smaller than .25 and ($\sin \Psi$), which occurs for rough ground near the transmitter site, at large grazing angles where refractivity is not relevant, depending upon the transmitter site effective height and the terrain irregularity, then the reflection coefficient is estimated as having a value equal to: $(\sin \Psi)^{1/2}$, with the sign of r (+ or -) retained by dividing r by its modulus, or absolute value.

Line new:

```
if (q<0.25 || q<sps)
{
        r=r*sqrt(sps/q);
}
```

7. Here *q* is again reassigned and reset to be equal to the value of delta (δ), representing the phase difference, as a multiple of wavelengths, between the direct ray r_0, and the primary reflected ray, $r_1 + r_2$, for vertical polarity above the Brewster angle, specified in radians, calculated as per [ITS67 3.3a] to be:

$$\delta_{ITS67} = 2\pi\Delta r/\lambda = \pi(\Delta r)f_{MHz}/150 = .041917\, f_{MHz}\,(he_1*he_2)/(d)$$

where: $f_{MHz}/300 = 1/\lambda$

$\Delta r = (he_1*he_2)/(d) = 2*(he_1*he_2)/(d)$

he_1, he_2 are the effective heights of the transmit and receive antennas

d is the line-of-sight path distance in meters.

NOTE: δ_{ITS67} is expressed in electrical radians. In c++ code, we will calculate this is terms of number of wavelengths:

*prop.wn*prop.he[0]*prop.he[1]/(d*π)*;

which is [ITS67 3.3a] or [Alg. 4.49] modified for units of wavelengths instead of radians.

where:

prop.wn	is the wave number, $f_{MHz}/47.4$
prop.he[0]	is the effective height of the transmit antenna
prop.he[1]	is the effective height of the transmit antenna
d	is the path distance

Note: Here we have another major weakness in the determination of Δr above. This calculation does not utilize the actual height of the reflection point, which we have now already determined above. Instead, it uses the effective heights of the terminals, which incorporates by reference the average terrain height line calculated by subroutine zlsq1. Clearly, as stated in the documentation, such information was to be used instead of calculated estimates such as the average terrain height line, where available. **So we modify the equation using the terrain reflection point height, *hrp*, which allows us to use the heights of the transmit (*ht*) and receive (*hr*) terminals above mean sea level (AMSL) for highest accuracy, when in the point-to-point mode.**

Line (new):

```
q=prop.wn*prop.he[0]*prop.he[1]/ d*3.1415926535897

if prop.mdp<0
        {
        q=prop.wn*((ht- hrp)*(hr- hrp))/d*3.1415926535897;
        }
```

8. In the original ***alos***, where the line above reports in units of radians, here an **if** statement was initiated; if *q*, currently holding the value of δ', the phase difference in radians between the direct ray, r_0, and the indirect ray path $r_1 + r_2$, is greater than 1.57, (equal to $\pi/2$, or one-quarter rotation, or 90°) then *q*, representing δ, was reset to be equal to: 3.14- (2.4649/*q*), a.k.a $(\pi - (\pi/2)^2/\delta')$.

This compresses all rotation after the first ¼ cycle into just under one single slow cycle that approaches, but never reaches, a single null (π radians) far beyond the horizon. [Alg. 4.50]

NOTE: The entire exercise of utilizing plane wave calculation near the transmitter site has been all-but negated by this compression of the cycle, apparently used to attempt to make the results of the plane wave calculations act like a cumulative, contiguous, non-cyclical curve distribution, so a pseudo-curve of plane-wave loss can be calculated. This compression all but eliminates the presentation of the fresnel zone nulls in the reception pattern, and forces an early entry into the post-fresnel-zone fade to a null at a far distance. This we will fix.

We do have to utilize a manipulation, as the computer computation can only handle ± half of a cycle in polar notation, and a full 2π radian rotation in polar notation computes the addition to null swing effect of two wavelength-difference cycles. We will use 0 to π/2 radians (0 to 90°) to represent an in-phase to an out-of phase swing of ½ of a wavelength-difference cycle (π radians), and then use a rocking-horse calculation that rocks back from π/2 to 0 radians during the second half of the phase rotation cycle, to mimic the action of the negative phase polarity half of each wavelength-difference cycle:

Line (original):

```
if (q>1.57)
q=3.14-2.4649/q;
```

Replacing this with:

Add the line: q-=floor(q), which converts the value of q from a multiple of wavelengths to a percentage (0 for an equal number of wavelengths, where the 2 rays add, through .5, where we have an out of phase cancellation null, and then almost to 1, where we again have an addition), of one rotational cycle; this is followed by an if statement, which converts q to radians/2 if q is less than ½ of a rotation, leaving δ with a range of 0 to π/2 radians (0 to 90°). If q is more than ½ a rotation, the line following the else statement converts q to radians/2 and rocks back from π/2 to 0 radians during the second half of the rotation.

Lines (new):

```
q-=floor(q);
if (q<0.5)
{
        q*=3.141592654;
else
        q=(1-q)* 3.141592654;
}
```

9. The subroutine ***abq_alos*** is called with input (complex<double>(cos(q),-sin(q))+r)).

The subroutine ***abq_alos ,*** in its entirety, consists of:

```
double abq_alos (complex<double> r)
{
return r.real()*r.real()+r.imag()*r.imag();
}
```

The subroutine ***abq_alos*** returns (r.real()*r.real()+(r.imag()*r.imag(), the sum of the squares of the values of the real and phase components of r. This subroutine takes as input the real and imaginary parts of a complex number expressed in Cartesian co-ordinates, in (a +bi) form, and outputs the square of the value of the polar co-ordinate r, the polar vector (as in the r in r(cos θ + sin θ), a complex number expressed in polar co-ordinates).

The equation we are working from is:

$$At = -10 \log_{10} |1 + R_e e^{i\delta}| \qquad \text{[Alg. 4.51]}$$

Let's take a close look at the $R_e e^{i\delta}$ term. The complex exponential notation here can be confusing; especially since R_e is now a complex number. It can be more easily understood if we write it as the multiplication of two complex numbers:

$$R_e e^{i\delta} = (r_e e^{i\theta})*(1e^{i\delta})$$

In complex numbers, the exponent does not indicate an exponent to the base of anything; it is, instead, used to indicate and define a complex number as consisting of the polar vectors r_1 and r_2 (in this case, r_e and 1), and the exponent $e^{i(\theta = \text{phase angle})}$.

From the mathematical textbooks, we find that the multiplication of complex numbers can be stated in exponential notation as:

$$z_1 z_2 = (r_1 e^{i\theta 1})*(r_2 e^{i\theta 2}) = r_1 r_2 e^{i(\theta 1+\theta 2)}$$

where θ_1 and θ_2 are any arguments of, respectively, z_1 and z_2. Then, in polar coordinates form, this can be stated as:

$$z_1 z_2 = r_1 r_2 (\cos(\theta_1 + \theta_2) + i \sin(\theta_1 + \theta_2))$$

Or, in this case:

$$R_e e^{i\delta} = (r_e e^{i\theta})*(1e^{i\delta}) = r_e (\cos(\theta_r + \delta) + i \sin(\theta_r + \delta))$$

At least, that is what the original coder attempted to do. But this only works correctly in polar co-ordinates. The problem here is that abq_alos, as a c++ subroutine, is set only to work with Cartesian co-ordinate inputs, even though its

output is the square of the polar vector. Since it is only a phase argument, the vector value, $r_{\delta,}$ of δ is always one, therefore, its Cartesian and polar co-ordinate values are the same. However, the vector value of r_e varies between zero and one; so its Cartesian and polar co-ordinate values are only the same when $r_e = 1$. The argument r is stated as r= a +bi, (Cartesian) not r(cos θ +sin θ) (polar). To work in c++ in polar co-ordinates, you must tell the computer you have switched from Cartesian to polar by using the argument: **polar**, not found in abq_alos.

So the addition in abq_alos, found in the ITM 1.2.2, gives faulty results when r_e is not equal to 1.

The usual arithmetic operators, including *, are overloaded, (in c++ - speak), for complex numbers (i.e. they work in complex numbers), so we can change the + to a * in the second call to abq_alos, and attempt to obtain more correct results using:

```
complex<double>(cos(q), sin(q))*r)
```

This will correct the function to match the Algorithm, by changing it to multiply the reflection coefficient, r, by the phase exponent σ, and add the 1 in the equation, [Alg. 4.51], below, to the real term. So, from ***abq_alos***, we get: $((\cos(q)*r).\text{real}())^2 + ((\sin(q)*r).\text{real}())^2$.

Note the – sign in the r.imag term of abq_alos, (– sin δ). In the ITM 1.2.2, this negative sign reverses the phase of δ, representing the 180 degree phase shift (equivalent to adding a ½ wavelength to the reflected path length) found at the reflection point for horizontal polarity, and for vertical polarity below the pseudo-Brewster angle. It does not function with the new version's phase manipulation, and is therefore removed; we will accomplish this adjustment later.

The maximum negative attenuation (ray additive combination) from this calculation would be -6.0 dB, and when the two rays are just enough out of phase for no additive gain, the result must be 0 dB. Determining the depth of the null, however, is a problem. In this case, a correct determination of $R_{e*}e^{i\delta} = 1$ represents a total cancellation of the direct ray (or wavefront) r_1, so the actual attenuation would theoretically approach infinity (in dB).

NOTE: In the old ITMDLL.cpp c++ source code, the 1 in $1 + R_{e*}e^{i\delta}$ is not in the ITMDLL.cpp v. 7 and previous c++ code; the – sign before the sin in the abq_alos function performed this task. Remember that the range of R_e is theoretically zero to one. A logarithm is, however, based on a ratio as an input, and the log of zero does not compute. Also, to start at 0 dB, or no loss, we must start from 0 dB = $\log_{10}(1/1)$. So we add a 1. Leaving it out, in front of a log conversion, is the equivalent of taking $R_{e*}e^{i\delta}$ and making it $(1/ R_{e*}e^{i\delta})$, converting the range of the logarithmic input ratio term from 1 to 0, to be 1 to approaching 1/0, (i.e. 1 to approaching infinity). The old ITMDLL 1.2.2

computer implementation imperfectly used this attribute to calculate the depth of the cancellation nulls. We will also use, but also improve on, this manipulation.

It functions by calculating -10 times the log10 (i.e. -4.343 times the natural logarithm, or log function) of $(R_e e^{i\delta})^2$. In the old implementation, the value of $e^{i\delta}$ is limited to below its normal range of zero to one, such that $R_e + e^{i\delta}$ cannot reach zero or become a negative number, neither of which the c++ log or log10 functions can compute, and normally will not reach higher than one. The range of $R_{e*}e^{i\delta}$ then became approximately 0.0000001 to 1, the modulus was 0.0000001 to 1, and for a log of, say 0.0000001, the result was: -10*-7, (a.k.a. –4.343 * -16.118) for an attenuation of 70 dB, reduced by the difference between the reflection coefficient and 1.0. For a result approaching 1, the result was –10*0, or 0 dB. So it produced results with a range that, at first look, appeared correct, but were only correct when r approached a value of 1.

Having removed the no longer functional negative sign from the second abs_alos calculation, it is necessary to restore the 1+ in [Alg. 4.51] prior to the logarithmic conversion. The squaring in the abs_alos function has removed the polarity sign from the Re function, so we must add a [1-] to invert the function to represent the phase reversal at the reflection point.

After cleaning up the calculations to match the Longley-Rice documentation, including removing the negative sign on the (sin δ), and allowing the range of δ to now represent a full cycle of path wavelength difference on the two-ray calculation, it is found that, even corrected to match At = $-10 \log_{10} |1 + R_{e*}e^{i\delta}|$, [Alg. 4.51], we still do not get good results.

The range of $e^{i\delta}$ is theoretically zero to one, here with a zero representing a deep null cancellation of the direct ray, and a one, an additive solution creating a +6 db gain, the value depending upon the multiple-of-wavelength-related phase, or phase relationship, between the direct and reflected rays. The range of R_e is also zero to one, with a zero having no effect, i.e. resulting in a 0 dB result, and a one enabling maximum potential effect, depending upon the phase, $e^{i\delta}$. The maximum combined range is therefore zero to one, with one representing an additive 6 dB of gain, a zero indicating a null, and a result of .707, equal to the sin of 45 degrees (p/4 radians), indicating no additive or canceling effect. At first, we used an iteration to find what combination of multiplication factor and additive factor (adjustment of gain and bias) would cause the logarithmic function to produce a 0 dB result where the direct and reflected ray signals are just out of phase enough to combine with no apparent gain effect, and to produce a result of 6 dB, when the two signals are in phase. This calculation would then automatically produce the correct null depth when the two signals combine as out of phase as the complex impedances will allow.

The result, of adding .765 and multiplying by .707, to the square root of $(R_e e^{i\delta})^2$, the result of the second abs_alos calculation, is shown on the Gain&Bias tab of

the spreadsheet. But it is interesting to note that the multiplying factor is equal to the value of sin at p/4 radians, equal to squaring the value. So what happens if we iterate the multiplication and additive constant for $(R_e\, e^{i\delta})^2$? On the Gain&Bias tab of the spreadsheet, you can see the result; the multiplication factor is 2, and the bias constant is zero!

Therefore, multiplying $(1- R_e e^{i\delta})^2$ by 2 will give good results when R_e is 1, or near 1. But as R_e approaches 0, for example, near the pseudo-Brewster angle or where Δh is high, the result would approach 6 dB, not 0 dB. By replacing the 2 with $(1+ R_e^2)$, when R_e^2 approaches 0, the calculation will approach 1, therefore, 0 dB, as is correct.

alosv is then set to be equal to At(d), the wavefront-tracing line of sight attenuation at a distance, *d*. Here, $((\cos(q)*r).\text{real}())^2 + ((\sin(q)*r).\text{real}())^2$ represents $(R_e e^{i\delta})^2$, or the square of the $R_e e^{i\delta}$ term in the equation:

$$At = -20 \log_{10} |1 + R_e e^{i\delta}| \qquad \text{[Alg. 4.51]}$$

The $|1 + R_{e*} e^{i\delta}|$ in the equation above, is the modulus of $1 + R_{e*} e^{i\delta}$. A modulus is a defined term for taking the square root of the square of a complex number, equivalent to the abs, or absolute value function in c++. Since the square of a negative number is a positive number, and a square root calculation primarily reports out only the positive result (there are two answers to a square root of x calculation; $x^{1/2}$ and $-x^{1/2}$), then the modulus is essentially the absolute value of the complex number. In short, it eliminates the negative sign, if it had one. Note that abq_alos squares the values of its input, performing the first half of the modulus function. To perform the second half of the modulus function, the log function is multiplied by ½, which, when passed across the log function, takes the square root of $(1+R_e e^{i\delta})^2$.

So why is the log multiplication factor 10, instead of 5? R_e has been derived from formulas for optic reflection coefficients converted to electric field coefficients, i.e. voltage. How do we convert a ratio to a logarithm with regards to voltage? By multiplying the result of the logarithm by 2*10 = 20, for voltage decibels, instead of 10, for power decibels.

The 2 *10 on the left side of the log function shown in Alg. 4.51, at first does not appear to be reflected in the old ITMDLL.cpp code, and is not explained in the documentation. The reason it is not visible is that it is cancelled out by the factor of ½ that takes the square root of $(1+R_{e*} e^{i\delta})^2$.

This equation is equivalent to the equation found in [ITS67 3.2], with modifications.

The R_e we have here, in the c++ ITM code, represented by r, has been modified upstream. It became a complex number when calculated from Zg; we have

corrected it to "center" at 0.5, and have multiplied it by a exponential number that attenuates the maximum effect that can be obtained by the action of the reflected signal combining with the direct signal in a ray-tracing calculation, for this location in this example, given the frequency, terrain roughness, and grazing angle.

So the line becomes:

```
alosv=(-10*log10(1- (abs( r ))*(1.0-(abq_alos(complex<double>(cos(q),sin(q))+r)));
```

Then, on checking the spreadsheet, we find an interesting thing. The value of r is equal to the computation of abq_alos(complex<double>$(\cos(q)*r)^2 + ((\sin(q)*r))^2$!

What is happening? Remember that the output of abq_alos is the square of the polar co-ordinate vector value of r, (using the pythegorian theorem, $a^2 + ib^2 = r^2$), here correctly multiplied by the value of the phase function, which is, by trigonometric definition, $(\cos\theta)^2 + (\sin\theta)^2 = 1$. So it is $r * 1 = r$, and only a part of the value we need; one we already have as *r*.

We, and the original coders, have been misled by the use of exponential notation in:

$$At = -20\log_{10} |1 + R_e e^{i\delta}| \qquad \text{[Alg. 4.51]}$$

When a much clearer and correct rendition would have been:

$$At = -20\log_{10} |1 + R_e(\cos\delta)| \qquad \text{[Alg. 4.51 CORRECTED]}$$

With the understanding that R_e represents the polar notation vector value already calculated as *r*, and that the result we need here is the "real" value of r along the polar "x" or "real value" co-ordinate line, described in polar notation as $r(\cos\theta)$.

However, we still want to make use of the logarithmic manipulation to calculate the null depths. So we modify the second ***abq_alos*** call to provide us with just the squared polar vector "r" representation of the reflection coefficient. R cannot exceed 1.0, and the square of 1.0 is 1.0, so we limit *re*; to avoid a "not-a-number" error from the log10 function attempting to compute the $\log_{10}$ of zero (1-1), we limit Re to equal a maximum of .9990, to eliminate rounding errors, and our c++ code line becomes:

Line new:

```
re=abq_alos (r);
re=mymax(0.999.0,re)
alosv=-10*log10((1+ re)*(1.0-re*cos(q)*cos(q)));
```

The results: For a recommended average terrain value of Δh = 90 m., (as per the Guide, Table 2, page 8), a transmit effective height of 300 meters, and a receive height of 9

meters, with the computations corrected, and with the Δh(d) and s equations as provided in the ITMDLL.cpp, the resultant loss due to ray-path effects is essentially zero out to 40 km., before adjustment for when the direct ray clutter loss at low grazing angles starts to equal the reflection clutter loss. Therefore, in most cases, it effectively exhibits almost no effect, and the original L-R code improperly relies on diffraction calculations to provide the main loss-in-addition-to-free-space effect in the line-of-sight range.

We can verify this by setting the transmit effective height to 1200 meters, receive height to 10 meters, reflection height to 1 meter, frequency to 100 MHz, terrain roughness factor to the U.S. average of 90 meters, and distance to 20 meters. Compare the results (zero dB ray-tracing loss past 40 km) with the 1200 meter curve on Figure 1 of the International Telecommunications Union Recommendation P.1546-2, (ITU-R P.1546-2) where we see a straight line, less than a dB below the free space loss line, extending from 1 km to 20 km. So ray-tracing effects and losses are primarily found over smooth, or nearly smooth earth, and through clutter at a distance. Over average irregular terrain, and beyond free space loss, a different factor takes over as the primary cause of received signal loss. We will refer to this as Clutter, which represents the loss due to absorption and scattering-to-absorption by the ground clutter layer, by absorption of signal by irregular terrain, and all other related sources of near-ground absorption (conversion to heat) of the signal.

10. The original Longley-Rice computer implementation, the ITMDLL.cpp, attempted to account for this loss by mixing in diffraction calculations performed at the horizon and at two points in the line of sight range, and adding these to the ray-tracing calculations in a weighted manner to create a curve calculation, which was then solved at the distance d. This is incorrect, and a discredited methodology. Diffraction calculations are proper to use only at and beyond where the signal grazes the earth at a mutual horizon, or after an obstruction.

    For this corrected subroutine, these calculations have been replaced in the line of sight range using the line of sight equations from Shumate's Approximations, as set of Clutter attenuation approximation equations deterministically derived from ITU-R P.1546-2 Figures 1, 9, and 17. To do this, we call subroutine ***ssalos***.

    The ITU-R P.1546-2 (P.1546-2) Shumate's Approximations equations require as input:

    d path distance from transmitter to receiver, in kilometers. Input as d.
    h_1 the transmitter antenna height in meters AGL, for which we use *prop.tgh*
    h_2 the receiver antenna height in meters AGL, for which we use *prop.he*[1], the effective receive height
    f frequency in MHz, which we will calculate from the wave number, obtained from *prop.wn*
    pol polarity of transmitted signal, (0=h, 1=v, 2=cp), derived from input pol, obtained as prop.ptx

and preset coefficient values:

CH clutter height, the average clutter canopy top height in meters. For compatibility of the results with (P.1546-2), use 25.3 meters as input clutter_height. Arrives at ***saalos*** as *prop.cch*.

η_s reflectivity coefficient of the atmosphere near the surface of the ground (substituted for the value near the top of the clutter canopy). Obtained as prop.ens, from ITM input eno_ns_surfref. Customary value: 301 N-units.

η_{cc} reflectivity coefficient of the average clutter layer at the canopy top, as derived from (P.1546-2). Obtained as prop.encc, from new ITWOM input encc_ncc_clcref. Customary value: 1000 N-units.

11.

The transmitter height AGL is set for saalos with prop.tgh=prop.hg[0], and then an if statement is initiated; if prop.dh, the delta h terrain irregularity factor is greater than 15.3 meters, and if prop.hg[1], the receiver height AGL, is less than the clutter canopy height, then:

subroutine ***saalos*** is called with inputs:

double d

prop_type & prop array with constants, including:

prop.tgh[0]
prop.hg[1]
prop.wn
prop.ptx
prop.ens
prop.encc
prop.cch

prop_type & propa

And returns *saalosv*, the Radiative Transfer Engine attenuation from clutter, which is loaded into alsov, replacing the resulting attenuation from the two-ray computations with results from the RTE approximations:

New line: alosv=saalos(d, prop, propa);

12. The subroutine ends; the result, *alosv*, representing the additional attenuation due to line of sight losses in addition to the free space loss, is returned to the calling subroutine, ***lrprop***:

Line new: return alsov;

}

SUBROUTINE ASCAT: A functional explanation, by Sid Shumate.

Last Revised: July 14, 2007.

Attenuation from Scatter subroutine, A_{scat}.

Note: Used with both point-to-point and area modes. Called by ***lrprop.*** Calls ***mymin, mymax, h0f,*** and ***ahd.***

Descriptions derived from the Irregular Terrain Model description by George Hufford, 2002, (ITMD), compared to the ITM.cpp prepared by J. D. McDonald and John Magliacane for compilation on unix and linux systems. "Line" numbers refer to the ITM.cpp as line numbered by Bloodshed Software's DevC++ print function. "Alg" numbers refer to the algorithm formula in "The ITS Irregular Terrain Model, version 1.22, the Algorithm" by G. A. Hufford, 1995 (the Algorithm). "ITS67" numbers refer to the algorithm formulas in "ESSA Technical Report ERL 79-ITS 67, Prediction of Tropospheric Radio Transmission Over Irregular Terrain, A Computer Method – 1968" by A.G.Longley and P.L.Rice. "TN101" numbers refer to the formulas in "National Bureau of Standards Technical Note 101; Transmission Loss Predictions for Tropospheric Communications Circuits, Volumes I and II," as revised January 1, 1967. (NBS TN101, or TN101).

From ITMD Sections 22, 23, and 24:

The function ***ascat*** finds the "scatter attenuation" for the path distance *d*. It uses an approximation to the methods of NBS TN101 with checks for inadmissible situations. For proper operation, the larger distance ($d = d_6$ must be the first called. A call with $d = 0.$ sets up initial constants.

From the Algorithm, Section 4.3.1:

Computation of this function uses an abbreviated version of the methods described in Section 9 and Annex III.5 of NBS Tech Note 101.

Note: THIS SUBROUTINE IS ELIGIBLE FOR UPDATE AND REVISION: On page 11 of the Algorithm, last paragraph, George Hufford stated:

"A difficulty with the present model is that there is not sufficient geometric data in the input variables to determine where the crossover point is. This is resolved by assuming it to be midway between the two horizons."

This statement and concept should be reviewed to determine if the "sheer magnitude" of data available in today's terrain databases is adequate to implement a more accurate geometric determination of the crossover point.

For now, here is how the current version of the ITM.cpp works:

Note: Not defined in the Algorithm section 4.3.1 below, the k in [Alg. 4.62] is the wave number, which is equal to the frequency in MHz divided by 47.7.

From the Algorithm, with clarifications:

"First, set:

$$\theta = \theta_e + \gamma_e^s \qquad \text{[Alg. 4.60]}$$

$$\theta' = \theta_{e1} + \theta_{e2} + \gamma_e^s \qquad \text{[Alg. 4.61]}$$

$$r_j = 2 * k * \theta' * h_{ej} \qquad \textit{for } j = 1, 2. \qquad \text{[Alg. 4.62]}$$

If both r_1 and r_2 are less than 0.2, the function A_{scat} is not defined, (or is infinite).

Otherwise, we put

$$A_{scat}(s) = 10 * \log(k*H*\theta^4) + F(\theta_s, N_s) + H_0 \qquad \text{(4.63), i.e. [Alg. 4.63]}$$

Where $F(\theta_s, N_s)$ is the function shown in Figure 9.1 of Tech Note 101, H_0 is the "frequency gain function", and H is 47.7 meters.

The frequency gain function H_0 is a function of:

r_1, defined in {Alg. 4.62]
r_2, defined in {Alg. 4.62]
η_s, the scatter efficiency factor, and
the "asymmetry factor", which we shall here call s_s.

A difficulty with the present model is that there is not sufficient geometric data in the input variables to determine where the crossover point is. This is resolved by assuming it to be midway between the two horizons The asymmetry factor, for example, is found by first defining the distance between horizons

$$d_s = s - d_{L1} - d_{L2} \qquad \text{[Alg. 4.64]}$$

whereupon

$$s_s = (d_{L2} + d_s/2) / (d_{L1} + d_s/2) \qquad \text{[Alg. 4.65]}$$

There then follows that the height of the crossover point is

$$z_0 = (s_s * d * \theta') / (1 + s_s)^2 \qquad \text{[Alg. 4.66]}$$

[Ed. where
d is the total path distance
θ' is the angular distance, defined in [Alg. 4.61], a.k.a. θ_{oo}, as shown on Figure 6.1 of TN101 on page 6 – 8.]

and then

$$\eta_s = (z_0 / Z_0) * [1 + (0.0.031 - N_s * 2.32 * 10^{-3} + N_s^2 * 5.67 * 10^{-6})e^{-(z0 / Z1)^{\wedge}6}]$$

(4.67), i.e. [Alg. 4.67]

where

Z_0 = 1.756 km or 1756 meters
Z1 = 8.0 km or 8000 meters
[Ed. (and N_s is the surface refractivity of the atmosphere, a.k.a *ens* or *prop.ens)]*

The computation of H_0 then proceeds according to the rules in Section 9.3 and Figure 9.3 of Tech Note 101.

The model requires these results at the two distances $s = d_5, d_6$ described above. One further precaution is taken to prevent anomalous results. If, at d_5, calculations show that H_0 will exceed 15 dB, they are replaced by the value it has at d_6. This helps keep the scatter mode slope within reasonable bounds."

Discussion:

TN101 defines "launch angles" for the signal path line as it leaves the transmit antenna toward the receive antenna, and from the receive antenna as received from the transmit antenna. These launch angles are defined in Section 6.4 of TN101, and are designated as θ_{et}, the angular elevation of the transmit horizon ray, and θ_{er}, the angular elevation of the receive horizon ray. They are shown on Figure 6.1 of TN101. They are calculated in subroutine ***hzns*** and provided to ***ascat*** as values stored in *prop.the[0]* and *prop.the[1]*.

The equation [Alg. 4.60] can be confusing here, due to its poorly defined use of θ. Alg. 4.60 is attempting to explain that in the general case, a launch angle must be adjusted for the earth's curvature; and is easier to understand if we give examples:

$$\theta_t = \theta_{et} + \gamma_{et}^{s} \qquad \text{[Alg. 4.60a]}$$
$$\theta_r = \theta_{er} + \gamma_{er}^{s} \qquad \text{[Alg. 4.60a]}$$

where

θ_{et} is the transmit site launch angle as shown on TN101 Figure 6.1.
θ_{er} is the receive site launch angle as shown on TN101 Figure 6.1.
s is either the asymmetry factor, or a distance
at the terminals, $\gamma_{e(t,r)}^{s} = d_{L(t,r)} / a = d_{L(t,r)} * gme$,

where

d_{Lt} is the distance from the transmit site to the horizon or obstacle, in meters.

d_{Lr} is the distance from the receive site to the horizon or obstacle, in meters.
gme is the earths curvature, equal to 1/*a* where *a* is the effective earths radius shown in Figure 6.1 of TN101. Here, *a* is in meters, so *gme* is in 1/meters.

In TN101, the angular distance θ is defined as:

$$\theta = \theta_{oo} = d / a + \theta_{et} + \theta_{er} \qquad \text{[TN101 6.14]}$$

where, as described in section 6.4 and on diagram 6.1 of NBS TN101;
d is the total path distance as shown on figure 6.1
a is the effective earth's radius, equal to 1/*gme,* the effective earth's curvature..
θ_{et} is the angle between the horizontal, at the transmitter site, and a line between the transmit antenna and the horizon (or the top of the obstruction). This is the value stored in *prop.the[0]*.
θ_{er} is the angle between the horizontal, at the receive site, and a line between the receive antenna and the horizon (or the top of the obstruction). This is the value stored in *prop.the[1]*.

In [Alg. 4.61], we again find γ_e^s replacing the d/a term in [TN101 6.14].

The angular distance θ_{oo} is the angle, as shown on diagram 6.1 of NBS TN101, between a line starting from the transmit antenna and touching the transmit horizon or the top of the receive obstacle, and a line starting from the receive antenna and touching the receive horizon or the top of the receive obstacle.

TN101 also defines "launch angles" for the signal path line as it grazes the horizon, (or crosses the top of the obstacles). These launch angles, or angular elevation of a horizon ray, are defined as the angle between the horizontal at the horizon (or obstacle), and the signal path line grazing the horizon (or touching the top of the obstacle), and in TN101 are designated as θ_{ot}, the angular elevation of the transmit horizon ray, and θ_{or}, the angular elevation of the receive horizon ray, as shown on Figure 6.1 of TN101. These are calculated using:

$$\theta_{ot} = \theta_{et} + d_{Lt}/a \qquad \theta_{or} = \theta_{er} + d_{Lr}/a \qquad \text{[TN101 6.16]}$$

where
d is the total path distance as shown on figure 6.1, and input to ***ascat*** as input *d*
θ_{et} is the angle between the horizontal, at the transmitter site, and a line between the transmit antenna and the horizon (or the top of the obstruction). This is the value stored in *prop.the[0]*.

θ_{er} is the angle between the horizontal, at the receive site, and a line between the receive antenna and the horizon (or the top of the obstruction). This is the value stored in *prop.the[1]*.
d_{Lt} is the distance from the transmit site to the horizon or obstruction, stored in *prop.dl[0]*.
d_{Lr} is the distance from the receive site to the horizon or obstruction stored in *prop.dl[1]*.
a is the effective earth's radius, equal to 1/*gme,* the effective earth's curvature, the value of which is stored in *prop.gme.*

If the earth is smooth, θ is approximately equal to D_s/a,

where:

"a" is the effective earth's radius (equal to 1/*gme*), and;

D_s (a.k.a. *ds* in the code) is the distance between the transmitter site horizon (or obstacle) location, and the receive site horizon (or obstacle) location.

and where: $D_s = d - d_{Lt} - d_{Lr}$ [TN101 6.17]

In order to properly calculate tropospheric scatter losses, Longley-Rice generates a "path asymmetry factor, identified in NBS TN101 as "s". In order to do this, TN101 starts by defining the angles α_{oo} and β_{oo}. On page 6.8 of TN101, Volume I, Figure 6.1 shows a graphic representation of the two angles. α_{oo} is the angle at the transmit site between a line drawn from the transmit antenna and grazing the horizon or tallest visible obstacle, and a theoretical line drawn directly from the transmit antenna to the receive antenna (passing through the earth for a beyond-the-horizon path). β_{oo} is the same angle from the point of view of the receive antenna.

In NBS TN101, for the general case of irregular terrain, the angles α_{oo} and β_{oo} are calculated using:

$$\alpha_{oo} = (d/2 * a) + \theta_{et} + (h_{ts} - h_{rs})/d \qquad \text{[TN101 6.18a]}$$

$$\beta_{oo} = (d/2 * a) + \theta_{er} + (h_{rs} - h_{ts})/d \qquad \text{[TN101 6.18b]}$$

where:

d is the total path distance as shown on figure 6.1, and input as *d*
a is the effective earth's radius, equal to 1/*gme,* the effective earth's curvature, the value of which is stored in *prop.gme.*
θ_{et} is the angle between the horizontal, at the transmitter site, and a line between the transmit antenna and the horizon (or the top of the obstruction). This is the value stored in *prop.the[0]*.
θ_{er} is the angle between the horizontal, at the receive site, and a line between the receive antenna and the horizon (or the top of the obstruction). This is the value stored in *prop.the[1]*.

h_{ts} is the transmit site antenna elevation, stored in *prop.he[0]*
h_{rs} is the receive site antenna elevation, stored in *prop.he[1]*.

These angles are positive for beyond-horizon paths. To allow for the effects of a non-linear refractivity gradient, α_{oo} and β_{oo} are modified by corrections $\Delta\alpha_o$ and $\Delta\beta_o$

where:

α_o is defined as $\alpha_{oo} + \Delta\alpha_o$ [TN 6.19a]

and

β_o is defined as $\beta_{oo} + \Delta\beta_o$ [TN 6.19b]

To give the angles α_o and β_o, whose sum is the angular distance theta, θ, and whose ratio defines a path asymmetry factor "s".

$$\theta = \alpha_o + \beta_o \qquad s = \alpha_o / \beta_o \qquad \text{[TN 6.19c]}$$

The corrections $\Delta\alpha_o$ and $\Delta\beta_o$ are functions of the angles θ_{ot} and θ_{or} , (see [TN 6.16], and of the distances d_{st} and d_{sr} from each horizon obstacle to the point where the horizon rays cross over. These distances are approximated as:

$$d_{st} = d\,(\beta_{oo} / \theta_{oo}) - d_{Lt}\ , \quad d_{sr} = d\,(\alpha_{oo} / \theta_{oo}) - d_{Lr} \qquad \text{[TN 6.20]}$$

The sum of distances d_{st} and d_{sr} is the distance D_s between horizon obstacles, defined by [TN 6.17]. Over a smooth earth, $d_{st} = d_{sr} = D_s$.

For small θ_{ot} or θ_{or} , no correction $\Delta\alpha_o$ or $\Delta\beta_o$ is required for values of d_{st} or d_{sr} less than 100,000 meters. When both $\Delta\alpha_o$ or $\Delta\beta_o$ are negligible;

$$\theta = \theta_{oo} = \alpha_{oo} + \beta_{oo} \qquad \text{[TN101 6.22]}$$

which is the same as [TN101 6.14], i.e.:

$$\theta = \theta_{oo} = \alpha_{oo} + \beta_{oo} = d / a + \theta_{et} + \theta_{er} \qquad \text{[TN101 6.14 with 6.22]}$$

If either θ_{ot} or θ_{or} is negative, indicating an obstruction taller than the terminal height, then compute:

$$d'_{st} = d_{st} - |\,\alpha * \theta_{ot}\,| \quad \text{or} \quad d'_{sr} = d_{sr} - |\,\alpha * \theta_{or}\,| \qquad \text{[TN101 6.23]}$$

substitute d'_{st} for d_{st} or d'_{sr} for d_{sr}, and read figure 6.9, on page 6-16 of TN101, using $\theta_{ot} = 0$ or $\theta_{ot} = 0$. [needless to say; the compeuter code cannot do this without using an approximation.

If either θ_{ot} or θ_{or} is greather than 0.1 radian and less than 0.9 radian, determine $\Delta\alpha_o$ or $\Delta\beta_o$ for θ_{ot} = 0.1 radian and add the additional correction term

$$N_s (9.97 - \cot \theta_{ot,r}) [1 - \exp (- 0.05 * d_{st,r})] * 10^{-6} \text{ radians}$$

The bending of radio rays elevated more than 0.9 radian above the horizon and passing all the way through the atmosphere is less than 0.0004 radian, and may be neglected.

Referring to the information obtained from the Algorithm above:

The equation for η_s, the scatter efficiency factor, given in [Alg. 4.67], is derived from an equation in TN101:

$$\eta_s = 0.5696 * h_o * [1 + (0.0.031 - N_s * 2.32 * 10^{-3} + 5.67 * N_s^2 * 10^{-6}) \exp(-3.8 * h_o^6 * 10^{-6})]$$

[TN101 9.3a]:

where

z0 is represented by h_o

Z_0, stated as 1,756 meters in the Algorithm, is replaced by 1.7556 kilometers, so 1/1.7556 = 0.5696;

Z1 = 8.0 km or 8000 meters, and $[1/(8.0)]^6 = (.125)^6 = 3.8$

Call inputs:

d	the total path distance
P*rop_type*	
&prop	array with elements
propa_type	
&propa	array with elements

defines private, or local, arguments:

Note: The following four arguments are static doubles:

ad	absolute value of the difference in distance between: the distance from the transmit site to the horizon, and the distance from the receive site to the horizon.
rr	ratio of the higher terminal's effective height, (transmit or receive site antenna) to the lower terminal's effective height
etq	a term in the equation for η_s
h0s	frequency gain function for (s) smooth earth

h0	frequency gain function
r1	transmit site angle calculated in [Alg. 4.62]
r2	receive site angle calculated in [Alg. 4.62]

z0 the height of the crossover point of the horizon or obstacle grazing lines from the terminal antennas, above a line drawn between the two terminal antennas

ss a.k.a. s_s the "asymmetry factor"

et

ett

th *a.k.a.* theta prime, or θ', the combined launch angle calculated in [Alg. 4.61]

q

In this subroutine:

1. An **if** statement is initiated to prepare the initial scatter constants; if *d* is equal to zero, then:
    a. *ad* is set to be equal to the difference in distance, in meters, between: the distance from the transmit site to the horizon, *prop.dl[0],* and the distance from the receive site to the horizon, *prop.dl[1]*.

    b. *rr* is set to be equal to the effective height of the transmit antenna, *prop.he[0]*, in meters, divided by the effective height of the receive antenna, *prop.he[1]*, in meters. At this moment, the argument *rr* represents the ratio of the transmit antenna effective height to the receive antenna effective height.

Line 282:
```
if (d= =0.0)
{
        ad=prop.dl[0]−prop.dl[1];
        rr=prop.he[1]/prop.he[0];
```

2. A second **if** statement is initiated, nested within the first; so if *d* is equal to zero, and if *ad* is less than zero, then:
    a. *ad* is made equal to *−ad;* as a result, the always positive resulting value stored in *ad* will represent the difference between the distances to the horizon from the transmit site and the receive site, measured in meters.
    b. *rr* is inverted, i.e. made equal to 1/*rr*. The argument *rr* then represents the dimensionless, and always positive, ratio of the higher terminal's effective height, (transmit or receive site antenna) to the lower terminal's effective height.

Line 287:
```
if (ad<0.0)
{
        ad=-ad;
        rr=1.0/rr;
}
```

3. The subroutine then continues under the first if statement, if d=0, to:

a. Set *etq* equal to [(5.67e–6∗*prop.ens*–2.32e–3)∗*prop.ens*+0.031].

What is this for? from: [Alg. 4.67]:
$\eta_s = (z_0 / Z_0) * [1 + (0.0.031 - N_s * 2.32 * 10^{-3} + N_s^2 * 5.67 * 10^{-6})e^{-(z0 / Z1)^{\wedge}6}]$;
from this equation, we take the term: $(0.0.031 - N_s * 2.32 * 10^{-3} + N_s^2 * 5.67 * 10^{-6})$;
re-ordering the term, we get: $[(5.67 * 10^{-6} * N_s - 2.32 * 10^{-3})* N_s + 0.0.031]$;
replacing N_s, the surface refractivity of the atmosphere, with the value of N_s a.k.a. *ens*, stored in *prop.ens*, we get: [(5.67e–6∗*prop.ens*–2.32e–3)∗*prop.ens*+0.031]. This term is later used to calculate the value for argument *et* , a.k.a. η_s, at line 321.

NOTE: There is a QUIRK here, AS *etq*, a **static double** argument, ONLY GETS CALCULATED IF D == 0. LRPROP does call ASCAT first with d = 0, at line 808, before calling with d = d6 and d5.

b. Set *h0s, a.k.a.* $H_0(s)$, or the H_0 frequency gain function over (s), smooth earth, equal to: –15.
c. Set *ascatv* equal to 0.0.

Line 293:
```
etq=(5.67e-6*prop.ens-2.32e-3)*prop.ens+0.031;
h0s=-15.0;
ascatv=0.0;
```

If at line 282, d was equal to zero, the program here jumps to line 345, and the program ends by returning a value of 0.0 for *ascatv.*

If at line 282, d was not equal to zero, the program ignores lines 282 to 297, and proceeds to line 298, where:

4. An ***else*** statement follows, and its action affects line 300 to 343.
If *d* is not equal to zero, then the program proceeds to an **if** statement nested within the ***else*** statement. If *h0s* is greater than 15, then *h0* is set to be equal to *h0s*.

Line 298:
```
else
{
    if (h0s>15.0)
        h0=h0s;
```

QUESTION: *h0s* is a **static double** type argument; is its value retained when the subroutine is not online? Does it come from the calling routine ***lrprop***? Need to determine the source of this value. Unless d=0, there is no value preset for *h0s*.

5. A second ***else*** statement at line 302, nested within the first ***else*** statement, provides an alternate path to the **if** statement nested within the first ***else*** statement. This else statement affects lines 304 to 338. So if *d* is not equal to zero, and if *h0s* is less than or equal to 15, then:

a. *th*,(a..k.a.theta, or in TN101 θ_{oo}, or in the Algorithm, *θ'*, theta prime), the angular distance, is set to be equal to:

*prop.the[0]+prop.the[1]+d*prop.gme*; [Alg.4.61] or [TN101 6.14]

where

prop.the[0] is the launch angle from the transmit antenna
prop.the[1] is the launch angle from the receive antenna
d is the total path distance
prop.gme is the effective earth's curvature, which is equal to 1/a, where a is the effective earth's radius

b. *r2* is set to be equal to 2 * *prop.wn* * *th*; this is a part of [Alg 4.62), in that *r2* is momentarily set to be equal to the term 2 * *k* * *θ'*, where:

prop.wn, is the wave number, *k,* equal to the frequency in MHz divided by 47.7, as defined in [Alg. 1.1] ;
th is *θ'*, or theta prime, the angular distance calculated in step a. above, as per [Alg.4.61].

c. *r1* is set to be equal to *r2* * *prop.he[0]*; calculated as per [Alg. 4.62].

where

r2 was determined in step b. above
prop.he[0] is the effective height of the transmit site antenna (in meters)

d. *r2* is then reset to be equal to the value of *r2* from step b. above, multiplied by the value of *prop.he[1]*, the effective height of the receive antenna. At this point, *r1* and *r2* have been calculated as per [Alg. 4.61].

This comes from the equations for the Frequency Gain Function, H_o, in Section 9.2 of TN101. On page 9–3 of TN101, the parameters r_1 and r_2 are defined as:

$$r_1 = 4 * \pi * \theta * h_{te,}/\lambda \quad \text{and } r_2 = 4 * \pi * \theta * h_{re,}/\lambda \qquad \text{[TN101 9.4b]}$$

In TN101, Section 9.2, the dimensions can be said to be either meters or kilometers for the wavelength λ, and for the effective antenna heights, $h_{te,}$ and h_{re}, as long as all are specified in either meters or kilometers; these units of measure cancel out. The angular distance θ is specified in radians in TN101 and in the Algorithm. This is not true in the code. Radians, in mathematics, are usually assumed to be the standard unit of angular measure, so the unit "rad" is customarily omitted, contributing to the confusion in attempting to study Longley Rice as it translates from TN101 to the computer code. In the ITM FORTRAN and c++ code, the angular distance θ is specified in a unique ratio;

the ratio of vertical distance change to horizontal distance change. The units must cancel out (both the numerator and denominator must be specified in the same units). The change in units for θ is not the only difference in r_1 and r_2. In the equation for r used in the code, the wave number is used instead of the wavelength.

The Algorithm, defines the wave number to be that of the carrier or central frequency. It is defined to be:

$$k = (2 * \pi / \lambda) = f / f_o \quad \text{with } f_o = 47.70 \text{ MHz} * \text{meters.} \qquad \text{[Alg. 1.1]}$$

Resorting r_1 and r_2 , we get:

$$r_1 = 2 * \theta * h_{te} * (2 * \pi / \lambda) \quad \text{and } r_2 = 2 * \theta * h_{re} * (2 * \pi / \lambda) \qquad \text{[TN101 9.4b]}$$

Replacing the term $(2 * \pi / \lambda)$ with *k:*

$$r_1 = 2 * \theta * h_{te} * k \quad \text{and } r_2 = 2 * \theta * h_{re} * k \text{, which is the same as [Alg. 4.62]}$$

The distance units, if all in meters, cancel out. So we have now successfully converted from wavelength to wave number. But in the equation [Alg. 4.62], in Section 4.3.1, "The Function A_{scat}." the angular distance θ is still being specified in radians, as becomes clear in Section 6, equations [Alg 6.13 and 6.14}.

How do we convert θ, a.k.a. *th*, to radians? There are 2π radians in a full cycle, or 360°. A radian is defined as the angle subtended at the center of a circle by an arc of circumference that is equal in length to the radius of the circle. Draw this construct on a circle, with one radii of length r on the horizontal plane, and a distance of r on the circumference between the two radii. Now draw a vertical line from the point where the non-horizontal radii touches the circumference of the circle, to a point perpendicular to the horizontal radii, forming a right triangle. The radius then becomes the hypotenuse of a right triangle with an angle, subtended at the center of the circle, between the two radii, of one radian, or 57.2958 degrees. The length of the vertical line is then equal to the sine function of the angle θ, which is equal to the ratio of the length of the vertical line to the length of the hypotenuse of the triangle, which is equal to r. So we can now obtain the length of the vertical line by multiplying sin θ by the hypotenuse length, r. This results in the equation:

$$V = (\sin \theta) * r$$

The length of the horizontal line is then equal to the cosine function of the angle θ, which is equal to the ratio of the length horizontal line of the triangle to the length of the hypotenuse of the triangle, which is equal to r. So we can obtain the length of the horizontal line by multiplying cos θ by the hypotenuse length, r. This results in the equation:

$$H = (\cos \theta) * r$$

We can now obtain the ratio of the vertical length to the horizontal length by dividing the equation for Y by the equation for X. and canceling out the "r" terms:

$$V/H = [(\sin\ \theta) * r] / [\ (\cos\theta) * r\ ] = (\sin\theta)/(\cos\theta)$$

In trigonometry, by definition of the tangent function, tan x = (sin x) / (cos x), so the equation becomes:

$$V/H = (\tan\theta) \text{ in radians}$$

This can be used to convert from the angle in radians or degrees, to the ratio used for θ in the code, but we also need to know how to convert from the vertical-distance-to-horizontal-distance ratio (V/H ratio) used for *th*, to radians, in case we later run into a formula that cannot handle the V/H ratio. For this we use the arctan subroutine function:

$$\arctan(V/H) = \theta \text{ in radians (rads)}$$

We will need these conversion factors later in the subroutine, and in subroutine ***h0f*** when we call it.

Line 298:
```
else
{
        th=prop.the[0]+prop.the[1]+d*prop.gme;
        r2=2.0*prop.wn*th;
        r1=r2*prop.he[0];
        r2*=prop.he[1];
```

6. An **if** statement is nested in the **else** statement at this point. If *r1* is less than 0.2 and if *r2* is less than 0.2, then the function A_{scat} is not defined, (or is infinite), and the subroutine returns 1001.0 as the value of *ascatv*. The subroutine ends early, and returns to the calling subroutine, ***lrprop***.

Line 309:
```
if (r1<0.2 && r2<0.2)
    return 1001.0;  // <==== early return
```

NOTE: Is there an **else** statement missing on line 311? Or does stating "return" get the job done?

7. If *r1* is equal to or more than 0.2, or if *r2* is equal to or more than 0.2, the program continues under the **else** statement, and:

    a. ss, a.k.a. s_s the "asymmetry factor" over smooth earth, is set to be equal to:
        =(*d*-*ad*)/(*d*+*ad*);

    where:

d	is the total path distance
ad	is the absolute value of the difference between the distances to the horizon from the transmit site and the receive site, measured in meters.

Which at first glance, appears to bear no relationship to the equations described in the Algorithm. However, In the Algorithm, the asymmetry factor is found by first defining the distance between horizons

$$d_s = s - d_{L1} - d_{L2} \qquad \text{[Alg. 4.64] also [TN101 6.17]}$$

From TN101, page 6-6, it states:

"The sum of d_{st} and d_{sr}, the distances from each horizon obstacle to the crossover of horizon rays, is the distance D_s (this D_s is the same as d_s in the Algorithm and c++ code). Over a smooth earth $d_{st} = d_{sr} = D_s / 2$.

The Algorithm states: "A difficulty with the present model is that there is not sufficient geometric data in the input variables to determine where the crossover point is. This is resolved by assuming it to be midway between the two horizons." Therefore, from this assumption, $d_s/2$ is equal to the distance from each horizon to the crossover point.

whereupon

$$s_s = (d_{L2} + d_s/2) / (d_{L1} + d_s/2) \qquad \text{[Alg. 4.65]}$$

where:

d_{L2}	is the distance from the receive site to the horizon
d_{L1}	is the distance from the transmit site to the horizon
d_s	is the distance between the transmit horizon and the receive horizon

And, $d_s = d - d_{L2} - d_{L1}$, the same as: [Alg. 4.64] also [TN101 6.17]

Or, restated; d, the total path distance, equals the sum of $d_{L2} + d_{L1} + d_s$

Therefore, $d_{L2} = - d_{L1} - d_s + d$, and $d_{L1} = - d_{L2} - d_s + d$,

So, substituting into [Alg. 4.65] we get:

$$s_s = (-d_{L1} - d_s + d + d_s/2) / (- d_{L2} - d_s + d + d_s/2)$$

by reordering: $s_s = (d - d_{L1} - d_s + d_s/2) / (d - d_{L2} - d_s + d_s/2)$

by consolidating: $s_s = (d - d_{L1} - d_s/2) / (d - d_{L2} - d_s/2)$

by multiplying both the numerator and denominator by 2:

$$s_s = (2d - 2d_{L1} - d_s) / (2d - 2\, d_{L2} - d_s)$$

and substituting $d - d_{L2} - d_{L1}$ *for* d_s

$$s_s = (2d - 2d_{L1} - d + d_{L2} + d_{L1}) / (2d - 2\, d_{L2} - d + d_{L2} + d_{L1})$$

by consolidating: $s_s = (d - d_{L1} + d_{L2}) / (d - d_{L2} + d_{L1})$
by reordering: $s_s = [d - (d_{L1} - d_{L2})] / [d + (d_{L1} - d_{L2})]$

Since *ad* has been made equal to the absolute value of prop.dl[0]–prop.dl[1], where prop.dl[0] = d_{L1} , and prop.dl[1] = d_{L2} , then the value of ad is equal to the term ($d_{L1} - d_{L2}$), and equations [Alg. 4.64 and 4.65] merge to create:

$$s_s = [d - (ad)] / [d + (ad)]$$

Therefore the value of *ss*, which represents s_s the "asymmetry factor" over smooth earth, is set to be equal to: (*d-ad*)/(*d+ad*).

An interesting point here is that *ad* at this point has been set to be the absolute value of *ad* in step 2 above. This refers to the procedure mentioned in the last paragraph on page 6-7 of TN101, where it states:

"Many of the graphs in this a subsequent sections assume that s is $\leq$ 1 (Ed. here TN101 is referring to "s" as the "asymmetry factor). It is therefore convenient, since the transmission loss is independent of the actual direction of transmission, to denote as the transmitting antenna whichever antenna will make s less than or equal to unity."

By using the absolute, or always positive, value of *ad,* we make sure that *ss* will be less than 1, as the numerator will always be smaller than the denominator.

Line 312: ss=(d-ad)/(d+ad);

b. in the next step, q is set to be equal to the ratio of rr/ss, where:
The argument *rr* represents the dimensionless, and absolute, (i.e. always positive), ratio of the higher terminal's effective height above ground level, (transmit or receive site antenna) to the lower terminal's effective height above ground level.
ss is the asymmetry factor over smooth earth.

Line 313: q=rr/ss;

c. the value of ss (a..k.a. s_s), the asymmetry factor, is then set to be no less than 0.1;

Line 314: ss=mymax(0.1,ss);

In section 9.2 of TN101, "The Frequency Gain Function, H_o " it states:

"For the great majority of transhorizon paths, s is within the range $0.7 \leq s \leq 1$. The effect of very small values of s, with $\alpha_o << \beta_o$, may be seen in figures III.15 to III.19,

which have been computed for the special case where effective transmitting and receiving antenna heights are equal."

The effect shown in these charts is that s (or for smooth earth, s_s or ss), has a minor effect for $0.7 \leq s \leq 1$. There is a greater effect as s becomes lower in value, i.e. as the asymmetry increases. The above action limits the maximum effect to be that obtained at an asymmetry ratio of 10 to 1, i.e. s = 0.1.

d. The value of *q* represents the ratio of the dimensionless, and absolute, (i.e. always positive), ratio of the higher terminal's effective height above ground level, (transmit or receive site antenna) to the lower terminal's effective height above ground level, divided by *ss*, the asymmetry factor over smooth earth (whose range has not yet been limited, and has been set to always be a positive value). Here, the range of *q* is limited, so that the value of *q* can be no less than 0.1, and no more than 10.0.

Line 315: q=mymin(mymax(0.1,q),10.0);

e. *z0*, the height of the crossover point, is calculated. The Algorithm states:

"There then follows that the height of the crossover point is:

$$z0 = (s_s * d * \theta')/(1 + s_s)^2 \qquad \text{[Alg. 4.66]"}$$

where

s_s is the asymmetry factor for smooth earth, = *ss*
d is the total path distance, in kilometers
θ' is theta prime, the angular distance; = *th*

A form of this equation is also found in [TN101 9.3b], where units are in kilometers.

However, this leaves out a lot of explanation. From TN101 Section 9.2, where *z0* is referred to as h_o, it is defined as the height of the crossover point *as referenced to a direct line drawn between the transmit antenna and the receive antenna,* not with reference to sea level or ground level. A visual depiction of h_o is shown in Figure 6.1 on page 6 – 8. In TN101, it is utilized in calculating H_o, the "frequency gain function".

Here, the code calculates *z0* to be equal to: (d-ad)*(d+ad)*th*0.25/d. How did the Irregular Terrain Model code writers get to this equation for *z0,* starting from [TN 101 9.3b] and [Alg. 4.66]?

First, we re-order Alg. 4.66;

$$z0 = (s_s * d * \theta')/(1 + s_s)^2 = (\theta' * d * s_s) * 1/(1 + s_s)^2 \text{ [TN 101 9.3b] and [Alg. 4.66]}$$

Then multiply both sides by $(1 + s_s)$:

$$z0 * (1 + s_s) = \theta' * d * s_s * [1/(1 + s_s)]$$

and then substitute the equation for *ss*, a.k.a. s_s the "asymmetry factor" over smooth earth, *ss* = (*d-ad*)/(*d+ad*), derived and used in Step 7 (a.) above, for the three s_s terms:

$$z0 * (1 + (d-ad)/(d+ad)) = \theta' * d * (d-ad)/(d+ad) * [1/(1 + (d-ad)/(d+ad))]$$

multiplying out the terms in the left hand side of the equation, and reordering the terms in the right hand side of the equation:

$$(z0 + z0 * (d-ad)/(d+ad)) = \theta' * d * [(d-ad)/(d+ad)] * [1/(1 + (d-ad)/(d+ad))]$$

recombining the numerator in the right hand side of the equation:

$$(z0 + z0 * (d-ad)/(d+ad)) = \theta' * d * (d-ad)/[(d+ad) * (1 + (d-ad)/(d+ad))]$$

multiplying out the terms in the numerator in the right hand side of the equation:

$$(z0 + z0 * (d-ad)/(d+ad)) = \theta' * d * (d-ad)/[(d+ad) + (d-ad) * (d+ad)/(d+ad))]$$

the term (*d+ad)*/(*d+ad*) in the right hand side denominator equals 1 (cancels out), and by adding up the terms in the right hand side denominator, we get:

$$(z0 + z0 * (d-ad)/(d+ad)) = \theta' * d * (d-ad)/(d + d + ad - ad) = \theta' * (d-ad) * d / 2d$$

since *d*/*d* = 1, the *d* /2d term in the right hand side denominator equals ½, so:

$$(z0 + z0 * (d-ad)/(d+ad)) = \theta' * (d-ad) * (1/2)$$

by multiplying both sides of the equation by the term (*d+ad*), we get:

$$z0 * (d+ad) + z0 * (d-ad) * [(d+ad)/(d+ad)] = \theta' * (d-ad) * (d+ad) / 2$$

the term (*d+ad)* /(*d+ad*) equals 1; also, by multiplying out the terms on the left side of the equation, and adding up, we get:

$$z0 * (d) + z0 * (ad) + z0 * (d) - z0 * (ad) = 2 * d * z0 = \theta' * (d-ad) * (d+ad) / 2$$

by dividing both sides of the equation by 2 * d, we get:

$$z0\ (2d/2d) = \theta' * (d-ad) * (d+ad) / (2 * 2 * d) = \theta' * (d-ad) * (d+ad) / (4 * d)$$

the term (2d/2d) = 1, and ¼ = 0.25, so the result is:

z0 = th* (*d-ad*)* (*d+ad*) *(0.25)/d, the equation used in the ITM code.

where:

th represents the angular distance, *θ'*, *(*theta prime).

Line 316: z0=(d-ad)*(d+ad)*th*0.25/d;

f. the working variable *temp* is set to be equal to *z*0 divided by 8,000 unless the results equal or exceed 1.7; in which case the value of *temp* is limited to 1.7 (limiting the value of *temp* where *z0* exceeds a ceiling of 13,600 feet).

Line 319: temp=mymin(1.7,z0/8.0e3);

g. the value of temp is then set to the value of temp set in step 7 (f.), multiplied to the sixth power, i.e. *temp* = $(temp)^6$. Now, *temp* represents $(z0/Z_1)^6$, a component of the equation for η_s, where Z_1 = 8,000, except that *temp* is limited to a maximum value of $(1.7)^6$ = 24.138.

Line 320: temp=temp*temp*temp*temp*temp*temp;

h. the value of temp is then used to calculate the value of et to be equal to: (etq*exp(-temp)+1.0)*z0/1.7556e3);

where:

At line 293, in step 3(a.), if d = 0, *etq* was calculated to be equal to [(5.67e–6∗*prop.ens*–2.32e–3)**prop.ens*+0.031].

The full equation we are working toward is:

$$\eta_s = (z_0 / Z_0) * [1 + (0.0.031 - N_s*2.32*10^{-3} + N_s^2 *5.67*10^{-6})e^{-(z0/Z1)^{\wedge}6}]$$

[Alg. 4.67]

where

η_s is the scatter efficiency factor

Z_0 = 1.756 km or 1756 meters

and now where:

temp represents the value of $(z0/Z_1)^6$, limited to a maximum value of 24.138.

etq is equal to the term: $(0.0.031 - N_s*2.32*10^{-3} + N_s^2 *5.67*10^{-6})$.

This shortens the equation for η_s to be: $\eta_s = (z_0 / Z_0)*[1 + (etq)e^{-(temp)}]$.

Replacing Z_0 with 1756 meters and reordering, allows us to clearly see that *et* has been set to be the calculated value of η_s, the scatter efficiency factor:

$$\eta_s = [(etq)e^{(-temp)} + 1]*(z_0 / 1.756e3)$$

Where *temp* has been limited to a maximum value of 24.138.

Line 321: et=(etq*exp(–temp)+1.0)*z0/1.7556e3)

i. The subroutine ***mymax*** is called to set *ett* to be equal to *et* unless *et* is equal to 1 or less; then the value is set to be equal to a minimum value of 1.0.

Line 323: ett=mymax(et,1.0);

From the Algorithm:

"The computation of H_0 then proceeds according to the rules in Section 9.3 and Figure 9.3 of Tech Note 101.

The model requires these results at the two distances $s = d_5$, d_6 described above. One further precaution is taken to prevent anomalous results. If, at d_5, calculations show that H_0 will exceed 15 dB, they are replaced by the value it has at d_6. This helps keep the scatter mode slope within reasonable bounds."

j. here, the subroutine ***h0f*** is called twice; the first time with inputs (r1,ett), and the second time with inputs (r2,ett),

where:

r1 here is defined as twice the angular distance th, times the effective height of the transmitter site in meters, times the wave number in units of 1/meters.

r2 here is defined as twice the angular distance th, times the effective height of the receive antenna in meters, times the wave number in units of 1/meters.

ett is the value of η_s, the scatter efficiency factor, limited to a maximum value of 1.0.

The subroutine ***h0f*** performs a function equal to that stated in TN101 in Section 9.2, on page 9-4, where it states:

"For η_s greater than or equal to 1; Read $H_o(r1)$ and $H_o(r2)$ from figure 9.3; then H_o is

$$H_o = [H_o(r1) + H_o(r2)]/2 + \Delta H_o \qquad \text{[TN101 9.5]}$$

where

$$\Delta H_o = 6 * (0.6 - \log \eta_s) \log s \log q$$

$$s = \alpha_o / \beta_o \quad q = r_2 /(s * r_1)$$ "

If $\eta_s > 5$, the value of H_o for $\eta_s = 5$ is used. The correction term ΔH_o is zero for $\eta_s = 4$, s = 1, or q = 1 and reaches a maximum value, $\Delta H_o = 3.6$

db, for highly asymmetrical paths when $\eta_s = 1$. The value of ΔH_o may be computed as shown."

Since we cannot use the table, the approximation used here, and in subroutine ***h0f***, is described in the Algorithm, section 6, starting with equation [Alg. 6.10], and continuing through [Alg. 6.14], where it states:

"The frequency gain function may be written as

$$H_o = [H_{oo}(r_1, r_2, \eta_s)] + \Delta H_o \qquad \text{[Alg. 6.10]}$$

where

$$\Delta H_o = 6 * (0.6 - \log \eta_s) \log s_s * \log r_2 /(s_s * r_1) \qquad \text{[Alg. 6.11]}$$

and where H_{oo} is obtained by linear interpolation between its values when η_s is an integer.

For $\eta_s = 1,\ldots,5$ we set

$$H_{oo}(r_1, r_2, j)] = \tfrac{1}{2} [H_{01}(r_1, j) + H_{01}(r_2, j)] \qquad \text{[Alg. 6.12]}$$

With $H_{01}(r_1, j)$ equal to:

$$\begin{array}{ll} 10 \log (1 + 24r^{-2} + 25r^{-4}) & j = 1 \\ 10 \log (1 + 45r^{-2} + 80r^{-4}) & j = 2 \\ 10 \log (1 + 68r^{-2} + 177r^{-4}) & j = 3 \\ 10 \log (1 + 80r^{-2} + 395r^{-4}) & j = 4 \\ 10 \log (1 + 105r^{-2} + 705r^{-4}) & j = 5 \end{array} \qquad \text{[Alg. 6.13]}$$

For $\eta_s > 5$ we use the value for $\eta_s = 5$, and for $\eta_s = 0$ we suppose

$$H_{oo}(r_1, r_2, 0) = 10*\log[(1+(2)^{1/2}/ r_1)^2 *(1+(2)^{1/2}/ r_2)^2 *(r_1+r_2)/(r_1+r_2+2*(2)^{1/2} \qquad \text{[Alg. 6.14]}$$

In all of this, we truncate the values of s_s and $q = r_2 /(s_s*r_1)$ at 0.1 and 10."

The equation given for ΔH_o, in the Algorithm,

$$\Delta H_o = 6 * (0.6 - \log \eta_s) \log s_s * \log r_2 /(s_s * r_1) \qquad \text{[Alg. 6.11]}$$

is the same as the equation for found in TN101:

$$\Delta H_o = 6 * (0.6 - \log \eta_s) \log s \log q \qquad \text{[TN101 9.5]}$$

where $q = r_2 /(s * r_1)$ "

The two calls return the value *hofv*, i.e. the values for $H_o(r1)$ and $H_o(r2)$; h0 is then set to be equal to the average value (½ the sum) of the results of the two calls to ***h0f:***

Line 324: h0=(h0f(r1,ett)+h0f(r2,ett))*0.5;

k. the value of ΔH_o is then calculated and added to the value of h0 using an equation for ΔH_o calibrated for meters, instead of the kilometers used in TN101; the call to subroutine ***mymin*** makes sure that the value of ΔH_o is no greater than the value of h0 obtained at line 324, limiting the maximum value of h0 to be equal to the sum of the two returns from the two calls to ***h0f***. This is in accord with TN101, section 9.2, page 9-4, where it states: "If $\Delta H_o \geq [H_o(r1) + H_o(r2)]/2$, use $H_o = [H_o(r1) + H_o(r2)]$."

Line 325: h0+=mymin(h0,(1.38-log(ett))*log(ss)*log(q)*0.49);

l. The subroutine ***FORTRAN_DIM*** is called with inputs (h0, 0.0); the subroutine returns the value of (h0 – 0.0), or h0, if h0 is greater than 0.0; if 0.0 is greater than h0, the subroutine returns zero. Here, this archaic subroutine could be replaced with ***mymax.***

   This is in accord with TN101, section 9.2, page 9-4, where it states: "If ΔH_o would make H_o negative, use $H_o = 0$."

Line 326: h0=FORTRAN_DIM(h0,0.0);

8. A second **if** statement is nested in the **else** statement at this point. If *et*, which holds the value of η_s, the scatter efficiency factor, is less than 1, then:

Line 328: if (et<1.0)
{

a. Here we reuse the working variable *temp*. The value of *temp* is reset to be equal to:

((1.0+1.4142/r1)*(1.0+1.4142/r2));

Line 332: temp=((1.0+1.4142/r1)*(1.0+1.4142/r2));

b. The value of h0 is set to be equal to:

h0 = et*h0+(1.0-et)*4.343*log((temp*temp)*(r1+r2)/(r1+r2+2.8284)).

Step 8(b.), incorporates an interpolation statement, referring to the statement in the Algorithm that: "H_{oo} is obtained by linear interpolation between its values

when η_s is an integer." If et, a.k.a. $\eta_s < 1$, then H_{oo} for ($\eta s = 1$) has been calculated at line 324 and the value of ΔHo for ($\eta s = 1$) is added at line 325, to make h0 equal to the value of H_o for ($\eta s = 1$).
At line 326, h0 is set to be the maximum of h0 as calculated on line 325, or zero. So the term (et*h0) is the value of h0 (for $\eta_s = 1$) multiplied by η_s, representing the portion of h0 for ($0 < \eta s < 1$) interpolated from the value of h0 for $\eta_s = 1$. The term (1- et)*4.343*log((temp*temp)*(r1+r2)/(r1+r2+2.8284)) is the value of h0 (for $\eta_s = 0$) multiplied by (1-η_s), representing the portion of h0 for ($0 < \eta s < 1$) interpolated from the value of h0 for $\eta_s = 0$.

The term (4.343*log((temp*temp)*(r1+r2)/(r1+r2+2.8284)) represents the value of h0 for $\eta_s = 0$, where $H_{o(\eta s = 0)} = H_{oo}(r_1, r_2, 0) + \Delta H_{o(\eta s = 0)}$. TN101, section 9.2, states: "The case $\eta_s = 0$ corresponds to the assumption of a constant atmospheric refractive index." The Algorithm, section 6, states:

"for $\eta_s = 0$ we suppose

$$H_{oo}(r_1, r_2, 0) = 10*\log[(1+(2)^{1/2}/ r_1)^2 *(1+(2)^{1/2}/ r_2)^2 *(r_1+r_2)/(r_1+r_2+2*(2)^{1/2})$$

[Alg. 6.14]"

Since $(2)^{1/2} = 1.4142$, the value of *temp* set in step 8 (a.) is equal to the term $(1+(2)^{1/2}/ r_1)^2 *(1+(2)^{1/2}/ r_2)^2$ in [Alg 6.14], simplifying $H_{oo}(r_1, r_2, 0)$ to be:

$$H_{oo}(r_1, r_2, 0) = 10*\log[(temp)*(temp)*(r_1+r_2)/(r_1+r_2+2.8284)]$$

Which we can easily recognize as being a part of the equation for h0 in step 8 (b.) above. In step 8(a.), by removing the interpolation terms, we can derive that h0 for $\eta_s = 0$, as used in the code, is:

h0 for ($\eta_s = 0$)= 4.343*log((temp*temp)*(r1+r2)/(r1+r2+2.8284)).

Since h0 is equal to: $H_o = [H_{oo}(r_1, r_2, \eta_s)] + \Delta H_o$ [Alg. 6.10]

For $\eta_s = 0$, $[H_{oo}(r_1, r_2, 0)]$ is equal to:

$$H_{oo}(r_1, r_2, 0)=10*\log[(1+(2)^{1/2}/ r_1)^2 *(1+(2)^{1/2}/ r_2)^2 *(r_1+r_2)/(r_1+r_2+2*(2)^{1/2})]$$

[Alg. 6.14]"

replacing the term $[1+(2)^{1/2}/ r_1)^2 *(1+(2)^{1/2}/ r_2)^2]$ in [Alg. 6.14] with (*temp***temp*), we get:

$$H_{oo}(r_1, r_2, 0)=10*\log[(temp*temp)*(r_1+r_2)/(r_1+r_2+2.8284)].$$

This is calculated as per [Alg 6.14], except that the constant multiplier value 4.343 replaces the constant multiplier value 10, as it did for the equations based

on [Alg. 6.13] used in the subroutine ***h0f***, previously called by ***ascat***. Why? I thought the answer was the unusual units of measure of *th*, used in calculating r_1 and r_2 , and I still hold that opinion. But a single change of a constant value, from 10 to 4.343, will not do the job of compensating for the different units properly; as the equations use values of $(r_1)^2$, $(r_2)^2$, $(r_1)^4$ and $(r_2)^4$ multiplied by varying constants.

MAJOR PROBLEM NOTE: ***Therefore, since r_1 and r_2 were calculated with a value of th in the wrong units, it appears to this author that the subroutine will produce errant and erratic results; the replacement of the constant value 10 by 4.343 may have occurred in order to produce results that were somewhat close to the empirical results from the field measurements. The author finds no other solid mathematical basis for the change from 10 to 4.343 in the equations in the code.***

How can this be corrected? By using the equation:

$$\arctan (V/H) = \theta \text{ in radians (rads)}$$

To convert the unit value of th to radians prior to its use in calculating r_1 and r_2, at line 305 of this subroutine; and replacing the constant 4.343 with the correct constant, 10, stated in [Alg. 6.13 and 6.14], in subroutines hof and ascat.

Using a different approach, we try reverse engineering. Solving [Alg. 6.10] for ΔH_o, we derive that:

$$\Delta H_o = H_o - [H_{oo}(r_1, r_2, \eta_s)]$$

therefore $\Delta H_{o(\eta s = 0)} = H_{o(\eta s = 0)} - [H_{oo}(r_1, r_2, 0)]$, and;

$$\Delta H_{o(\eta s = 0)} = [4.343*\log((temp*temp)*(r1+r2)/(r1+r2+2.8284))] - 10*\log[(temp*temp)*(r_1+r_2)/(r_1+r_2+2.8284)]$$

which shortens to become:

$$\Delta H_{o(\eta s = 0)} = (4.343 - 10)*[\log((temp*temp)*(r1+r2)/(r1+r2+2.8284))]$$

and adds up to:

$$\Delta H_{o(\eta s = 0)} = (-5.657)*[\log((temp*temp)*(r1+r2)/(r1+r2+2.8284))]$$

and in the code, ΔH_o is added to $H_{oo}(r_1, r_2, 0)$ by:

```
h0+=mymin(h0,(1.38-log(ett))*log(ss)*log(q)*0.49);
```

WORKING POINT:

Since we cannot use the table, the approximation used here, and in subroutine ***h0f***, is described in the Algorithm, section 6, starting with equation [Alg. 6.10], and continuing through [Alg. 6.14], where it states:

"The frequency gain function may be written as

$$H_o = [H_{oo}(r_1, r_2, \eta_s)] + \Delta H_o \qquad \text{[Alg. 6.10]}$$

where

$$\Delta H_o = 6 * (0.6 - \log \eta_s) \log s_s * \log r_2 /(s_s * r_1) \qquad \text{[Alg. 6.11]}$$

and where H_{oo}
is obtained by linear interpolation between its values when η_s is an integer.
For $\eta_s = 1,\ldots,5$ we set

$$H_{oo}(r_1, r_2, j)] = \tfrac{1}{2} [H_{01}(r_1, j) + H_{01}(r_2, j)] \qquad \text{[Alg. 6.12]}$$

With $H_{01}(r_1, j)$ equal to:

$$\begin{array}{ll} 10 \log (1 + 24r^{-2} + 25r^{-4}) & j = 1 \\ 10 \log (1 + 45r^{-2} + 80r^{-4}) & j = 2 \\ 10 \log (1 + 68r^{-2} + 177r^{-4}) & j = 3 \\ 10 \log (1 + 80r^{-2} + 395r^{-4}) & j = 4 \\ 10 \log (1 + 105r^{-2} + 705r^{-4}) & j = 5 \end{array} \qquad \text{[Alg. 6.13]}$$

For $\eta_s > 5$ we use the value for $\eta_s = 5$, and for $\eta_s = 0$ we suppose

$$H_{oo}(r_1, r_2, 0) = 10*\log[(1+(2)^{1/2}/ r_1)^2 *(1+(2)^{1/2}/ r_2)^2 *(r_1+r_2)/(r_1+r_2+2*(2)^{1/2} \qquad \text{[Alg. 6.14]}$$

In all of this, we truncate the values of s_s and $q = r_2 /(s_s*r_1)$ at 0.1 and 10."

The equation given for ΔH_o, in the Algorithm,

$$\Delta H_o = 6 * (0.6 - \log \eta_s) \log s_s * \log r_2 /(s_s * r_1) \qquad \text{[Alg. 6.11]}$$

is the same as the equation for found in TN101:

$$\Delta H_o = 6 * (0.6 - \log \eta_s) \log s \log q \qquad \text{[TN101 9.5]}$$

where $q = r_2 /(s * r_1)$ "

This is used in lieu of the procedure from TN101 section 9.2(b), where it states:
"For η_s less than 1:

First, obtain H_o for $\eta_s = 1$, as described above, then read H_o for $\eta_s = 0$ from figure 9.5. Figure 9.5b shows H_o ($\eta_s = 0$) for the special case of equal antenna heights. The desired value is found by interpolation: $H_o(\eta_s < 1) = H_o(\eta_s = 0) + \eta_s [H_o(\eta_s = 1) - (H_o(\eta_s = 0)]$."

Since the program cannot read the value from figure 9.5;

$H_o = [H_o(r1) + H_o(r2)]/2 + \Delta H_o$ [TN101 9.5]

Where:

Line 333: h0=et*h0+(1.0-et)*4.343*log((temp*temp)*(r1+r2)/(r1+r2+2.8284));
}

The above steps 8 (a.) and (b.) are used to approximate the results obtained from the procedure found in TN101 section 9.2(b).

This is calculated as per [Alg 6.13], except that the constant multiplier value 4.343 replaces the constant multiplier value 10. Why? I thought the answer was the units of measure of *th*, used in calculating r_1 and r_2 , and I still hold that opinion. But a single change of a constant value, from 10 to 4.343, will not do the job properly; as the formula uses values of $(r_1)^2$, $(r_2)^2$, $(r_1)^4$ and $(r_2)^4$ multiplied by varying constants.

MAJOR PROBLEM NOTE: ***Therefore, since r_1 and r_2 were calculated with a value of *th* in the wrong units, it appears to this author that the subroutine will produce errant and erratic results; the replacement of the constant value 10 by 4.343 may have occurred in order to produce results that were somewhat close to the empirical results from the field measurements. The author finds no other solid mathematical basis for the change from 10 to 4.343 in the equations in the code.***

How can this be corrected? By using the equation:

arctan (V/H) = θ in radians (rads)

To convert the value of *th* to radians prior to its use in calculating r_1 and r_2; and replacing the constant 4.343 with the correct constant, 10, stated in [Alg. 6.13 and 6.14], in subroutines hof and ascat.

9. A third **if** statement is nested in the **else** statement at this point. If both: h0 is greater than 15.0, and h0s is greater than or equal to 0.0, then h0 is set to be equal to h0s;

Line 336: if (h0>15.0 && h0s>=0.0)
h0 = h0s;

10. The second **else** statement then ends, but the first **else** statement is still active, and continues;

    a. *h0s* is set to be equal to *h0*. Here *h0s*, the H_o value for smooth earth, is preset to be equal to the value of *h0*, unless h0 was >15 and h0s > 0, where both would have been set to the value for h0s.
    b. *th* is set to be equal to: *propa.tha* + *d* * *prop.gme*;

    where:

propa.tha	is the total bending angle, set in ***lrprop***;
d	is the path distance
prop.gme;	is the effective earth's curvature.

NOTE: HERE, *th IS RECALCULATED USING propa.tha, which may be derived from prop.the[0] and prop.the[1] in step 3(c.) of* ***lrprop.***.

    c. ascatv is set to be equal to:
    ahd(th*d)+4.343*log(47.7*prop.wn*(th*th*th*th))
    – 0.1*(prop.ens-301.0)*exp(-th*d/40e3)+h0;

    The subroutine ***ahd*** is called with input equal to: (th*d). The subroutine ***ahd*** returns the value of F_0 (D), a.k.a. $F(\theta_d,\ N_s = 301)$, used below in the equation for $F(\theta_d,\ N_s)$.

From Section 4.3.1 , "**The function** A_{scat},"

"we put

$$A_{scat}(s) = 10\log(k * H * \theta^4) + F(\theta_s,\ N_s) + H_o \qquad \text{[Alg. 4.63]}$$

where

$F(\theta_s,\ N_s)$	is the function shown in Figure 9.1 of TN101
H_o	is the "frequency gain function"
H	is 47.7 meters."

also:

k,	the wave number, is the value stored in *prop.wn;*
th	is the angular distance, recalculated in step 10(b.) above;

Figure 9.1 of TN101 is a table; so its function is approximated by the equation listed after III.48 in Annex III, Section 5 – Forward Scatter, where it states:

The function $F(\theta_d)$ may be obtained for any value of N_s, by modifying the value computed for $N_s = 301$:

$$F(\theta_d,\ N_s) = F(\theta_d,\ N_s = 301) - 0.1*(N_s - 301.0)*\exp(-\theta*d/40)$$

Which, with the constant 40 converted to 40,000 to adjust for a change from kilometers to meters as the unit value of *d*, is the same as:

$$F(\theta_s,\ N_s) = -0.1*(\text{prop.ens-301.0})*\exp(\text{-th*d/40e3})$$

where

prop.ens is the earth's surface refractivity, a.k.a. N_s
th is the angular distance, θ, recalculated in step 10(b.) above;
d is the path distance

NOTE: *4.343 is being substituted for 10 again, as discussed above.*

d. The first **else** statement then ends its run.

```
Line 340:    h0s=h0;
Line 341:    th=propa.tha+d*prop.gme;
Line 343:    ascatv=ahd(th*d)+4.343*log(47.7*prop.wn*(th*th*th*th))-0.1*(prop.ens-
             301.0)*exp(-th*d/40e3)+h0;
        }
```

11. The subroutine then ends by returning the value of *ascatv*, the "scatter attenuation" at a distance *d*.

```
Line 346: return ascatv;
```

SUBROUTINE D1THX: A functional explanation, by Sid Shumate.
Last Revised July 27, 2008.

Delta h (experimental) subroutine

Note: Used with point-to-point mode. Called by qlrpfl, mid-routine.
Calls mymin, mymax, assert, zlsql, qtile.

Used to find dh, (a.k.a. delta h) the terrain irregularity factor.

Derived from the Irregular Terrain Model description by George Hufford, 2002, (ITMD), compared to the ITM.cpp prepared by J. D. McDonald and John Magliacane for compilation on unix and linux systems. "Line" numbers refer to the ITM.cpp as line numbered by Bloodshed Software's DevC++ print function. "Alg" numbers refer to the algorithm formula in "The ITS Irregular Terrain Model, version 1.22, the ***Algorithm***" (the Algorithm) by G. A. Hufford, 1995. There are also quotes and references, in the background section and subroutine description, from the Appendices to Tech Note 101, Volume II, from NTIA Report TR-82-100, "A Guide to the Use of the ITS Irregular Terrain Model in the Area Prediction Mode" (NTIA TR-82-100), and from "A manual for ITM, "Irregular Terrain Model", released by the NTIA as itm_man.txt, (ITM Manual).

Computes the terrain irregularity parameter *dh* from the profile elevation array *pfl* between points located at x1 and x2. x1 is defined as a distance from the transmitter site to the start point of a path of elevation points considered; x2 is defined as a distance from the transmitter site to the end point. Both x1 and x2 must be specified in meters.

Please note that the ***qlrpfl*** subroutine, and the ***d1thx, hzns*** and ***z1sq1*** subroutines that are called during ***qlrpfl,*** were intended to be experimental early versions of L-R software. They are still in use today, with few modifications or corrections. The ITM Manual states:

> "It should be noted that the original ITM is silent on many of the details for defining some of the path parameters. This is particularly true of the effective heights HE, and, to some lesser degree, of the terrain irregularity parameter DH. The effective height, for example is defined as the height above the "effective reflecting plane," and in the past the investigator has been urged to use his own best judgement as to where that plane should be placed. The subroutine QLRPFL, in trying to automate the definition of all parameters, has been forced to define explicitly all missing details. It has done this in a way that seems reasonable and in full accord with the intent of the model. One should not, however, conclude that these efforts are completed. Hopefully, better results are obtainable."

Background on Delta H, (Δh), the Terrain Irregularity Parameter

Also described as "Δh, the interdecile range of elevations between the two points x1 and x2". The **interdecile range** is a specific interquartile range; it is computed as the

difference between the 10th and 90th percentiles. It comprises the middle 80% sample of the population: in this case, a set of elevation values derived from the elevation data contained in the *pfl* array. The term x1 is defined as a distance from the transmitter site to the start point; x2 is defined as a distance from the transmitter site to the end point. We find by studying the Δh, or delta h, used in the irregular terrain model, that the concept and calculation does not closely follow the definition of the terrain roughness factor, σ_h, specified in Tech Note 101.

In Tech Note 101, on page 5-9, it states:

> "5.2.2. **The terrain roughness factor, σ_h.** The terrain roughness factor σ_h in (5.1) is the root-mean-square deviation of modified terrain elevations, y_i, relative to the smooth curve defined by (5.16), within the limits of the first Fresnel zone in the horizontal reflecting plane. The outline of a first Fresnel zone ellipse is determined by the condition that:
>
> $$r_{11} + r_{21} = r_1 + r_2 + \lambda/2$$
>
> Where:
>
> $r_{11} + r_{21}$ is the length of a ray path corresponding to reflection from a point on the edge of the Fresnel zone,
>
> $r_1 + r_2$ is the length of the reflected ray for which angles of incidence and reflection are equal.

Norton and Omberg [1947] give general formulas for determining a first Fresnel zone ellipse in the reflecting plane. Formulas are given in annex III for calculating distances x_a and x_b from the transmitter to the two points where the first Fresnel ellipse cuts the great circle plane."

Later, on page 5-13, it states: "the terrain roughness factor σ_h, (in Tech Note 101 here designated by a lowercase sigma sub h), is the root-mean-square deviation of modified terrain elevations relative to the curve y(x) within the limits of the first Fresnel zone in the horizontal reflecting plane. The first Fresnel ellipse cuts the great circle plane at two points, x_a and x_b kilometers from the transmitter. The distances x_a and x_b may be computed using equations (III.18) or (III.19) to (III.21) of annex III."

The root-mean-square is the square root of the average of the squares of a set of numbers. If we have a set of values in an array: $x_1, x_2, x_3, \ldots, x_n$, the root-mean-square can be computed, using a for loop, as the square root of the sum of the squared array values divided by n, or:

$$x_{rms} = ((x_1^2 + x_2^2 + x_3^2 \ldots + x_n^2)/n)^{1/2}$$

Deviation refers to the amount of difference between the value being considered and the arithmetic mean value. One of the most important uses of the root-mean-square is to

determine the *standard deviation* from the arithmetic mean. The standard deviation from the mean is the root-mean-square of the deviations from the mean.

In classical statistics, the formula for calculating the variance of an unknown population variance is:

$$\sigma^2 = \frac{\Sigma(x - \mu)^2}{N}$$

Here the population parameter is abbreviated with the Greek letter sigma in lower case, the mean is a population parameter (mu), and the number of samples is represented with a capital N. The term $[x - \mu]$ represents the difference between the sample value (x) and the mean value μ; this term is the deviation of the sample.

The standard deviation, σ, is simply the square root of the variance, or:

$$\sigma = ((\Sigma(x - \mu)^2)/N)^{1/2}$$

Here we are using a sample of the total population. For calculating statistics of a sample of the population, statisticians indicate that the mean is of a sample population by replacing μ with the arithmetic mean $\bar{x}$, and they decrease the denominator by 1, to account for the percent probability that a wider deviation exists in the population than in the sample of the population. The formula then becomes:

$$\sigma = ((\Sigma(x - \bar{x})^2)/(n))^{1/2}$$

Where $n = (N - 1)$

If the arithmetic mean $\bar{x}$ is equal to:

$$\bar{x} = (x_1 + x_2 + x_3 \ldots + x_n)/n,$$

then the standard <u>deviation</u> s can be computed, using a for loop, as:

$$s = (((x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + (x_3 - \bar{x})^2 \ldots + (x_n - \bar{x})^2)/n)^{1/2}.$$

The terrain roughness factor is the root-mean-square deviation of modified terrain elevations relative to the curve y(x) within the limits of the first Fresnel zone in the horizontal reflecting plane. We can now see how to obtain the root-mean-square deviation of terrain height data taken from a selected part of the path between the transmitter and the receiver; the information is in the pfl array. But what is meant by "modified terrain elevations relative to the curve y(x)"?

In 5.2.1, a Curve-Fit to Terrain, found on page 5-8 of Tech Note 101, it states:

> "A smooth curve is fitted to terrain visible from both antennas. It is used to define antenna heights h'_1 and h'_2, as well as to determine a single reflection point where the angle of incidence of a ray r_1 is equal to the angle of a reflection of a

ray r_2 in figure 5.1. This curve is also required to obtain the deviation of terrain heights used in computing R_e in (5.1).

First, a straight line is fitted by least squares to equidistant heights $h_i(x_i)$ above sea level, and $(x_i)^2/(2a)$ is then subtracted to allow for the sea level curvature 1/a illustrated in figure 6.4. The following equation describes a straight-line h(x) fitted to 21 equidistant values of $h_i(x_i)$ for terrain between $x_i = x_0$ and $x_i = x_{20}$ kilometers from the transmitting antenna. The points x_0 and x_{20} are chosen to exclude terrain adjacent to either antenna which is not visible from the other:"

$$h(x) = \bar{h} + m(x - \bar{x}) \qquad (5.15a)$$

Here, $\bar{h}$ is the arithmetic mean of the elevation heights, i.e. the average terrain height:

$$\bar{h} = ((h_1 + h_2 + h_3 \ldots + h_{20})/21 \qquad (5.15b, 1 \text{ of } 3)$$

$\bar{x}$ is the arithmetic mean of the distances from the transmitter site to the to $x_0 + x_{20}$, i.e. the distance from the transmitter site to the center of the x path:

$$\bar{x} = (x_0 + x_{20})/2 \qquad (5.15b, 2 \text{ of } 3)$$

m is the slope of the line. To simplify the later calculation of the least squares solution, the slope is calculated with reference to an x co-ordinate referenced to the center of the x path. If *n* is the number of elevation points, in this example 21, then (*n*-1) is equal to the number of intervals between the elevations points, equal in this example to 21 – 1 = 20. The center of this path of equidistant intervals is then at i = (*n* – 1)/2, or (21 – 1)/2 = 10. So if we want to start a for loop calculation from the transmitter end of the path, we have to start at the recalibrated x path position, the interval closest to the transmitter site, in this example now equal to (i – 10). So the slope calculation will start at i = 0 and proceed to i = 20, represented in the below calculation by i – 10, therefore calculating from –10 to +10.

$$m = \frac{2 * (h_1(i-10) + h_2(i-10) + h_3(i-10) \ldots + h_{20}(i-10))}{77 * (x_{20} - x_0)} \qquad (5.15b, 3 \text{ of } 3)$$

This is a slightly different notation, but is still the same formula given in 5.15b, 3rd of 3. <u>The derivation of the 2/(77*(77 * (x_{20} - x_0))) set of terms is not explained in Tech Note 101; *and provides a correct answer only if the number of increments equals 21.*</u>

The derivation of a more universal version of this formula, which will work with any number of increments, is shown in the description of the subroutine ***z1sq1,*** which this subroutine, ***d1thx*** calls, in an attempt to perform this least-squares-fit-

to-a-line calculation. Here, we will simply state that the version of the formula for the slope, m, which provides correct results for any number of increments, is:

$$m = \frac{(h_1(i-10) + h_2(i-10)^{-} + h_3(i-10) \ldots + h_{20}(i-10))}{((i_0-10)^2 + (i_1 10)^2 + (i_2-10)^2 \ldots + (i_{20}-10)^2)}$$

And is intended to be incorporated in a revised version of ***z1sq1***, to be named ***zlsq2.***

After 5.15b, Tech Note 101 states:

"Smooth modified terrain values given by

$$y(x) = h(x) - x^2/(2a) \qquad (5.16)$$

will then define a curve of radius a which is extrapolated to include all values of x from x = 0 to x =d, the positions of the antennas."

Here, a represents the effective radius of the earth under the great circle signal path. The $- x^2/(2a)$ term accounts for the effective curvature of the earth; it adds the effective increase in terrain height due to the effective curvature of the earth, to the "flat earth" signal path average terrain line formula. Section 4 of Tech Note 101 describes a method of deriving the effective earth's radius, a, from n_s, the atmospheric refractive index at the surface of the earth, and a_o, the actual radius of the earth.

Note before reading further: The Δh correction term defined by (6.12) in the next quote, mentioned after (5.17), is not the same Δh we have previously discussed in this section. It is only distantly related to, and not the same as, the delta h (Δh) function calculated by, and result *d1thxv* provided by, subroutine ***d1thx***.

Continuing with Tech Note 101 after (5.16). it states:

"The heights of the antennas above this curve are:

$$h'_1 = h_{ts} - h(0), \qquad h'_2 = h_{rs} - h(d) \qquad (5.17)$$

If h'_1 or h'_2 is greater than one kilometer, a correction term, Δh, defined by (6.12) and shown on figure 6.7, is used to reduce the value given by (5.17).

Where terrain is so irregular that it cannot be reasonably well approximated by a single curve, σ_h is large and $R_e = 0$, not because the terrain is very rough, but because it is irregular. In such a situation, method 3 of section 5.1 may be useful."

The Federal Communications Commission's Terrain Roughness Factor

The following is excerpted from the Federal Communications Commission (FCC) Rules and Regulations, as published as Code of Federal Regulations Title 47, Volume 4, Subpart B, FM Broadcast Stations, Sec. 73.313, Prediction of coverage: [CITE: 47CFR73.313, and for the diagrams, 47CFR73.333]:

(f) The effect of terrain roughness on the predicted field strength of a signal at points distant from an FM transmitting antenna is assumed to depend on the magnitude of a terrain roughness factor (h) which, for a specific propagation path, is determined by the characteristics of a segment of the terrain profile for that path 40 kilometers in length located between 10 and 50 kilometers from the antenna. The terrain roughness factor has a value equal to the distance, in meters, between elevations exceeded by all points on the profile for 10% and 90% respectively, of the length of the profile segment. (See Sec. 73.333, Figure 4.)

(g) If the lowest field strength value of interest is initially predicted to occur over a particular propagation path at a distance that is less than 50 kilometers from the antenna, the terrain profile segment used in the determination of terrain roughness factor over that path must be that included between points 10 kilometers from the transmitter and such lesser distances. No terrain roughness correction need be applied when all field strength values of interest are predicted to occur 10 kilometers or less from the transmitting antenna.

(h) Profile segments prepared for terrain roughness factor determinations are to be plotted in rectangular coordinates, with no less than 50 points evenly spaced within the segment using data obtained from topographic maps with contour intervals of approximately 15 meters (50 feet) or less if available.

(i) The field strength charts (Sec. 73.333, Figs. 1-1a) were developed assuming a terrain roughness factor of 50 meters, which is considered to be representative of average terrain in the United States. Where the roughness factor for a particular propagation path is found to depart appreciably from this value, a terrain roughness correction ([Delta]F) should be applied to field strength values along this path, as predicted with the use of these charts. The magnitude and sign of this correction, for any value of [Delta]h, may be determined from a chart included in Sec. 73.333 as Figure 5.

(j) Alternatively, the terrain roughness correction may be computed using the following formula:

[Delta]F=1.9-0.03([Delta]h)(1+f/300)

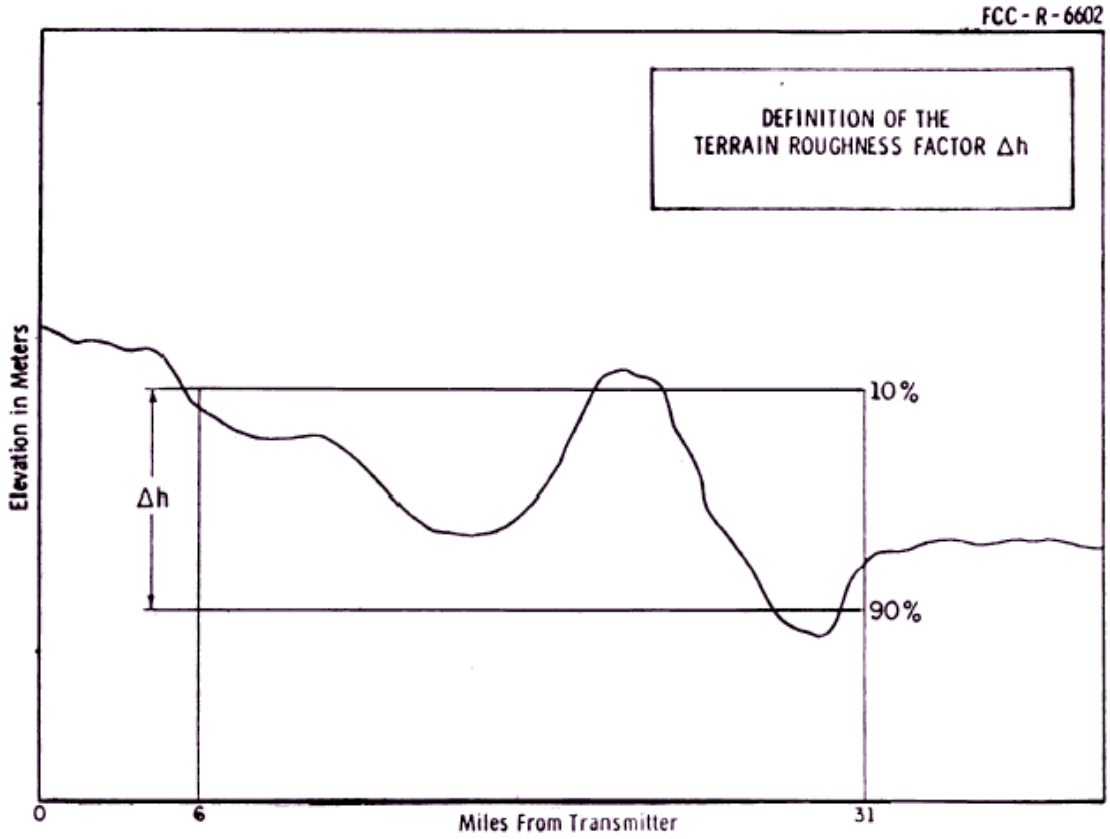
Where:
[Delta]F=terrain roughness correction in dB

[Delta]h=terrain roughness factor in meters
f=frequency of signal in MHz (MHz)

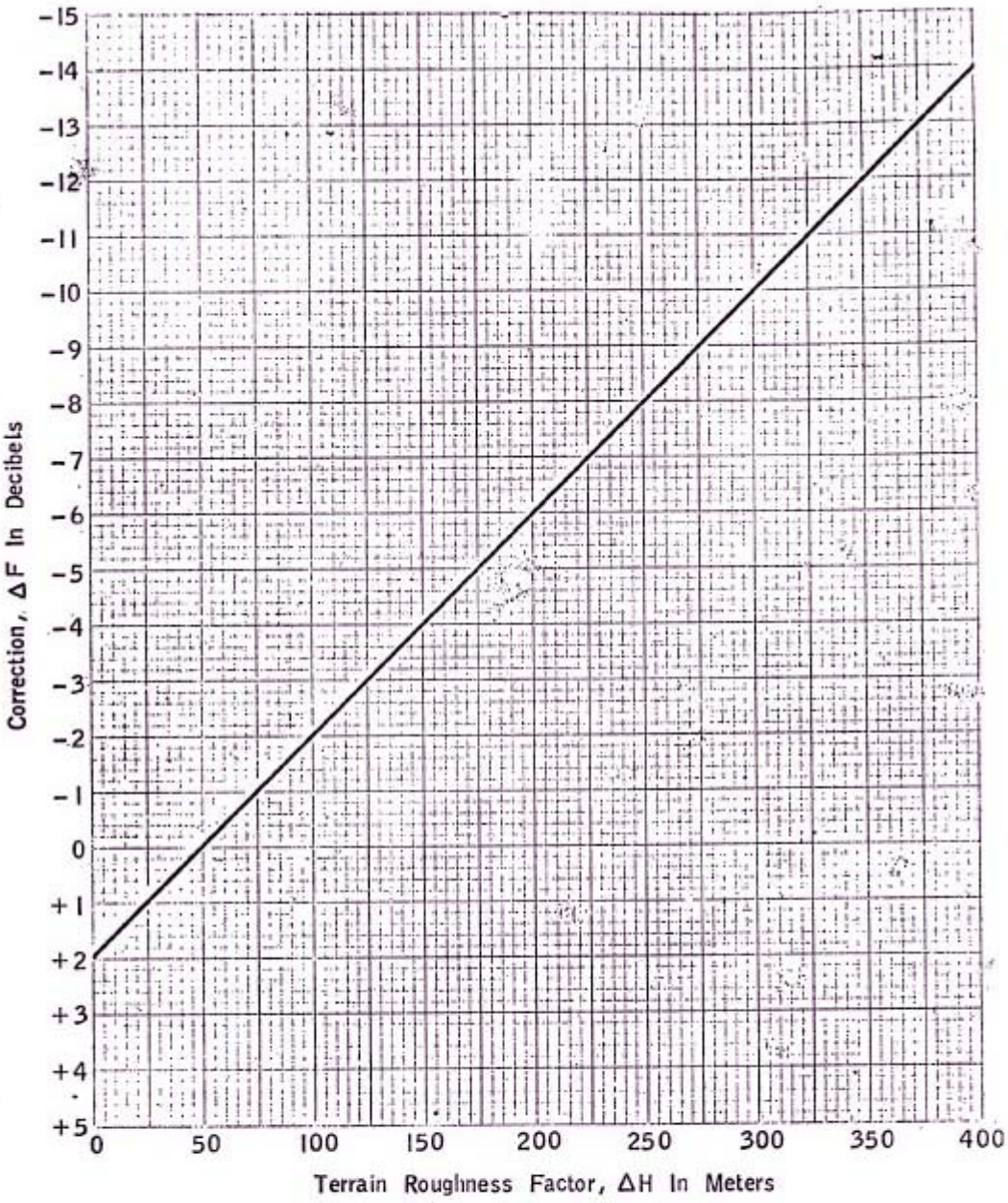
Effective Date Note: At 42 FR 25736, May 19, 1977, the effective date of Sec. 73.313 paragraphs (i) and (j) was stayed indefinitely.

NOTE: The FCC's terrain roughness factor is a related concept, but not the same thing, as the Tech Note 101 terrain roughness factor. In Report No. R-6602, "Development of VHF and UHF Propagation Curves for TV and FM Broadcasting," by Damelin, Daniel, Fine and Waldo, Sept.1966, it states:

"The corrections for terrain roughness are intended for application in estimating median (or average) field strengths over areas where the general character of the terrain is fairly uniform, or where there is no abrupt change in terrain roughness. It is not possible to accurately predict the field strength at any given receiver site."



FCC § 73.333 FIGURE 4
(new)



TERRAIN ROUGHNESS CORRECTION

for use with estimated FM F(50,50) and F(50,10) field strength curves

FCC §73.333 FIGURE 5

(Secs. 4, 5, 303, 48 Stat., as amended, 1066, 1068, 1082 (47 U.S.C. 154, 155, 303))

[28 FR 13623, Dec. 14, 1963, as amended at 35 FR 2591, Feb. 5, 1970; 40 FR 27679, July 1, 1975; 45 FR 28141, Apr. 28, 1980; 48 FR 29508, June 27, 1983; 49 FR 19670, May 9, 1984]

EFFECTIVE DATE NOTE: At 42 FR 25736, May 19, 1977, in §73.333, the effective date of Figures 4 and 5 was stayed indefinitely.

Now, on to the description of the subroutine. From ITMD Section 44:

Call inputs:

pfl	terrain elevation profile array, starting at transmitter site and ending at receiver site, following great circle path, with: pfl[0] =*enp*, the number of increments pfl[1] = *xi*, distance per increment (in meters) pfl[2] = *z(0)*, the transmitter tower base AMSL, or elevation height pfl[[np+2] = *z(np)*, the receive location AMSL, the last elevation.
&x1	x1, input from qlrpfl as xl[0], the start point of the section of the total path to be considered, defined as a distance (measured in meters) from the transmitter site. The value is equal to the lesser of: 15 times the transmitter height AGL, or 1/10 of the distance from the transmitter site to the horizon.
&x2	x2, output from qlrpfl as xl[1], the end point of the section of the total path to be considered, defined as a distance (in meters) from the transmitter site. The value is equal to the lesser of: 15 times the receiver antenna height AGL, or 1/10 of the distance from the receive site to the horizon.

This subroutine declares the following private, or local, arguments:

Here the argument types are noted (int, and double, in this case) because this subroutine incorporates conversions between (int) and (double).

Declared as type integers (int):

np	number of points in pfl elevation array.
ka	equal to 1/10 of (8+length of the section of the total path to be considered), range limited to between 4 and 25.
kb	equal to n- (ka+1).
n	number of intervals between the transmitter site and the end point of a path to be considered in this subroutine only.
k	initially set to be one more than xa.
j	counting variable in a ***for*** loop.

Declared as type double decimal precision (double):

d1thxv calculated terrain irregularity parameter; output from d1thx.

sn one less than n. The total path length, measured in increments, of path n.

xa distance from transmitter to start of test path for s array.

xb distance from transmitter to end of test path for s array.

xc working variable for increment distance of s loop

*s an array of elevations interpolated from terrain elevations in the pfl array, the renamed elev array relayed by the point_to_point (or point_to_point_two) subroutine from the wrap-around input-output software.

This subroutine:

1. Defines *np*, number of points, to be equal to *pfl [0]*, the number of increments in array pfl.

Line 1249: np=(int)pfl [0];

2. Defines *xa* to be equal to xl/*pfl [1], equal to x1(x-one, the distance from the transmitter site to the start point being considered), divided by the increment distance*, initially making double *xa* equal to the number of increments from the transmitter site to the start point of the section of the total path to be considered.

Line 1250: xa=x1/pfl[1];

3. Defines *xb* to be equal to x2/*pfl [1], equal to x2(receiver site) divided by the increment distance*, initially making double *xb* equal to the number of increments from the transmitter site to the end point of the section of the total path to be considered.

Line 1251: xb=x2/pfl[1];

4. Presets d1thxv=0.0

Line 1252: d1thxv=0.0.

5. An ***if*** statement used to check if (xb - xa<2.0). If the number of increments in the considered path is less than 2, indicating a path length too short to calculate d1thxv, it causes the program to exit, returning d1thxv = 0.0 .

```
Line 1254:    if  (xb-xa<2.0)
Line 1255:            return d1thxv;
```

6. Sets *ka* to be equal to 1/10 of (xb-xa+8), or 1/10 of (8+length of the section of the total path to be considered, measured in increments), then sets the range of *ka* to be between 4 and 25 increments.

Since *ka* is declared as an int, the data used to calculate its value must be int; but the declared local xb and xa are of type double. The function, by stating (int) after the equal sign and before the calculation, forces (creates) an output that is (int).

```
Line 1257:    ka=(int)(0.1*(xb-xa+8.0));
Line 1258:    ka=mymin(mymax(4,ka),25);
```

7. Sets value of *n* equal to (10 times ka) – 5; since ka is limited in range to between 4 and 25, then n is limited in range to between 35 and 245 increments.

NOTE: THIS IS AN ARCHAIC, PROBLEM-CAUSING RANGE LIMIT FOR n, FAR TOO SMALL FOR TODAY'S 90 METER (3 ARC-SECOND), 30 METER (1 ARC-SECOND) AND 10 METER (1/3 ARC-SECOND) PATHS, WITH THOUSANDS OF INTERVALS. It limits n to 245 times 90 m, or 22 km, for 3 arc-second database interval sizes, which is acceptable; but it limits the maximum consideration to only 7.35 km for 1 arc-second database all-points intervals, and 2.45 km for 1/3 arc second database all-points intervals. In the ITWOM version, this subroutine has been modified to set ka based on path length of up to 20 kilometers, as per TN101 section 5.2.1. TN101 uses 20 km as a suggested range to include the majority of the middle 80% of terrain for line-of-sight paths. This is a gross simplification that assumes that this middle 80% of terrain can be approximated by a straight line, an assumption that fails for highly irregular terrain.

The FCC regulations for their version of [delta] h specifies a minimum 10 km, and up to a 50 km maximum, limit; the n limit of 7.35 km for 1 arc-second database all-points intervals, and 2.45 km for 1/3 arc second database intervals, are both below the minimum FCC limit for consideration.

In continuing the comparison to the FCC's terrain consideration rules, we find that FCC 47CFR73.7313 indicates that the consideration of terrain roughness extends from n = 10 to 50 km; i.e. the maximum length of the path to be considered was 50 km, the calculation was to be made using terrain information on the signal path starting at 10 km and extending to the receive point, up to a maximum of 50 km. No consideration was to be made for terrain roughness if the path was less than 10 km. The length of n, being 10 times ka less 5, can only compare well with the 10 to 50 meter maximum range of the FCC's delta h determination method, unless ka = 1, where n would = 5; since ka is range-limited to be not less than 4, this is not possible. So let us look at another part of the FCC regulations; where a minimum number of 50 increments over the 40 km consideration path section are required. Converting from paper maps to a digital database, this is 1.2 increments per km, equivalent to an 830-meter database, or roughly equivalent to, and

therefore compatible with, the 900 meter GLOBE 30 arc-second database utilized by the Windows version of the ITM (itm.exe), made available by NTIA.

The maximum and minimum limits for ka, therefore, need to be set based upon the path lengths, not the number of intervals. We only need to reset the maximum limit, as ka is calculated to be far less than xb. It would be in accordance with the FCC guidelines to limit the maximum length of n to 50 km, or 50,005 meters. Distance in meters can be converted to intervals by dividing by pfl[1], the increment width.

On Line 1245; add kmx, between k, and j on the int declaration line.

The units of pfl[1] is meters/increment, so ***d1thx,*** line 1258 should be replaced by:

Line 1258r: kmx=mymax(25,(int)(5000.5/(pfl[1])));
Line 1259r: ka=mymin(mymax(4,ka),kmx);

Line 1258r sets the kmx, or ka maximum, to be the maximum of 25 or the integer value of x2, the path length to the transmitter end of the short path considered, divided by 10 times the interval width, leaving kmx measured in increments. For example, for a 3-arc-second database with maximum 90 m. intervals, the interval width is approximately 79 meters; for a 20 km (12.4 mi.) total path, kmx would be limited to approximately 25 intervals, and for a 1-arc second database would be limited to approximately 76 intervals. Line 1259r replaces 25 with kmx.

END FIRST CORRECTION NOTE. Returning to the discussion of ***d1thx***;

The argument n is set to be equal to 10 times ka, less 5 intervals, in intervals.

Line 1259: n=10*ka-5

8. Sets value of *kb* equal to *n-ka*+1, or equal to 10*ka-5-ka+1 = 9*ka-4.

Line 1260: kb= n–ka+1

9. Sets value of *sn* equal to *n*-1, the total path length, measured in increments, of path n.

Line 1261: sn=n-1,

10. Calls subroutine ***assert*** with input parameters (s=new double[n+2]) !=0). ***assert*** is a standard c function prototype that, in c++, returns an error message and aborts the program if the expression, in this case s=new double[n+2], is equal to zero, i.e. the expression is false. If *n* +2 = 0, the program aborts, returning an error message. Note the use of new to create and allocate memory for the s array,

and of double to declare the s array values to be doubles, where n was declared as an integer (int).

Line 1262: assert (s=new double[n+2]) !=0)

11. Sets value of array value s[0] equal to sn, which is equal to the length, in increments, of the path considered, also = (n – 1).

Line 1263: s[0]= sn

12. Sets value of array value s[1] equal to 1.0

Line 1264: s[1]=1.0

13. Sets value of *xb* to be equal to *(*xb-*xa)/sn*, where:

(xb, prior to step 13) = the number of increments from the transmitter site to x2, the end point of the path considered.
xa = the number of increments from the transmitter to the start point, x1, of the path considered.
sn is the length of the path considered, measured in increments.

So (xb-*xa*) is the length, measured in increments, of the path between x1 and x2. *xb* is redefined to be the length of the path between x1 and x2, divided by the length of sn, the length of the "n" path defined by 10*ka – 6, measured in intervals. This makes *xb* equal to the ratio of: the length of the x1 to x2 path, to the length of the "n" path derived from ka.

Line 1265: xb=(xb-xa)/sn

Warning! This causes computational error of if ka is restricted so that the value of sn causes xb to exceed 1.0.

14. Sets value of *k* equal to xa+1.0, equal to the number of increments from the transmitter to the start point, plus one. The use of (int) after the equal sign allows and forces a calculation using (double) xa, to produce an integer result. The value of xa is truncated, not rounded off, to zero decimal places, so 3.9 and 3.3 both would calculate as 3, i.e. if xa is 3.9, *k* = 3 +1 = 4, if xa is 3.3, *k* = 3 +1 = 4.

Line 1266: k=(int (xa+1.0));

NOTE CHANGE BELOW FOR TEST PURPOSES; REPLACES XA BY XC FOR FOR/WHILE LOOP USE.

15. Here, the original version of the ITM resets the value of double xa by subtracting value of k. However, the value of *xa* is needed later; so the argument *ac* has been

declared and substituted here for the ITWOM. The use of the (double) before k allows and forces a type double result to a calculation incorporating the integer (int) k. Since k = (xa +1), with the value truncated to zero decimal places by the double to integer conversion, this calculation will produce a double xa that is equal to the value of xa less the value of (xa +1) truncated to no decimal places, resulting in xa = a negative value, between -1 and slightly negative of zero. The new xa value after the calculation on line 1267 is equal to the negative of the amount to the right of the decimal point of xa prior to the calculation on line 1267.

Line 1267: xa-=(double (k));

For test purposes, replaced by:

Line (new): xc=xa-(double (k));

16. Initiates **for** loop, starting with j=0, continuing until j=n;

Line 1269: for (j=0, j<n; j++)

a. A **while** loop is initiated within the **for** loop, running while *xc* > 0.0 and k<np. When *xc* is greater than zero, it:
   i. Subtracts 1 from *xc*.
   ii. Increments the value of *k*, increasing the value of *k* by 1.

Note that k is not incremented after it reaches k=np, so as not to exceed the quantity of terrain data stored in pfl.

```
Line 1271:  while (xc>0.0 && k<np)
Line 1273:          xc-=1.0;
Line 1274:          ++k
```

The **for** loop then continues;

b. It sets value of s[j+2) equal to pfl[k+2]+(pfl[k+2]-pfl[k+1])*xc,
c. And then redefines the value of xc, this time equal to xc + xb.

```
Line 1277:  s[j+2]= pfl[k+2]+(pfl[k+2]-pfl[k+1])*xc
Line 1278:  xc=xc+xb
```

An interesting sequence of events happens in this **for** loop. The loop populates the s array elevation values in s array positions s[2] to s[n +1], i.e. s[(n –1) + 2] with values derived from the values in the pfl array. Derived is the key word here; the values in pfl from pfl array locations pfl [k+2], with k incrementing approximately every 2nd s array increment, and proceeding toward location pfl [np –1], are being interpolated to fill a set

of intervals equal to (n –1), the number of intervals in path “n”, with interval widths equal to the intervals in path “n”.

17. Subroutine ***z1sq1*** is then called, with inputs (s , 0.0, sn) where:

s is an array with values:
s[0] = sn (see 11. above)
s[1] = 1.0 (see 12. above)
s[2….(n-1)] = elevations calculated in for loop (see 16. above)

0.0 indicating that ***z1sq1*** is to start at the s array elevation value stored in s[2].
sn which is equal to the length, in increments, of the path considered, also = (n – 1), indicating that ***z1sq1*** is to continue considering the s array elevation values all the way to the value stored in s [n + 1], i.e. s ((n – 1)+2).

Output values of ***z1sq1***:

xa now redefined as the z0, or value of average terrain height line at transmitter site.

xb working variable; after z1sq1 called, is the z1, or value of average terrain height line at receiver site.

18. ***z1sq1*** then returns:

xa = z0, the elevation value of the average terrain line at the transmitter site.
xb = z1, the elevation value of the average terrain line at the receive site.

19. The value of xb is then again redefined to be equal to (xb-xa)/sn, or (the elevation value of the average terrain height at the receive site above the elevation value of the average terrain height at the transmitter site), divided by (n – 1).

Line 1282: xb=(xb-xa)/sn

20. A **for** loop is initiated, starting at j=0; running until j is no longer less than n.

Line 1284: for (j=0; j<n; j++)

a. The loop first subtracts the value of xa from s[j+2], subtracting the average elevation height calculated by ***z1sq1*** from the first elevation height in the s array, leaving as a value, only the amount of deviation, in meters, from the average height, stored in the each of the s array locations.

In Tech Note 101, at 5.16, the term [$-x^2/(2a)$], adding consideration of the effective curvature of the earth, is added to the straight line formula y(x) = h(x), and to the elevation heights, before the plotting of the terrain and calculation of the deviations. However, since here we are not plotting, and the deviations are being obtained by subtracting the straight-line formula results, (the value of xa as incremented by the ***for*** loop), from the terrain heights (found in the s array values prior to processing by this ***for*** loop), the effective curvature term would cancel out. Therefore, the values of the elevation deviations obtained and stored in the s array will be the same with or without consideration of the effective earth radius.

Line 1286: s[j+2]-=xa

b. It then adds the value of (double) xb to (double) xa. As the loop advances, this causes the value of xa to proceed from the z0 value to the z1 value, following along the average elevation height line y= a + bx calculated by ***z1sq1.***

Here again this subroutine branches off from following Tech Note 101, specifically, the description of methodology and procedure specified for determining the terrain roughness factor, σ_h, described in Tech Note 101 at 5.2.2. In its place, the subroutine determines a related value, described as a delta h [Δh]. First, note that this Δh is NOT the same as the Δh correction term defined by Tech Note 101 at (6.12), a term that is to be applied only when the effective heights of the transmitter and/or receiver are greater than a kilometer above sea level. It is related to, but not the same as, the delta h (Δh) function calculated by, and result *d1thxv* provided by, subroutine ***d1thx***.

The subroutine ***d1thx*** is described in Appendix A of NTIA Report TR-82-100, "A Guide to the Use of the ITS Irregular Terrain Model in the Area Prediction Mode", as:

> "Computes the terrain irregularity parameter *dh* from the profile *pfl* between points at x1 [less than] x2.

And in the ITMD, ***d1thx*** is described as:

Using the terrain profile *pfl* we find *Δh*, the interdecile range of elevations between the two points *x1* and *x2*".

The **interdecile range** is a specific interquartile range; it is computed as the difference between the 10th and 90th percentiles. It comprises the middle 80% sample of the population. The term x1 is defined as a distance from the transmitter site to the start point; x2 is defined as a distance from the transmitter site to the end point.

Why is this quantile-based system used to derive *Δh*? Quantiles are less susceptible to long tailed distributions and outliers. Since the elevation data may have anomalies, such as voids in SRTM data, causing outliers far removed from the mean, as long as the

outliers are less than 10% of the total data, causing them to be lost in the lower 10% and/or upper 10% of data that is abandoned in the ***qtile*** subroutine, this methodology provides more accurate results than means and other moment-related statistics.

Further explanation of quantiles can be found in the chapter on the ***qtile*** subroutine, which ***d1thx*** calls below in order to determine the 10th and 90th percentile quantiles.

Line 1287: xa=xa+xb

21. ***qtile*** is a subroutine that returns a quantile. The subroutine ***qtile*** is called twice, with the same path length (n-1), using array s and starting at s array location s[2]; once for quantile (ka-1), intended to be the 90th percentile quantile, and the second time for quantile (kb-1), the 10th percentile quantile. The s+2 term causes the subroutine to skip the s array locations s[0] and s[1] which store the increment length and quantity values, and start with the elevation values in s[2].

22. The value of d1thxv is set to be equal to the difference between the two quantile values obtained from the deviation-from-average-terrain values in the s array, equal to quantile (ka-1)-quantile (kb-1).

Line 1290: d1thxv= qtile(n-1,s+2,ka-1)-qtile(n-1,s+2,kb-1)

With regard to describing the methodology and procedure to this point, for determining this Δh, the interdecile range of elevations between the two points *x1* and *x2*", the NTIA is not specific. George Hufford stated in the Algorithm, Section 1.3:

> "These quantities, together with Δh, are all geometric and should be determined from the terrain profile that lies between the two terminals. We shall not go into detail here."

23. The Algorithm does provide information as to the source of the formula for the next step, where it states in Section 3.2., Preparatory calculations for both modes:

    "We also note here the definitions of two functions of a distance s:

    $$\Delta h(s) = (1 - 0.8\, e^{-s/D})\Delta h \quad \text{with } D = 50 \text{ km.} \qquad \text{Alg. (3.9)}$$

    and

    $$\sigma_h(s) = 0.78\, \Delta h(s) \exp\left[-\left(\Delta h(s)/H\right)^{1/4}\right] \quad \text{with } H = 16 \text{ meters."} \qquad \text{Alg. (3.10)}$$

the second formula, Alg. (3.10), shows a relationship between Δh and the terrain roughness factor σ_h used in Tech Note 101.

The formula Alg. (3.9) can be manipulated for use here; replacing s in Alg. (3.9) with the distance (x2 –x1), the distance between the end point and the start point of the path of elevations considered, specified in meters;

$\Delta h(s) = (1 - 0.8\, e^{-s/D})\Delta h$ with D = 50 km, (50,000 meters) becomes:

$$\frac{\Delta h(s)}{(1 - 0.8\, e^{-s/D})} = \Delta h \quad \text{where } D = 50{,}000 \text{ meters, } s = (x2 - x1)$$

The term $(1 - 0.8\, e^{-s/D})$ ranges in value from .2 for s = 0, up to .706 for s = 50 km.

The value obtained at Line 1290 for *d1thxv*, a.k.a. *Δh(s)*, is then divided by 1.0-.8*exp(– (x2-x1)/50,000), a path distance adjustment factor that appears to be an empirical (data matching) adjustment. For x2-x1= 1,000 m. (1 km) this factor divides the value on Line 1290 by approximately .2; for 49 km, by approximately 0.7, and for 70 km by approximately 0.8, to obtain Δh;

Line 1291: d1thxv/=1.0-.8*exp(-(x2-x1)/50.0e3)

Note: In the FORTRAN version of this program found in Appendix B of NTIA TR-82-100, this line reads:

D1THX=D1THX/(1.0–0.8*EXP(-AMIN1(20.,(X2-X1)/50E3)))

The archaic intrinsic function AMIN1, a minimum determining function similar to min in c++, has been removed in the c++ version of the code.

24. Here, subroutine ***delete []*** is called with regard to array s; to manage the removal of all traces of array s from the computer's working memory that were created by the command **new *[]*** on line 1262.

Line 1292: delete[] s:

25. The output returns the value of d1thxv; Δh, or delta h.

Line 1294: return d1thxv;

SUBROUTINE D1THX2: A functional explanation, by Sid Shumate.
Last Revised July 27, 2008.

Delta h (experimental) subroutine

Note: Used with point-to-point mode. Called by qlrpfl, mid-routine.
Calls mymin, mymax, assert, zlsql, qtile.

Used to find dh, (a.k.a. delta h) the terrain irregularity factor.

Derived from the Irregular Terrain Model description by George Hufford, 2002, (ITMD), compared to the ITM.cpp prepared by J. D. McDonald and John Magliacane for compilation on unix and linux systems. "Line" numbers refer to the ITM.cpp as line numbered by Bloodshed Software's DevC++ print function. "Alg" numbers refer to the algorithm formula in "The ITS Irregular Terrain Model, version 1.22, the ***Algorithm***" (the Algorithm) by G. A. Hufford, 1995. There are also quotes and references, in the background section and subroutine description, from the Appendices to Tech Note 101, Volume II, from NTIA Report TR-82-100, "A Guide to the Use of the ITS Irregular Terrain Model in the Area Prediction Mode" (NTIA TR-82-100), and from "A manual for ITM, "Irregular Terrain Model", released by the NTIA as itm_man.txt, (ITM Manual).

Computes the terrain irregularity parameter *dh* from the profile elevation array *pfl* between points located at x1 and x2. x1 is defined as a distance from the transmitter site to the start point of a path of elevation points considered; x2 is defined as a distance from the transmitter site to the end point. Both x1 and x2 must be specified in meters.

Please note that the ***qlrpfl*** subroutine, and the ***d1thx, hzns*** and ***z1sq1*** subroutines that are called during ***qlrpfl,*** were intended to be experimental early versions of L-R software. They are still in use today, with few modifications or corrections. The ITM Manual states:

> "It should be noted that the original ITM is silent on many of the details for defining some of the path parameters. This is particularly true of the effective heights HE, and, to some lesser degree, of the terrain irregularity parameter DH. The effective height, for example is defined as the height above the "effective reflecting plane," and in the past the investigator has been urged to use his own best judgement as to where that plane should be placed. The subroutine QLRPFL, in trying to automate the definition of all parameters, has been forced to define explicitly all missing details. It has done this in a way that seems reasonable and in full accord with the intent of the model. One should not, however, conclude that these efforts are completed. Hopefully, better results are obtainable."

Background on Delta H, (Δh), the Terrain Irregularity Parameter

Also described as "Δh, the interdecile range of elevations between the two points x1 and x2". The **interdecile range** is a specific interquartile range; it is computed as the

difference between the 10th and 90th percentiles. It comprises the middle 80% sample of the population: in this case, a set of elevation values derived from the elevation data contained in the *pfl* array. The term x1 is defined as a distance from the transmitter site to the start point; x2 is defined as a distance from the transmitter site to the end point. We find by studying the Δh, or delta h, used in the irregular terrain model, that the concept and calculation does not closely follow the definition of the terrain roughness factor, σ_h, specified in Tech Note 101.

In Tech Note 101, on page 5-9, it states:

> "5.2.2. **The terrain roughness factor, σ_h.** The terrain roughness factor σ_h in (5.1) is the root-mean-square deviation of modified terrain elevations, y_i, relative to the smooth curve defined by (5.16), within the limits of the first Fresnel zone in the horizontal reflecting plane. The outline of a first Fresnel zone ellipse is determined by the condition that:
>
> $$r_{11} + r_{21} = r_1 + r_2 + \lambda/2$$
>
> Where:
>
> $r_{11} + r_{21}$ is the length of a ray path corresponding to reflection from a point on the edge of the Fresnel zone,
>
> $r_1 + r_2$ is the length of the reflected ray for which angles of incidence and reflection are equal.

Norton and Omberg [1947] give general formulas for determining a first Fresnel zone ellipse in the reflecting plane. Formulas are given in annex III for calculating distances x_a and x_b from the transmitter to the two points where the first Fresnel ellipse cuts the great circle plane."

Later, on page 5-13, it states: "the terrain roughness factor σ_h, (in Tech Note 101 here designated by a lowercase sigma sub h), is the root-mean-square deviation of modified terrain elevations relative to the curve y(x) within the limits of the first Fresnel zone in the horizontal reflecting plane. The first Fresnel ellipse cuts the great circle plane at two points, x_a and x_b kilometers from the transmitter. The distances x_a and x_b may be computed using equations (III.18) or (III.19) to (III.21) of annex III."

The root-mean-square is the square root of the average of the squares of a set of numbers. If we have a set of values in an array: $x_1, x_2, x_3, \ldots, x_n$, the root-mean-square can be computed, using a for loop, as the square root of the sum of the squared array values divided by n, or:

$$x_{rms} = ((x_1^2 + x_2^2 + x_3^2 \ldots + x_n^2)/n)^{1/2}$$

Deviation refers to the amount of difference between the value being considered and the arithmetic mean value. One of the most important uses of the root-mean-square is to

determine the *standard deviation* from the arithmetic mean. The standard deviation from the mean is the root-mean-square of the deviations from the mean.

In classical statistics, the formula for calculating the variance of an unknown population variance is:

$$\sigma^2 = \frac{\Sigma(x - \mu)^2}{N}$$

Here the population parameter is abbreviated with the Greek letter sigma in lower case, the mean is a population parameter (mu), and the number of samples is represented with a capital N. The term $[x - \mu]$ represents the difference between the sample value (x) and the mean value μ; this term is the deviation of the sample.

The standard deviation, σ, is simply the square root of the variance, or:

$$\sigma = ((\Sigma(x - \mu)^2)/N)^{1/2}$$

Here we are using a sample of the total population. For calculating statistics of a sample of the population, statisticians indicate that the mean is of a sample population by replacing μ with the arithmetic mean $\bar{x}$, and they decrease the denominator by 1, to account for the percent probability that a wider deviation exists in the population than in the sample of the population. The formula then becomes:

$$\sigma = ((\Sigma(x - \bar{x})^2)/(n))^{1/2}$$

Where $n = (N - 1)$

If the arithmetic mean $\bar{x}$ is equal to:

$$\bar{x} = (x_1 + x_2 + x_3 \ldots + x_n)/n,$$

then the standard <u>deviation</u> s can be computed, using a for loop, as:

$$s = (((x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + (x_3 - \bar{x})^2 \ldots + (x_n - \bar{x})^2)/n)^{1/2}.$$

The terrain roughness factor is the root-mean-square deviation of modified terrain elevations relative to the curve y(x) within the limits of the first Fresnel zone in the horizontal reflecting plane. We can now see how to obtain the root-mean-square deviation of terrain height data taken from a selected part of the path between the transmitter and the receiver; the information is in the pfl array. But what is meant by "modified terrain elevations relative to the curve y(x)"?

In 5.2.1, a Curve-Fit to Terrain, found on page 5-8 of Tech Note 101, it states:

> "A smooth curve is fitted to terrain visible from both antennas. It is used to define antenna heights h'_1 and h'_2, as well as to determine a single reflection point where the angle of incidence of a ray r_1 is equal to the angle of a reflection of a

ray r_2 in figure 5.1. This curve is also required to obtain the deviation of terrain heights used in computing R_e in (5.1).

First, a straight line is fitted by least squares to equidistant heights $h_i(x_i)$ above sea level, and $(x_i)^2/(2a)$ is then subtracted to allow for the sea level curvature 1/a illustrated in figure 6.4. The following equation describes a straight-line h(x) fitted to 21 equidistant values of $h_i(x_i)$ for terrain between $x_i = x_0$ and $x_i = x_{20}$ kilometers from the transmitting antenna. The points x_0 and x_{20} are chosen to exclude terrain adjacent to either antenna which is not visible from the other:"

$$h(x) = \bar{h} + m(x - \bar{x}) \qquad (5.15a)$$

Here, $\bar{h}$ is the arithmetic mean of the elevation heights, i.e. the average terrain height:

$$\bar{h} = ((h_1 + h_2 + h_3 \ldots + h_{20})/21 \qquad (5.15b, 1 \text{ of } 3)$$

$\bar{x}$ is the arithmetic mean of the distances from the transmitter site to the to $x_0 + x_{20}$, i.e. the distance from the transmitter site to the center of the x path:

$$\bar{x} = (x_0 + x_{20})/2 \qquad (5.15b, 2 \text{ of } 3)$$

m is the slope of the line. To simplify the later calculation of the least squares solution, the slope is calculated with reference to an x co-ordinate referenced to the center of the x path. If *n* is the number of elevation points, in this example 21, then (*n*-1) is equal to the number of intervals between the elevations points, equal in this example to 21 – 1 = 20. The center of this path of equidistant intervals is then at i = (*n* – 1)/2, or (21 – 1)/2 = 10. So if we want to start a for loop calculation from the transmitter end of the path, we have to start at the recalibrated x path position, the interval closest to the transmitter site, in this example now equal to (i – 10). So the slope calculation will start at i = 0 and proceed to i = 20, represented in the below calculation by i – 10, therefore calculating from –10 to +10.

$$m = \frac{2 * (h_1(i-10) + h_2(i-10) + h_3(i-10) \ldots + h_{20}(i-10))}{77 * (x_{20} - x_0)} \qquad (5.15b, 3 \text{ of } 3)$$

This is a slightly different notation, but is still the same formula given in 5.15b, 3rd of 3. <u>The derivation of the 2/(77*(77 * (x_{20} - x_0))) set of terms is not explained in Tech Note 101; *and provides a correct answer only if the number of increments equals 21.*</u>

The derivation of a more universal version of this formula, which will work with any number of increments, is shown in the description of the subroutine ***z1sq1,*** which this subroutine, ***d1thx2*** calls, in an attempt to perform this least-squares-fit-

to-a-line calculation. Here, we will simply state that the version of the formula for the slope, m, which provides correct results for any number of increments, is:

$$m = \frac{(h_1(i-10) + h_2(i-10) + h_3(i-10) \ldots + h_{20}(i-10))}{((i_0-10)^2 + (i_1 10)^2 + (i_2-10)^2 \ldots + (i_{20}-10)^2)}$$

And is intended to be incorporated in a revised version of ***z1sq1***, to be named ***zlsq2.***

After 5.15b, Tech Note 101 states:

"Smooth modified terrain values given by

$$y(x) = h(x) - x^2/(2a) \qquad (5.16)$$

will then define a curve of radius a which is extrapolated to include all values of x from x = 0 to x =d, the positions of the antennas."

Here, a represents the effective radius of the earth under the great circle signal path. The $- x^2/(2a)$ term accounts for the effective curvature of the earth; it adds the effective increase in terrain height due to the effective curvature of the earth, to the "flat earth" signal path average terrain line formula. Section 4 of Tech Note 101 describes a method of deriving the effective earth's radius, a, from n_s, the atmospheric refractive index at the surface of the earth, and a_o, the actual radius of the earth.

Note before reading further: The Δh correction term defined by (6.12) in the next quote, mentioned after (5.17), is not the same Δh we have previously discussed in this section. It is only distantly related to, and not the same as, the delta h (Δh) function calculated by, and result *d1thx2v* provided by, subroutine ***d1thx2***.

Continuing with Tech Note 101 after (5.16). it states:

"The heights of the antennas above this curve are:

$$h'_1 = h_{ts} - h(0), \qquad h'_2 = h_{rs} - h(d) \qquad (5.17)$$

If h'_1 or h'_2 is greater than one kilometer, a correction term, Δh, defined by (6.12) and shown on figure 6.7, is used to reduce the value given by (5.17).

Where terrain is so irregular that it cannot be reasonably well approximated by a single curve, σ_h is large and $R_e = 0$, not because the terrain is very rough, but because it is irregular. In such a situation, method 3 of section 5.1 may be useful."

The Federal Communications Commission's Terrain Roughness Factor

The following is excerpted from the Federal Communications Commission (FCC) Rules and Regulations, as published as Code of Federal Regulations Title 47, Volume 4, Subpart B, FM Broadcast Stations, Sec. 73.313, Prediction of coverage: [CITE: 47CFR73.313, and for the diagrams, 47CFR73.333]:

(f) The effect of terrain roughness on the predicted field strength of a signal at points distant from an FM transmitting antenna is assumed to depend on the magnitude of a terrain roughness factor (h) which, for a specific propagation path, is determined by the characteristics of a segment of the terrain profile for that path 40 kilometers in length located between 10 and 50 kilometers from the antenna. The terrain roughness factor has a value equal to the distance, in meters, between elevations exceeded by all points on the profile for 10% and 90% respectively, of the length of the profile segment. (See Sec. 73.333, Figure 4.)

(g) If the lowest field strength value of interest is initially predicted to occur over a particular propagation path at a distance that is less than 50 kilometers from the antenna, the terrain profile segment used in the determination of terrain roughness factor over that path must be that included between points 10 kilometers from the transmitter and such lesser distances. No terrain roughness correction need be applied when all field strength values of interest are predicted to occur 10 kilometers or less from the transmitting antenna.

(h) Profile segments prepared for terrain roughness factor determinations are to be plotted in rectangular coordinates, with no less than 50 points evenly spaced within the segment using data obtained from topographic maps with contour intervals of approximately 15 meters (50 feet) or less if available.

(i) The field strength charts (Sec. 73.333, Figs. 1-1a) were developed assuming a terrain roughness factor of 50 meters, which is considered to be representative of average terrain in the United States. Where the roughness factor for a particular propagation path is found to depart appreciably from this value, a terrain roughness correction ([Delta]F) should be applied to field strength values along this path, as predicted with the use of these charts. The magnitude and sign of this correction, for any value of [Delta]h, may be determined from a chart included in Sec. 73.333 as Figure 5.

(j) Alternatively, the terrain roughness correction may be computed using the following formula:

[Delta]F=1.9-0.03([Delta]h)(1+f/300)

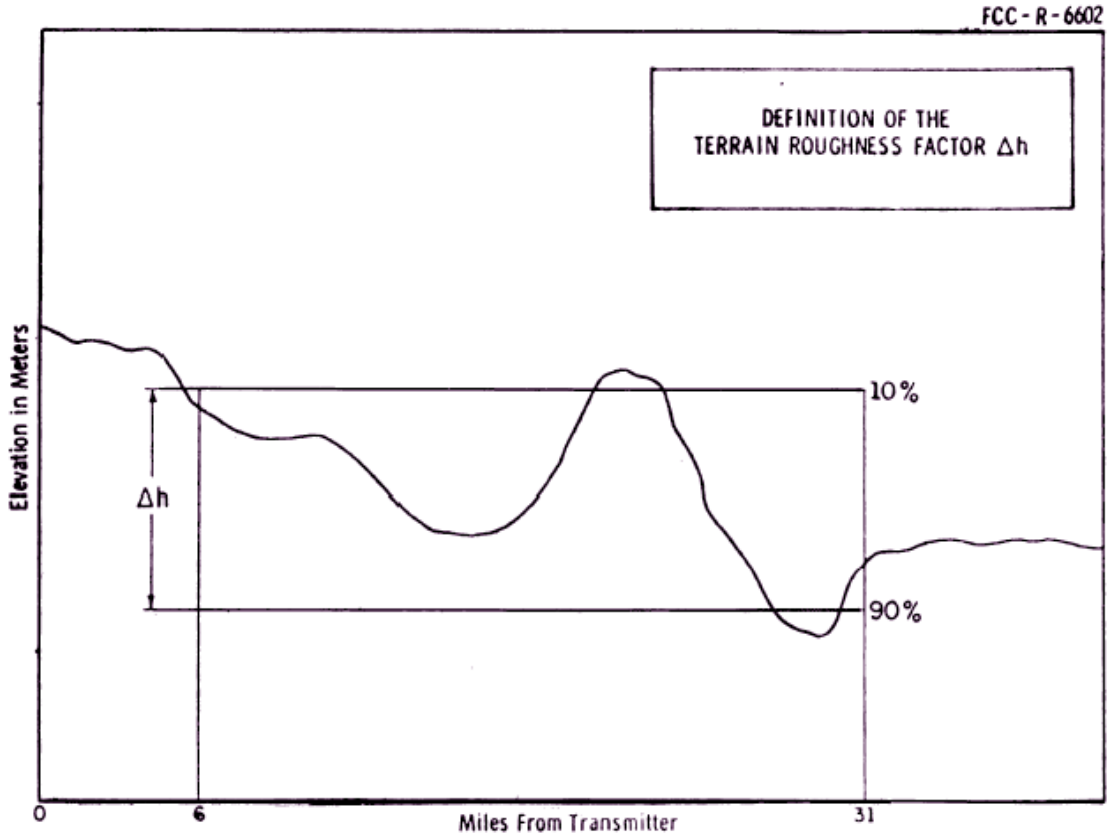
Where:
[Delta]F=terrain roughness correction in dB

[Delta]h=terrain roughness factor in meters
f=frequency of signal in MHz (MHz)

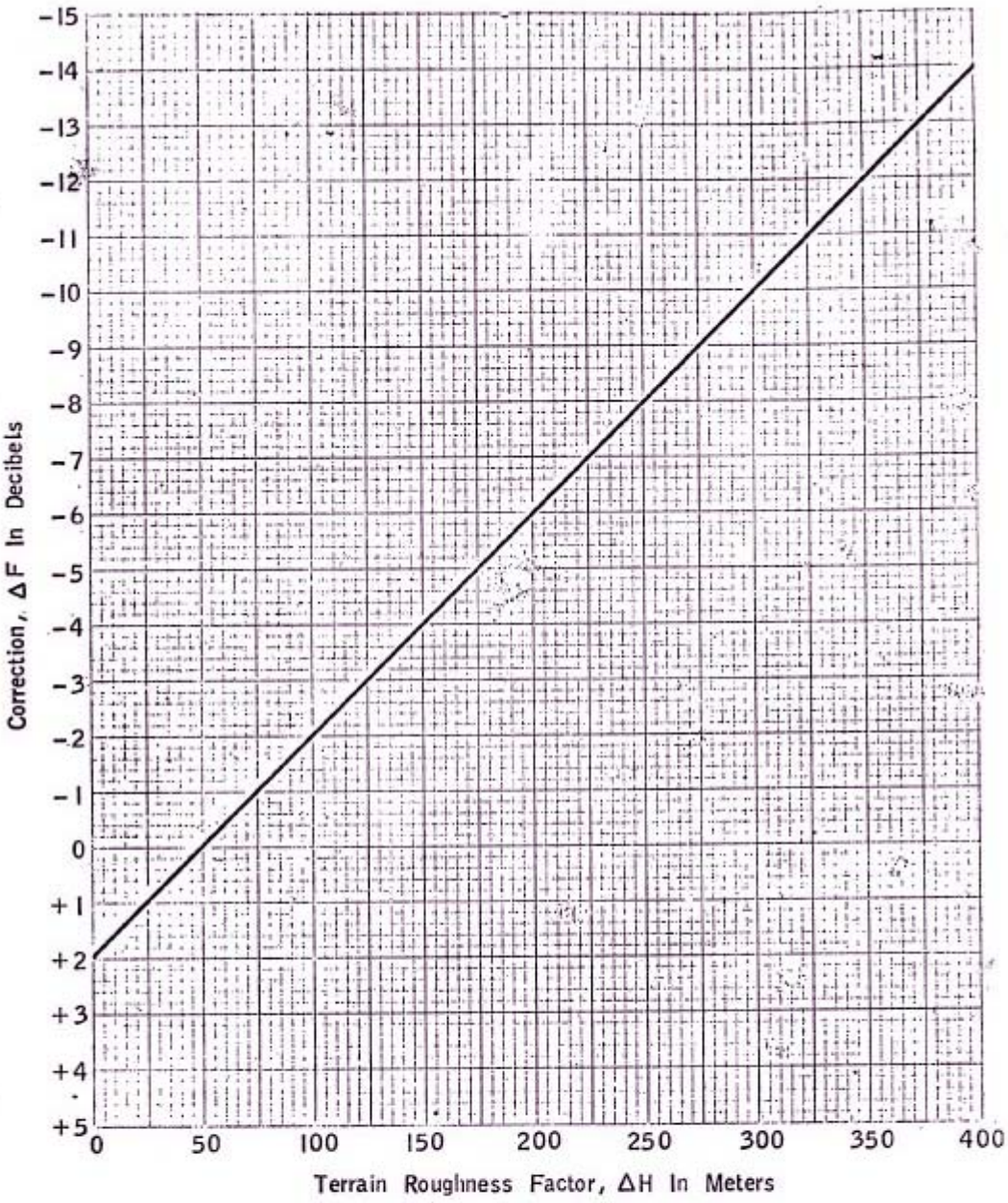
Effective Date Note: At 42 FR 25736, May 19, 1977, the effective date of Sec. 73.313 paragraphs (i) and (j) was stayed indefinitely.

NOTE: The FCC's terrain roughness factor is a related concept, but not the same thing, as the Tech Note 101 terrain roughness factor. In Report No. R-6602, "Development of VHF and UHF Propagation Curves for TV and FM Broadcasting," by Damelin, Daniel, Fine and Waldo, Sept.1966, it states:

"The corrections for terrain roughness are intended for application in estimating median (or average) field strengths over areas where the general character of the terrain is fairly uniform, or where there is no abrupt change in terrain roughness. It is not possible to accurately predict the field strength at any given receiver site."



FCC § 73.333 FIGURE 4
(new)



TERRAIN ROUGHNESS CORRECTION

for use with estimated FM F(50,50) and F(50,10) field strength curves

FCC §73.333 FIGURE 5

(Secs. 4, 5, 303, 48 Stat., as amended, 1066, 1068, 1082 (47 U.S.C. 154, 155, 303))

[28 FR 13623, Dec. 14, 1963, as amended at 35 FR 2591, Feb. 5, 1970; 40 FR 27679, July 1, 1975; 45 FR 28141, Apr. 28, 1980; 48 FR 29508, June 27, 1983; 49 FR 19670, May 9, 1984]

EFFECTIVE DATE NOTE: At 42 FR 25736, May 19, 1977, in §73.333, the effective date of Figures 4 and 5 was stayed indefinitely.

Now, on to the description of the subroutine. From ITMD Section 44:

Call inputs:

pfl	terrain elevation profile array, starting at transmitter site and ending at receiver site, following great circle path, with: pfl[0] =*enp*, the number of increments pfl[1] = *xi*, distance per increment (in meters) pfl[2] = *z(0)*, the transmitter tower base AMSL, or elevation height pfl[[np+2] = *z(np)*, the receive location AMSL, the last elevation.
&x1	x1, input from qlrpfl as xl[0], the start point of the section of the total path to be considered, defined as a distance (measured in meters) from the transmitter site. The value is equal to the lesser of: 15 times the transmitter height AGL, or 1/10 of the distance from the transmitter site to the horizon.
&x2	x2, output from qlrpfl as xl[1], the end point of the section of the total path to be considered, defined as a distance (in meters) from the transmitter site. The value is equal to the lesser of: 15 times the receiver antenna height AGL, or 1/10 of the distance from the receive site to the horizon.

This subroutine declares the following private, or local, arguments:

Here the argument types are noted (int, and double, in this case) because this subroutine incorporates conversions between (int) and (double).

Declared as type integers (int):

np	number of points in pfl elevation array.
ka	equal to 1/10 of (8+length of the section of the total path to be considered), range limited by kmx.
kb	equal to n- (ka+1).
n	number of intervals between the transmitter site and the end point of a path to be considered in this subroutine only.
k	initially set to be one more than xa.
kmx	maximum range limit for ka.
j	counting variable in a ***for*** loop.

Declared as type double decimal precision (double):

d1thx2v	calculated terrain irregularity parameter; output from d1thx2.
sn	one less than n. The total path length, measured in increments, of path n.
xa	distance from transmitter to start of test path for s array.
xb	distance from transmitter to end of test path for s array.
xc	working variable for increment distance of s loop
*s	an array of elevations interpolated from terrain elevations in the pfl array, the renamed elev array relayed by the point_to_point (or point_to_point_two) subroutine from the wrap-around input-output software.

This subroutine:

1. Defines *np*, number of points, to be equal to *pfl [0]*, the number of increments in array pfl.

Line 1249: np=(int)pfl [0];

2. Defines *xa* to be equal to xl/*pfl [1], equal to x1(x-one, the distance from the transmitter site to the start point being considered), divided by the increment distance,* initially making double *xa* equal to the number of increments from the transmitter site to the start point of the section of the total path to be considered.

Line 1250: xa=x1/pfl[1];

3. Defines *xb* to be equal to x2/*pfl [1], equal to x2(receiver site) divided by the increment distance,* initially making double *xb* equal to the number of increments from the transmitter site to the end point of the section of the total path to be considered.

Line 1251: xb=x2/pfl[1];

4. Presets d1thx2v=0.0

Line 1252: d1thx2v=0.0.

5. An ***if*** statement used to check if (xb - xa<2.0). If the number of increments in the considered path is less than 2, indicating a path length too short to calculate d1thx2v, it causes the program to exit, returning d1thx2v = 0.0 .

```
Line 1254:      if  (xb-xa<2.0)
Line 1255:              return d1thx2v;
```

6. The variable *ka* is set to be equal to 1/10 of (xb-xa+8), or 1/10 of (8+length of the section of the total path to be considered, measured in increments), then sets the range of *ka* to be between 4 and 25 increments. This will be the start point of the delta-h analysis path.

Since *ka* is declared as an int, the data used to calculate its value must be int; but the declared local xb and xa are of type double. The function, by stating (int) after the equal sign and before the calculation, forces (creates) an output that is (int).

```
Line 1257:      ka=(int)(0.1*(xb-xa+8.0));
In ITM:         ka=mymin(mymax(4,ka),25);
```

7. Sets value of *n* equal to (10 times ka) – 5; since ka is limited in range to between 4 and 25, then n is limited in range to between 35 and 245 increments.

NOTE: In ITM, the limitation of ka to be a maximum of 25 increments CREATES AN ARCHAIC, PROBLEM-CAUSING RANGE LIMIT FOR n, FAR TOO SMALL FOR TODAY'S 90 METER (3 ARC-SECOND), 30 METER (1 ARC-SECOND) AND 10 METER (1/3 ARC-SECOND) PATHS, WITH THOUSANDS OF INTERVALS. It limits n to a maximum of 245 times 111 m, or 27 km, for 3 arc-second database interval sizes, which is marginally acceptable; but it limits the maximum consideration to only 9.065 km for 1 arc-second database all-points intervals, and 3.02 km for 1/3 arc second database all-points intervals. In the ITWOM version, this subroutine has been modified to set ka based on path length of up to 20 kilometers, as per TN101 section 5.2.1. TN101 uses 20 km as a suggested range to include the majority of the middle 80% of terrain for line-of-sight paths. This is a gross simplification that assumes that this middle 80% of terrain can be approximated by a straight line, an assumption that fails for highly irregular terrain.

The FCC regulations for their version of [delta] h specifies a minimum 10 km, and up to a 50 km maximum, limit; the n limit of 7.35 km for 1 arc-second database all-points intervals, and 2.45 km for 1/3 arc second database intervals, are both below the minimum FCC limit for consideration.

In continuing the comparison to the FCC's terrain consideration rules, we find that FCC 47CFR73.7313 indicates that the consideration of terrain roughness extends from n = 10 to 50 km; i.e. the maximum length of the path to be considered was 50 km, the calculation was to be made using terrain information on the signal path starting at 10 km and extending to the receive point, up to a maximum of 50 km. No consideration was to be made for terrain roughness if the path was less than 10 km. The length of n, being 10 times ka less 5, can only compare well with the 10 to 50 meter maximum range of the FCC's delta h determination method, unless ka = 1, where n would = 5; since ka is range-

limited to be not less than 4, this is not possible. So let us look at another part of the FCC regulations; where a minimum number of 50 increments over the 40 km consideration path section are required. Converting from paper maps to a digital database, this is 1.2 increments per km, equivalent to an 830-meter database, or roughly equivalent to, and therefore compatible with, the 900 meter GLOBE 30 arc-second database utilized by the Windows version of the ITM (itm.exe), made available by NTIA.

The maximum and minimum limits for ka, therefore, need to be set based upon the path lengths, not the number of intervals. We only need to reset the maximum limit, as ka is calculated to be far less than xb. It would be in accordance with the FCC guidelines to limit the maximum length of n to 50 km, or 50,005 meters. Distance in meters can be converted to intervals by dividing by pfl[1], the increment width. But for now we will remain with the ITM convention of 20 km.

On Line 1245; in dithx2, we have added kmx, between k, and j on the int declaration line.

The units of pfl[1] is meters/increment, so the ***d1thx*** line:

```
ka=mymin(mymax(4,ka),25);
```

should be replaced by:

```
kmx=mymax(25,(int)(20005/(pfl[1])));
ka=mymin(mymax(4,ka),kmx);
```

Line 1258r sets the kmx, or <u>k</u>a <u>ma</u>ximum, to be the maximum of 25 or the integer value of x2, the path length to the transmitter end of the short path considered, divided by 10 times the interval width, leaving kmx measured in increments. For example, for a 3-arc-second database with maximum 90 m. intervals, the interval width is approximately 79 meters; for a 20 km (12.4 mi.) total path, kmx would be limited to approximately 25 intervals, and for a 1-arc second database would be limited to approximately 76 intervals. The limit of 25 intervals is replaced with kmx, which divides the limiting distance of 25000 meters by the width of the interval to provide a maximum number of intervals.

END FIRST CORRECTION NOTE. Returning to the discussion of ***d1thx2***;

The argument n is set to be equal to 10 times ka, less 5 intervals, in intervals.

Line 1259: n=10*ka-5

8. Sets value of *kb* equal to *n*-*ka*+1, or equal to 10*ka-5-ka+1 = 9*ka-4.

Line 1260: kb= n–ka+1

9. Sets value of *sn* equal to *n*-1, the total path length, measured in increments, of path n.

Line 1261: sn=n-1,

10. Calls subroutine ***assert*** with input parameters (s=new double[n+2]) !=0). ***assert*** is a standard c function prototype that, in c++, returns an error message and aborts the program if the expression, in this case s=new double[n+2], is equal to zero, i.e. the expression is false. If *n* +2 = 0, the program aborts, returning an error message. Note the use of new to create and allocate memory for the s array, and of double to declare the s array values to be doubles, where n was declared as an integer (int).

Line 1262: assert (s=new double[n+2]) !=0)

11. Sets value of array value s[0] equal to sn, which is equal to the length, in increments, of the path considered, also = (n – 1).

Line 1263: s[0]= sn

12. Sets value of array value s[1] equal to 1.0

Line 1264: s[1]=1.0

13. Sets value of *xb* to be equal to *(*xb-*xa)/sn*, where:

(xb, prior to step 13) = the number of increments from the transmitter site to x2, the end point of the path considered.
xa = the number of increments from the transmitter to the start point, x1, of the path considered.
sn is the length of the path considered, measured in increments.

So (xb-*xa*) is the length, measured in increments, of the path between x1 and x2. *xb* is redefined to be the length of the path between x1 and x2, divided by the length of sn, the length of the "n" path defined by 10*ka – 6, measured in intervals. This makes *xb* equal to the ratio of: the length of the x1 to x2 path, to the length of the "n" path derived from ka.

Line 1265: xb=(xb-xa)/sn

Warning! This causes computational error of if ka is restricted so that the value of sn causes xb to exceed 1.0.

14. Sets value of *k* equal to xa+1.0, equal to the number of increments from the transmitter to the start point, plus one. The use of (int) after the equal sign allows and forces a calculation using (double) xa, to produce an integer result. The value of xa is truncated, not rounded off, to zero decimal places, so 3.9 and 3.3 both would calculate as 3, i.e. if xa is 3.9, *k* = 3 +1 = 4, if xa is 3.3, *k* = 3 +1 = 4.

Line 1266: k=(int (xa+1.0));

NOTE CHANGE BELOW FOR TEST PURPOSES; REPLACES XA BY XC FOR FOR/WHILE LOOP USE.

15. Here, the original version of the ITM resets the value of double xa by subtracting value of k. However, the value of *xa* is needed later; so the argument *ac* has been declared and substituted here for the ITWOM. The use of the (double) before k allows and forces a type double result to a calculation incorporating the integer (int) k. Since k = (xa +1), with the value truncated to zero decimal places by the double to integer conversion, this calculation will produce a double xa that is equal to the value of xa less the value of (xa +1) truncated to no decimal places, resulting in xa = a negative value, between -1 and slightly negative of zero. The new xa value after the calculation on line 1267 is equal to the negative of the amount to the right of the decimal point of xa prior to the calculation on line 1267.

Line 1267: xa-=(double (k));

For test purposes, replaced by:

Line (new): xc=xa-(double (k));

16. Initiates **for** loop, starting with j=0, continuing until j=n;

Line 1269: for (j=0, j<n; j++)

a. A **while** loop is initiated within the **for** loop, running while *xc* > 0.0 and k<np. When *xc* is greater than zero, it:
   i. Subtracts 1 from *xc*.
   ii. Increments the value of *k*, increasing the value of *k* by 1.

Note that k is not incremented after it reaches k=np, so as not to exceed the quantity of terrain data stored in pfl.

```
Line 1271:    while (xc>0.0 && k<np)
Line 1273:            xc-=1.0;
Line 1274:            ++k
```

The **for** loop then continues;

b. It sets value of s[j+2) equal to pfl[k+2]+(pfl[k+2]-pfl[k+1])*xc,
c. And then redefines the value of xc, this time equal to xc + xb.

Line 1277: s[j+2]= pfl[k+2]+(pfl[k+2]-pfl[k+1])*xc

Line 1278: xc=xc+xb

An interesting sequence of events happens in this **for** loop. The loop populates the s array elevation values in s array positions s[2] to s[n +1], i.e. s[(n –1) + 2] with values derived from the values in the pfl array. Derived is the key word here; the values in pfl from pfl array locations pfl [k+2], with k incrementing approximately every 2nd s array increment, and proceeding toward location pfl [np –1], are being interpolated to fill a set of intervals equal to (n –1), the number of intervals in path "n", with interval widths equal to the intervals in path "n".

17. Subroutine ***z1sq1*** is then called, with inputs (s , 0.0, sn) where:

s is an array with values:
s[0] = sn (see 11. above)
s[1] = 1.0 (see 12. above)
s[2….(n-1)] = elevations calculated in for loop (see 16. above)

0.0 indicating that ***z1sq1*** is to start at the s array elevation value stored in s[2].
sn which is equal to the length, in increments, of the path considered, also = (n – 1), indicating that ***z1sq1*** is to continue considering the s array elevation values all the way to the value stored in s [n + 1], i.e. s ((n – 1)+2).

Output values of ***z1sq1***:

xa now redefined as the z0, or value of average terrain height line at transmitter site.

xb working variable; after z1sq1 called, is the z1, or value of average terrain height line at receiver site.

18. ***z1sq1*** then returns:

xa = z0, the elevation value of the average terrain line at the transmitter site.
xb = z1, the elevation value of the average terrain line at the receive site.

19. The value of xb is then again redefined to be equal to (xb-xa)/sn, or (the elevation value of the average terrain height at the receive site above the elevation value of the average terrain height at the transmitter site), divided by (n – 1).

Line 1282: xb=(xb-xa)/sn

20. A **for** loop is initiated, starting at j=0; running until j is no longer less than n.

Line 1284: for (j=0; j<n; j++)

a. The loop first subtracts the value of xa from s[j+2], subtracting the average elevation height calculated by ***z1sq1*** from the first elevation height in the s array, leaving as a value, only the amount of deviation, in meters, from the average height, stored in the each of the s array locations.

In Tech Note 101, at 5.16, the term [$-x^2/(2a)$], adding consideration of the effective curvature of the earth, is added to the straight line formula y(x) = h(x), and to the elevation heights, before the plotting of the terrain and calculation of the deviations. However, since here we are not plotting, and the deviations are being obtained by subtracting the straight-line formula results, (the value of xa as incremented by the ***for*** loop), from the terrain heights (found in the s array values prior to processing by this ***for*** loop), the effective curvature term would cancel out. Therefore, the values of the elevation deviations obtained and stored in the s array will be the same with or without consideration of the effective earth radius.

Line 1286: s[j+2]-=xa

b. It then adds the value of (double) xb to (double) xa. As the loop advances, this causes the value of xa to proceed from the z0 value to the z1 value, following along the average elevation height line y= a + bx calculated by ***z1sq1.***

Here again this subroutine branches off from following Tech Note 101, specifically, the description of methodology and procedure specified for determining the terrain roughness factor, σ_h, described in Tech Note 101 at 5.2.2. In its place, the subroutine determines a related value, described as a delta h [Δh]. First, note that this Δh is NOT the same as the Δh correction term defined by Tech Note 101 at (6.12), a term that is to be applied only when the effective heights of the transmitter and/or receiver are greater than a kilometer above sea level. It is related to, but not the same as, the delta h (Δh) function calculated by, and result *d1thx2v* provided by, subroutine ***d1thx2***.

The subroutine ***d1thx*** is described in Appendix A of NTIA Report TR-82-100, "A Guide to the Use of the ITS Irregular Terrain Model in the Area Prediction Mode", as:

> "Computes the terrain irregularity parameter *dh* from the profile *pfl* between points at x1 [less than] x2.

And in the ITMD, ***d1thx*** is described as:

Using the terrain profile *pfl* we find *Δh*, the interdecile range of elevations between the two points *x1* and *x2*".

The **interdecile range** is a specific interquartile range; it is computed as the difference between the 10th and 90th percentiles. It comprises the middle 80% sample of the

population. The term x1 is defined as a distance from the transmitter site to the start point; x2 is defined as a distance from the transmitter site to the end point.

Why is this quantile-based system used to derive Δh? Quantiles are less susceptible to long tailed distributions and outliers. Since the elevation data may have anomalies, such as voids in SRTM data, causing outliers far removed from the mean, as long as the outliers are less than 10% of the total data, causing them to be lost in the lower 10% and/or upper 10% of data that is abandoned in the ***qtile*** subroutine, this methodology provides more accurate results than means and other moment-related statistics.

Further explanation of quantiles can be found in the chapter on the ***qtile*** subroutine, which ***d1thx2*** calls below in order to determine the 10th and 90th percentile quantiles.

Line 1287: xa=xa+xb

21. ***qtile*** is a subroutine that returns a quantile. The subroutine ***qtile*** is called twice, with the same path length (n-1), using array s and starting at s array location s[2]; once for quantile (ka-1), intended to be the 90th percentile quantile, and the second time for quantile (kb-1), the 10th percentile quantile. The s+2 term causes the subroutine to skip the s array locations s[0] and s[1] which store the increment length and quantity values, and start with the elevation values in s[2].

22. The value of d1thx2v is set to be equal to the difference between the two quantile values obtained from the deviation-from-average-terrain values in the s array, equal to quantile (ka-1)-quantile (kb-1).

Line 1290: d1thx2v= qtile(n-1,s+2,ka-1)-qtile(n-1,s+2,kb-1)

With regard to describing the methodology and procedure to this point, for determining this Δh, the interdecile range of elevations between the two points *x1* and *x2*", the NTIA is not specific. George Hufford stated in the Algorithm, Section 1.3:

> "These quantities, together with Δh, are all geometric and should be determined from the terrain profile that lies between the two terminals. We shall not go into detail here."

23. The Algorithm does provide information as to the source of the formula for the next step, where it states in Section 3.2., Preparatory calculations for both modes:

    "We also note here the definitions of two functions of a distance s:

    $$\Delta h(s) = (1 - 0.8\, e^{-s/D})\Delta h \quad \text{with } D = 50 \text{ km.} \qquad \text{Alg. (3.9)}$$

    and

    $$\sigma_h(s) = 0.78\, \Delta h(s) \exp\left[-\left(\Delta h(s)/H\right)^{1/4}\right] \quad \text{with } H = 16 \text{ meters."} \qquad \text{Alg. (3.10)}$$

the second formula, Alg. (3.10), shows a relationship between Δh and the terrain roughness factor σ_h used in Tech Note 101.

The formula Alg. (3.9) can be manipulated for use here; replacing s in Alg. (3.9) with the distance (x2 –x1), the distance between the end point and the start point of the path of elevations considered, specified in meters;

$$\Delta h(s) = (1 - 0.8\, e^{-s/D})\Delta h \quad \text{with } D = 50 \text{ km, (50,000 meters) becomes:}$$

$$\frac{\Delta h(s)}{(1 - 0.8\, e^{-s/D})} = \Delta h \quad \text{where } D = 50{,}000 \text{ meters, } s = (x2 - x1)$$

The term $(1 - 0.8\, e^{-s/D})$ ranges in value from .2 for s = 0, up to .706 for s = 50 km.

The value obtained at Line 1290 for *d1thx2v*, a.k.a. *Δh(s)*, is then divided by 1.0-.8*exp(– (x2-x1)/50,000), a path distance adjustment factor that appears to be an empirical (data matching) adjustment. For x2-x1= 1,000 m. (1 km) this factor divides the value on Line 1290 by approximately .2; for 49 km, by approximately 0.7, and for 70 km by approximately 0.8, to obtain Δh;

Line 1291: d1thx2v/=1.0-.8*exp(-(x2-x1)/50.0e3)

Note: In the FORTRAN version of this program found in Appendix B of NTIA TR-82-100, this line reads:

D1THX=D1THX/(1.0–0.8*EXP(-AMIN1(20.,(X2-X1)/50E3)))

The archaic intrinsic function AMIN1, a minimum determining function similar to min in c++, has been removed in the c++ version of the code.

24. Here, subroutine ***delete []*** is called with regard to array s; to manage the removal of all traces of array s from the computer's working memory that were created by the command **new *[]*** on line 1262.

Line 1292: delete[] s:

25. The output returns the value of d1thx2v; Δh, or delta h.

Line 1294: return d1thx2v;

SUBROUTINE FHT: A functional explanation, by Sid Shumate.
Last Revised July 1, 2007.

Function Height-Gain for Three-Radii method; subroutine: ***fht***.

Note: Used with both point-to-point mode and area mode. Called by ***adiff.***

Descriptions derived from the Irregular Terrain Model description by George Hufford, 2002, (ITMD), compared to the ITM.cpp prepared by J. D. McDonald and John Magliacane for compilation on unix and linux systems. "Line" numbers refer to the ITM.cpp as line numbered by Bloodshed Software's DevC++ print function. "Alg" numbers refer to the algorithm formula in "The ITS Irregular Terrain Model, version 1.22, the Algorithm" by G. A. Hufford, 1995.

From ITMD Section 14:

Calculates the height-gain over a smooth spherical earth – to be used in the "three radii" method. The approximation is that given in [Alg 6.4].

Note that in the Algorithm, in the first paragraph in Section 6, "Addenda – numerical approximations", from which the formulas below are taken, George Hufford states:

> "Part of the algorithm for the ITM consists in approximations for the standard functions that have been used. In these approximations, computational simplicity has often taken greater priority than accuracy."

Call inputs:
& *x*
& *pk*

Declares private, or local, arguments:
w
fhtv height gain over a smooth spherical earth

This subroutine:

1. Initiates an **if** statement. If *x* is less than 200, then *w* is set to be = (–log(*pk*)).

Line 133:
```
if (x<200.0)
        if (x<200.0)
        w=-log(pk);
```

2. An ***if*** statement is nested within the first ***if*** statement. If *x* is less than 200, and; *pk* is less than 1.0e–5, or if ($x*w^3$) is greater than 5,495, then:
   a. *fhtv* is set to be equal to –117.0;
   b. An ***if*** statement is nested within the second ***if*** statement. Therefore:

(1.) if *x* is less than 200, and:
(2.) *pk* is less than 1.0e–5, or ($x*w^3$) is greater than 5,495, and:
(3.) *x* is greater than 1.0, then:

c. *fhtv* is reset to be equal to 17.372 *log (*x)* + value of *fhtv* from 2(a).

```
Line 138:     if (pk<1.0e-5 || x*w*w*w > 5495.0)
              {
                      fhtv=-117.0;

                      if (x>1.0)
                              fhtv=17.372*log(x)+fhtv              [Alg. 6.5]
              }
```

3. The second **if** statement, found on line 138, has an offsetting **else** statement. So if:
   a. *x* is less than 200, and:
   b. *pk* is not less than 1.0e–5, or ($x*w^3$) is not greater than 5,495, then:
   fhtv is reset to be = (2.5e –5)*(x^2/pk) –(8.686*w) –15.0 [Alg. 6.6]

```
Line 145:     else
                      fhtv=2.5e-5*x*x/pk–8.686*w–15.0;
      }
```

4. The first **if** statement, found on line 133, has an offsetting **else** statement. So if *x* is greater than or equal to 200, then:

   fhtv is reset to be = 0.05751*x–4.343*log(x) [Alg. 6.3]

An ***if*** statement is nested within this **else** statement. So if the value of *x* is greater than or equal to 200, and less than 2000, then:

a. *w* is set to be equal to 0.0134**x**exp(–0.005**x*)
b. *fhtv* is reset to be = (1.0-*w*)* *fhtv*+*w**(17.372*log(x) –117.0)
   [Alg. 6.4]

```
Line 149:     else
              {
                      fhtv=0.05751*x-4.343*log(x);
                      if (x<2000.0)
                      {
                              w=0.0134*x*exp(-0.005*x);
                              fhtv=(1.0-w)*fhtv+w*(17.372*log(x)-117.0);
                      }
              }
```

5. Subroutine ***fht*** then returns *fhtv*, the height-gain over a smooth spherical earth.

SUBROUTINE H0F: A functional explanation, by Sid Shumate.
Last Revised July 15, 2007.

H0 Frequency gain function for scatter fields; subroutine: ***h0f***.

Note: Used with both point-to-point mode and area mode. Called by ***ascat.***

Descriptions derived from the Irregular Terrain Model description by George Hufford, 2002, (ITMD), compared to the ITM.cpp prepared by J. D. McDonald and John Magliacane for compilation on unix and linux systems. "Line" numbers refer to the ITM.cpp as line numbered by Bloodshed Software's DevC++ print function. "Alg" numbers refer to the algorithm formula in "The ITS Irregular Terrain Model, version 1.22, the Algorithm" by G. A. Hufford, 1995.

From ITMD Section 25:

This is the H_{01} function for scatter fields as defined in the Algorithm, Section 6, "Addenda – numerical approximations."

Background:

From the Algorithm, Section 6, "Addenda – numerical approximations."

This section starts by mentioning:

> "Part of the algorithm for the ITM consists in approximations for the standard functions that have been used. In these approximations, computational simplicity has often taken greater priority than accuracy."

The Algorithm later states, following equation (6.9):

> "The frequency gain function may be written as
>
> $$H_0 = H_{00}(r_1, r_2, \eta_s,) + \Delta H_0 \qquad (6.10)$$
>
> where
>
> $$\Delta H_0 = 6\,(0.6 - \log n_s) * \log s_s * \log r_2/s_s\, r_1 \qquad (6.11)$$
>
> and where H_{00} is obtained by linear interpolation between its values when n_s is an integer. For $\eta_s = 1, \ldots, 5$ we set
>
> $$H_{00}(r_1, r_2, j) = \tfrac{1}{2}[\, H_{01}(r_1, j) + H_{01}(r_2, j) \qquad (6.12)$$
>
> with
>
> $$H_{01}(r, j) = \begin{cases} 10 \log (1 + 24r^{-2} + 25\, r^{-4}) & \text{for } j = 1, \\ 10 \log (1 + 45r^{-2} + 80\, r^{-4}) & \text{for } j = 2, \end{cases} \qquad (6.13)$$

$$10 \log (1 + 68r^{-2} + 177\, r^{-4}) \quad \text{for } j = 3,$$
$$10 \log (1 + 80r^{-2} + 395\, r^{-4}) \quad \text{for } j = 4,$$
$$10 \log (1 + 105r^{-2} + 705\, r^{-4}) \quad \text{for } j = 5.$$

For $\eta_s > 5$, we use the value for $\eta_s = 5$, and for $\eta_s = 0$ we suppose

$$H_{00}(r_1, r_2, 0) = 10 \log [(1 + (2)^{1/2}/ r_1)^2\ (1 + (2)^{1/2}/ r_2)^2 * (r_1 + r_2)/(r_1 + r_2 + 2(2)^{1/2})] \tag{6.14}$$

In all of this, we truncate the values of s_s and $q = r_2 / s_s\, r_1$ at 0.1 and 10."

Call inputs for subroutine ***h0f***:

r twice the angular distance th, (measured in a ratio of meters, vertical to meters, horizontal, not radians as in TN101 Section 9.2) times the effective height of the terminal antenna (r1 = transmit, r2 = receive) in meters, divided by a wavelength at the frequency selected, in meters. Units (for itm.cpp) cancel out to be dimensionless, not radians as in TN101 Section 9.2.

et the value of η_s, the "scatter efficiency factor"

Declares private, or local, arguments:

double *a[5]*={25.0, 80.0, 177.0, 395.0, 705.0};
double *b[5]*={24.0, 45.0, 68.0, 80.0, 105.0};
double *q*,
double *x*;
double *h0fv*,
double *temp*;
int *it*;

This subroutine:

1. Presets *it* to be equal to: the integer value of input value *et*, (which is equal to the value of η_s, the scatter efficiency factor). This follows from the statement in the Algorithm, Section 6, following equation (6.11), which states: "and where H_{00} is obtained by linear interpolation between its values when η_s is an integer.

Line 170: it=(int)et;

2. Initiates an **if** statement. If *it* is less than or equal to zero, then:
   a. the value of *it* is reset to be equal to 1.
   b. *q* is set to be equal to 0.0.

Line 172:
```
if (it<=0)
{
it=1;
```

```
q=0.0;
}
```

3. An ***else if*** statement follows; if *it*, at line 172, was equal to or greater than 5, then:
   a. the value of *it* is reset to be equal to 5.
   b. *q* is set to be equal to 0.0.

Line 178:
```
else if (it>=5)
{
        it=5;
        q=0.0;
}
```

Steps 2 and 3 prepare for the use of the procedure associated with equations [Alg. 6.12 and Alg 6.13] stated in the background section, above.

4. An ***else*** statement follows; so if *it*, at line 172, was more than 0, and less than 5, then:

   q is set to be equal to the value of *et*, less the value of *it*.

Line 184:
```
else
q=et-it;
```

5. The value of *temp* is set to be equal to (1/*r*), and then the value of *x* is set to be equal to the value of $temp^2$. The value of *x* therefore becomes equal to $(1/r)^2$.

Line 189:
```
temp=1.0/r;
x=temp*temp;
```

The value of *h0fv* is set to be equal to: 4.343 * log((*a*[*it*-1]**x*+*b*[*it*-1])**x*+1.0); this calculates the $H_{01}(r, j) = 10 \log (1 + (b)r^{-2} + (a) r^{-4})$ for j = *it* -1, as per [Alg 6.13], except that the constant multiplier value 4.343 replaces the constant multiplier value 10.

Why is the 10 replaced with 4.343? The obvious expectation is that it is due to the nonstandard units of angular measure used for *th*, which was used to calculate the values of r1 and r2 in the ***ascat*** subroutine, which later called this subroutine. The angular distance, th, a.k.a. θ, or theta) is calculated and defined as (vertical distance in meters/horizontal distance in meters) in the code, instead of in radians.

Tech Note 101 states that θ is in units of radians. George Hufford, the author of the Algorithm, does not note the use of any such conversion factor in the Algorithm equations 6.13 and 6.14, which use the constant multiplier value 10, that is correct for r1 and r2 calculated with θ defined in units of radians.

In the equation [Alg. 4.62], in Section 4.3.1, "The Function A_{scat}." the angular distance θ is still being specified in radians, as becomes clear in Section 6, equations [Alg 6.13 and 6.14].

How do we convert θ, a.k.a. *th*, to radians? There are 2π radians in a full cycle, or 360°. A radian is defined as the angle subtended at the center of a circle by an arc of circumference that is equal in length to the radius of the circle. Draw this construct on a circle, with one radii of length r on the horizontal plane, and a distance of r on the circumference between the two radii. Now draw a vertical line from the point where the non-horizontal radii touches the circumference of the circle, to a point perpendicular to the horizontal radii, forming a right triangle. The radius then becomes the hypotenuse of a right triangle with an angle, subtended at the center of the circle, between the two radii, of one radian, or 57.2958 degrees. The length of the vertical line is then equal to the sine function of the angle θ, which is equal to the ratio of the length of the vertical line to the length of the hypotenuse of the triangle, which is equal to r. So we can now obtain the length of the vertical line by multiplying sin θ by the hypotenuse length, r. This results in the equation:

$$V = (\sin \theta) * r$$

The length of the horizontal line is then equal to the cosine function of the angle θ, which is equal to the ratio of the length horizontal line of the triangle to the length of the hypotenuse of the triangle, which is equal to r. So we can obtain the length of the horizontal line by multiplying cos θ by the hypotenuse length, r. This results in the equation:

$$H = (\cos \theta) * r$$

We can now obtain the ratio of the vertical length to the horizontal length by dividing the equation for Y by the equation for X. and canceling out the "r" terms:

$$V/H = [(\sin \theta) * r] / [(\cos \theta) * r] = (\sin \theta)/(\cos \theta)$$

In trigonometry, by definition of the tangent function, tan x = (sin x) / (cos x), so the equation becomes:

$$V/H = (\tan \theta) \text{ in radians}$$

This can be used to convert from the angle in radians or degrees, to the ratio used for θ in the code, but we also need to know how to convert from the vertical-distance-to-horizontal-distance ratio (V/H ratio) used for *th*, to radians, in case we later run into a formula that cannot handle the V/H ratio. For this we use the arctan subroutine function:

$$\arctan (V/H) = \theta \text{ in radians (rads)}$$

Here, we will attempt to utilize these conversion ratios.

Line 192: h0fv=4.343*log((a[it-1]*x+b[it-1])*x+1.0);

6. Initiates an **if** statement. If *q* , which holds the difference value between (double) *et* and (integer) *it*, is not equal to zero (which would indicate *et* = *it*), then: *h0fv* is set to be equal to:

   (1.0-*q*) * *h0fv*+*q**4.343*log((*a*[*it*]**x*+*b*[*it*])**x*+1.0)

   Here the value of *h0fv* is interpolated. The H_{01} value for *it* is calculated to be: H_{01}(for *it*) = 4.343 * log((*a*[*it*]**x*+*b*[*it*])**x*+1.0). The term (1 – *q*) interpolates the H_{01} value calculated for *it* , the integer value of η_s, the scatter efficiency factor, to approximate the H_{01} value for *et* = η_s.

   This is calculated as per [Alg 6.13], except that the constant multiplier value 4.343 replaces the constant multiplier value 10. Why? I thought the answer was the units of measure of *th*, used in calculating r_1 and r_2 , and I still hold that opinion. But a single change of a constant value, from 10 to 4.343, will not do the job properly; as the equations use values of $(r_1)^2$, $(r_2)^2$, $(r_1)^4$ and $(r_2)^4$ multiplied by varying constants.

MAJOR PROBLEM NOTE: ***Therefore, since r_1 and r_2 were calculated with a value of th in the wrong units, it appears to this author that the subroutine will produce errant and erratic results; the replacement of the constant value 10 by 4.343 may have occurred in order to produce results that were somewhat close to the empirical results from the field measurements. The author finds no other solid mathematical basis for the change from 10 to 4.343 in the equations in the code.***

How can this be corrected? By using the equation:

arctan (V/H) = θ in radians (rads)

To convert the unit value of th to radians prior to its use in calculating r_1 and r_2; and replacing the constant 4.343 with the correct constant, 10, stated in [Alg. 6.13 and 6.14], in subroutines hof and ascat.

Line 194: if (q!=0.0)

h0fv=(1.0-q)*h0fv+q*4.343*log((a[it]*x+b[it])*x+1.0);

7. Subroutine ***h0f*** ends by returning the H_{01} (*r*, *et*) function output value stored in *h0fv:*

Line 197: return h0fv;

SUBROUTINE HZNS2: A functional explanation, by Sid Shumate.
Revised 16 October, 2010, matching itwom 2.0u.cpp to add post obstruction 2ray calcs.
Previously revised 23 Sept. 2010.
Previous revision Aug. 2010 To correct calculation of location and height of receiver obstructions.

HoriZoNS for ITWOM, subroutine ***hzns2***

Note: Used with point-to-point mode. Called by ***qlrpfl***, mid-routine.

Derived from the Irregular Terrain Model description by George Hufford, 2002, (ITMD), compared to the ITM.cpp prepared by J. D. McDonald and John Magliacane for compilation on unix and linux systems. "Line" numbers refer to the ITM.cpp as line numbered by Bloodshed Software's DevC++ print function. "Alg" numbers refer to the algorithm formula in "The ITS Irregular Terrain Model, version 1.22, the Algorithm" by G. A. Hufford, 1995. "TN101" numbers refer to the formulas in "National Bureau of Standards Technical Note 101; Transmission Loss Predictions for Tropospheric Communications Circuits, Volumes I and II," as revised January 1, 1967. (NBS TN101, or TN101).

From ITMD Sections 44 and 47:

Background; Part I: Tech Note Section 6.4, Equation 6.15:

NBS TN101 defines "launch angles" for the signal path line as it leaves the transmitter site, and as it arrives at the receive site. These launch angles, described as θ_{et}, the angular elevation of the transmit horizon ray, and θ_{er}, the angular elevation of the receive horizon ray, may be determined by field survey or from a terrain profile plot, and are computed in the ITM using:

$$\theta_{et} = [(h_{Lt} - h_{ts}) / (d_{Lt})] - (d_{Lt}/2a) \text{ and } \theta_{er} = [(h_{Lr} - h_{rs}) / (d_{Lr})] - (d_{Lr}/2a) \quad \text{[TN101 6.15]}$$

where:

- h_{Lt}, h_{Lr} are the heights of the horizon obstacle (or obstacle peaks), above mean sea level
- h_{ts}, h_{tr} are antenna elevations above sea level, (i.e. effective height of antenna above ground level plus the ground elevation height above mean sea level)
- d_{Lt}, d_{Lr} are the distances from the terminals (the transmit site and the receive site) to the horizon (obstacle peak).
- a the effective earth's radius (utilized in the c++ code as gme, the effective earth's curvature, where gme is equal to 1/a.)

From the NBS TN101, Volume I, Section 6.4, note that it states: "As a general rule, the location of a horizon obstacle is determined from the terrain profile by using [TN101 6.15] to test all possible horizon locations. The correct horizon point is the one for

which the horizon elevation angle θ_{et} or θ_{er} is maximum. When the trial values are negative, the maximum is the value nearest zero."

Background; Part II: A mathematical proof for the conversion of a measurement of the length ratio of the non-hypoteneuse sides of a right triangle, to an angle measured in radians.

How do we convert θ, a.k.a. *th*, calculated as a vertical to horizontal ratio in rectangular co-ordinates, to radians? There are 2π radians in a full cycle, or 360°. A radian is defined as the angle subtended at the center of a circle by an arc of circumference that is equal in length to the radius of the circle. Draw this construct on a circle, with one radii of length r on the horizontal plane, and a distance of r on the circumference between the two radii. Now draw a vertical line from the point where the non-horizontal radii touches the circumference of the circle, to a point perpendicular to the horizontal radii, forming a right triangle. The radius then becomes the hypotenuse of a right triangle with an angle, subtended at the center of the circle, between the two radii, of one radian, or 57.2958 degrees. The length of the vertical line is then equal to the sine function of the angle θ, which is equal to the ratio of the length of the vertical line to the length of the hypotenuse of the triangle, which is equal to r. So we can now obtain the length of the vertical line by multiplying sin θ by the hypotenuse length, r. This results in the equation:

$$V = (\sin \theta) * r$$

The length of the horizontal line is then equal to the cosine function of the angle θ, which is equal to the ratio of the length horizontal line of the triangle to the length of the hypotenuse of the triangle, which is equal to r. So we can obtain the length of the horizontal line by multiplying cos θ by the hypotenuse length, r. This results in the equation:

$$H = (\cos \theta) * r$$

We can now obtain the ratio of the vertical length to the horizontal length by dividing the equation for Y by the equation for H. and canceling out the "r" terms:

$$V/H = [(\sin \theta) * r] / [(\cos \theta) * r] = (\sin \theta)/(\cos \theta)$$

In trigonometry, by definition, the tangent function is: $\tan \theta = (\sin \theta)/(\cos \theta)$, so the equation becomes:

$$V/H = (\tan \theta)$$

Here we will refer to the units as radians, as in c++ code, the trigonometric functions report out in radians, not degrees. This can be used to convert from the angle in radians, to the ratio used for the take-off angle in the code, but we also need to know how to convert from the vertical-distance-to- horizontal-distance ratio (V/H ratio) used for *th*, to radians. For this we use the arc tangent (a.k.a. $\tan^{-1}$) subroutine function:

$$\text{atan}\ (V/H) = \theta \text{ , in radians (rads)}$$

Conversely, when the need arises to convert the effective earth curvature, gme, to a height to distance ratio, we use:

tan(gme)= earth curvature as a tangent ratio, vertical/horizontal.

These conversion ratios will be used below.

Subroutine Call inputs:

pfl terrain elevation profile array, starting at tx, ending at rcvr, following great circle path, with:
pfl[1] =*enp*, the number of increments
pfl[2] = *xi*, distance per increment
pfl[3] = *z(0)*, the transmitter tower base AMSL, or elevation height
pfl[[np+3] = *z(np)*, the receive location AMSL, the last elevation.

Inputs from (and later outputs to) prop_type &prop structure, consisting of:
(not all structure elements listed here are used by ***hzns***.)

aref the reference attenuation
dist total propagation path distance (in km?)
hg(0) transmitter site radiation center above ground level (RCAGL).
hg(1) receive site radiation center above ground level (RCAGL).
wn the wave number = freq. in MHz/47.7 MHz*m.; units in 1/meters.
dh delta H, (Δh), the terrain irregularity factor
ens refractivity of the atmosphere at sea level
gme effective earth curvature, (actual + refraction)
zgndreal resistance component of earth impedance
zgndimag reactive component of earth impedance
he(0) effective height of transmit antenna
he(1) effective height of receive antenna
dl(0) transmitter antenna horizon (or highest "visible" obstacle) distance
dl(1) receive antenna horizon (or highest "visible" obstacle) distance
the(0) take off angle, at transmit terminal, from the antenna to the transmit horizon or highest "visible" obstacle.
the(1) take off angle, at receive terminal, from the antenna to the receive horizon or highest "visible" obstacle.
kwx the error indicator value, a.k.a. errnum, or error number
mdp the mode of propagation model used. The mdp mode code is:
- 1 point to point mode
1 initializing area prediction mode
0 area prediction mode has initialized and is continuing

defines private, or local, arguments:

wq "false" if q is less than zero during for loop; (true=entire path is Line of Sight.)

np	number of points in pfl elevation array
xi	increment distance between points in pfl elevation array
za	transmitter site radiation center height above mean sea level (RCAMSL), in meters.
zb	receive site antenna center height above mean sea level in meters.
qc	equal to ½ of the effective earth curvature (*gme*); units in 1/meters
q	utility variable; starts as half of the effective earth curvature times the path distance, is redefined several times to represent the working variable at the moment.
sb	distance, in meters, from receive site to elevation point being studied in the for loop
sa	distance, in meters, from transmit site to elevation point being studied in for loop
dr	the path length divided by the sum of 1 plus the ratio of the receive height AMSL to the transmitter height AMSL, i.e. the distance from the transmitter to the reflection point for a 2-ray calculation
dor	first, the distance between the receive site obstruction peak and the 2-way reflection point after the last obstruction, to which the distance from the transmitter to the last obstruction is added to create the distance between the transmitter and the post-obstruction reflection point.
dshh	distance between obstacle peaks, in meters

integers:

rp	Integer value of reflection point, first for l-o-s, later for post-obstruction.

This subroutine:

1. Defines *np*, number of points, to be equal to *pfl[0]*.

Line 1067: np=(int)pfl (0);

2. Defines *xi*, increment distance, in meters/increment, to be equal to *pfl [1]*.

Line 1068: xi=pfl (1);

3. Calculates *za* to be equal to the transmitter site ground elevation height, *pfl[2]*, added to *prop.hg[0]*, the transmitter site radiation center height above ground level (RCAGL). *za* becomes the transmitter site radiation center height above

mean sea level. (RCAMSL). [Note: for dual database use, prop.hg must be adjusted to compensate for the height difference between the pfl array database and the ground level database, and may be a negative number.]

Line 1069: za=pfl[2] +prop.hg[0]

4. Calculates *zb*, the receive site reception center above mean sea level, using pfl [np+2], the last elevation point, and prop.hg[1], the receive site height above ground level. [Note: for dual database, prop.hg must be adjusted to compensate for the height difference between the pfl array database and the ground level database, and may be a negative number.]

Line 1070: zb= pfl[np+2] +prop.hg[1]

5. In subroutine ***alos2***, to avoid having to load the entire pfl array into ***lrprop2***, and then into alos2, we can put the 2-ray coefficients to be used in ***alos2*** and ***mpath*** in the prop array. We can then retrieve these values from the prop array while in ***hzns2***, as well as move the calculation of rp, the reflection point, and hrp, the height of the reflection point, to ***hzns2***. So we have added new in ***hzns2***:

6. Include new parameter values in the prop array:
    a. Set prop.tiw = xi, the increment distance between points in pfl elevation array.
    b. Set prop.ght = za, the transmitter site radiation center height above mean sea level (RCAMSL), in meters.
    c. Set prop.ghr = zb, the receive site antenna center height above mean sea level in meters.

Line new: prop.tiw=xi
prop.ght=za
prop.ghr=zb

7. Calculates *qc* , to be equal to one-half of *gme*, the effective actual earth curvature. The constant *gme, (a.k.a. prop.gme),* equal to 157e-9 /meter, is the inverse of earth's effective radius.

Line 1071: qc= 0.5*prop.gme

8. Calculates *q* to be equal to *qc*, half of the effective earth curvature, times *prop.dist*, the propagation path total distance.

Line 1072: q=qc*prop.dist;

Why does this use ½ of the earth curvature? For each degree of effective earth curvature, the grazing angles between the transmitter and receiver increases by ½ degree. This can be seen if considering the case where a transmitter is at the north pole, and the receiver is on the equator. The receiver is located theoretically ¼ of the circumference around the earth from the transmitter, where the earth curvature is $-\pi/2$ radians (-90 deg.). For this situation, compared to horizontal at the north pole, the downward angle from horizontal at the north pole to the equator is $-\pi/4$ radians (-45 deg.), or ½ of the earth curvature angle. Similarly, if the transmitter is at the north pole, and the receiver is at the south pole, the earth curvature is $-\pi$ radians (-180 deg.), and the transmitter look down angle increases to the maximum of $-\pi/2$ radians (-90 deg.).

9. Temporarily presets *prop.the[1]* , which will later represent the receive antenna angle of departure toward the horizon, to be equal to the height of the receive antenna, *zb*, above the height of the transmit antenna, *za*, divided by the path distance, *prop.dist*. If the receive antenna is below the transmit antenna, this will be a negative number. Provides the change in height between the receive antenna and transmit antenna divided by the path distance, or the ratio of difference in height to the path distance; this can also be stated as representing the flat-earth tangent of the angle *prop.the[0]*, the transmitter grazing angle. Units should be in meters for the heights and distances. This is more accurately calculated in the ITWOM by taking the arctangent of this calculation to obtain the angle in radians.

Line 1073: prop.the[1] =atan((zb-za)/prop.dist);

10. Presets *prop.the[0]* , the transmitter look-up, or grazing, angle to be equal to *prop.the[1]* , the receiver grazing angle calculated above, less the value held by utility argument q, which at this moment holds the additional angle due to the effective earth curvature. This adds the effective earth curvature angle between the transmitter and receiver to the starting-point transmitter look-up angle. If the transmit antenna is below the receive antenna, this will be a negative number.

11. Then presets *prop.the[0],* the transmitter take-off angle, to be the change in height between the transmit antenna and receive antenna divided by the path distance, less the path distance divided by the effective earth radius, or:

In the ITM:

$$\text{the[0]} = \frac{\text{difference in height between tx and receiver}}{\text{total path distance}} - \frac{0.5 * \text{path distance}}{\text{effective earth radius}}$$

In the ITWOM:

$$\text{the[0]} = \arctan\left(\frac{\text{height difference between tx and receiver}}{\text{total path distance}}\right) - \frac{0.5 * \text{path distance}}{\text{effective earth radius}}$$

[TN101 6.15a] Units should be in radians. But the first term in the ITM version is in meters of height per meter of distance, i.e. the tangent of an angle, and the

second is in radians. In the ITWOM, both are in radians due to the use of the arctangent above.

Line 1074: prop.the[0] =prop.the[1] – q;

12. Recalculates *prop.the[1]* to be equal to the negative sum of *prop.the[1]* and the current value held by *q*, or:

In the ITM:

$$\text{the[1]} = \frac{\text{difference in height between receiver and tx,}}{\text{total path distance}} - \frac{0.5*\text{path distance}}{\text{effective earth radius}}$$

[TN101 6.15b] Units should be in radians. But the first term is in meters of height per meter of distance, the tangent of an angle, and the second term is in radians. We use the arctangent in the ITWOM to obtain the more correct value for the angle.

In the ITWOM:

$$\text{the[1]} = \arctan\left(\frac{\text{height difference between receiver and tx,}}{\text{total path distance}}\right) - \frac{0.5*\text{path distance}}{\text{effective earth radius}}$$

Line 1075: prop.the[1] = – prop.the[1] – q;

Note: Either *the[1]* or *the[0]* will be negative, indicating which site is lower, unless they are equal, indicating that the transmitter is the same height as the receiver. However, reportedly, *the[0] = the[1]* may cause a calculation problem later.

Here we have a take-off angle calculation problem. This is not due to the error specified by Hammett and Edison in several comments to the FCC regarding the use of Longley Rice for TV reception prediction; that is a problem with the wrap-around input-output software. These problems are in the ITMDLL 1.2.2 core.

The calculation problem here can be tracked back all the way to Tech Note 101, Section 6.4; and relate to equation [TN101 6.15]. The problem appears to come from not keeping track of units, especially when mixing co-ordinate systems.

The first term of the equations in 6.15 is calculated as a vertical distance change over a horizontal distance change ratio, i.e. a tangent of an angle, in rectangular co-ordinates over theoretical flat earth. But TN101 specifies, in Section 6 in the paragraph preceding [TN 6.15], that <u>all</u> angles are to be in radians unless otherwise specified.

The second term, (1/2*path distance/effective earth's radius), adds the angle reduction due to the curvature of the earth and refractivity; it divides a partial circumference of a circle by a radius, and is therefore, by definition, in polar geometric co-ordinates, resulting in output units in radians.

For a smooth earth model calculation, this is an approximation that works fine, as the error at a smooth earth horizon distance, from not converting the term: (distance in height between terminals/total path distance) to radians, does not show up until the eighth decimal place. But for Irregular Terrain Model calculations with a nearby obstruction, or take off angles calculated at the base of a tall tower, skyscraper, or mountain on which is located a transmit terminal, the error becomes significant. So the ITWOM uses arctan (in c++, the atan command) to convert the flat earth angle between the transmitter and receiver to a value in radians. The second place we apply the correction occurs when recalculating the take-off angles to obstruction peaks for maximum accuracy.

13. Presets both *prop.d1(0)*, the actual transmitter to horizon distance, and *prop.d1(1)*, the actual receiver to horizon distance, to be equal to the propagation path distance *prop.dist*, which is true for a clear line-of-sight path only. If the path is line-of-sight, this will remain the default value.

Line 1076: prop.dl[0] = prop.dist;
prop.dl[1] = prop.dist;

14. Presets:

a. hht, the height of the highest obstacle or horizon "visible" from the transmitter site, and
b. hhr, the height of the highest obstacle or horizon "visible" from the receive site, to both be 0.0 meters.
c. Also presets prop.los to be equal to 1, indicating a line-of-sight condition.

Line : hht=0.0;
hhr=0.0;
prop.los=1;

15. If there are at least two points, a minimum of the transmitter and receiver, or more, then:

Line 1079: if (np>=2)

16. The value of *sa* is preset to be equal to 0.0. The argument *sa* will represent the distance from the transmitter site to a elevation point being considered. Setting *sa* to 0.0 starts the following loop calculation from the transmitter site.

Line 1081: *sa* = 0.0;

17. The value of *sb* is preset to start at the path distance, *prop.dist*. The argument *sb* will hold the distance from the receive site to an elevation point being considered. This second **for** loop works from the receive site toward the transmit site.

Line : sb=prop.dist;

18. A **for** loop is started that starts with i = 1 and continues until i is no longer less than the number of elevation points *np*. This causes it to consider each elevation point individually along the path from the transmitter to the receiver, starting with the first point past the transmitter.

Line 1085: for (j=1; j<np; j++)

a. *sa* starts at 0.0, and is increased at the start of each pass by an amount equal to *xi*, the interval distance between elevation points, measured in meters. *sa* then represents the distance between the transmit terminal and the elevation point being studied, with units in meters.

Line 1087: sa+=xi;

b. *q* will represent the difference between the height of the profile elevation point height being considered on this loop, and the height represented by the theoretical height of a theoretical point at the same distance from the transmit antenna as the profile elevation point, calculated from the current transmitter take off (grazing) angle value held in prop.the[0]. It is calculated by taking *pfl[j+2]*, the elevation point height being studied on this loop, and subtracting the combination of:
   i. the effective curvature angle in radians of the earth (actual + refraction) between the transmitter and the point being studied, (qc*sa), added to the take-off angle prop.the[0], all multiplied by:
   ii. The distance between the transmitter and the profile elevation point, *sa*. NOTE: This is not a precise calculation; multiplying the angle by the distance is a polar, not Cartesian, solution, and is, for this purpose, an adequate approximation, not a rigorous trigonometric solution.
   iii. Finally, the transmitter elevation above mean sea level (*za)* is also subtracted.

If the value held by *q* is greater than zero, there is an obstruction at this location that blocks the transmitter's view of the receiver and is higher than any other obstruction found so far; and the vertical distance *q* represents the vertical distance change at the obstruction that must be added to the horizon take-off angle to compute the take off angle from the terminal to the peak of the most recently found candidate to be the highest "visible" transmitter obstruction found, during the sweep along the radial occurring in this loop.

Line 1089: q=pfl[j+2]–(qc*sa+prop.the[0])*sa – za;

c. If a point is reached during the for loop where *q* is greater than 0.0, indicating that either the actual horizon, and/or a first diffraction point as

seen from the transmitter, has been reached, the value of prop.los, line of site, is set to zero, *prop.the[0]*, the transmitter take-off angle, is increased by an amount equal to *q/sa*, to direct it toward the top of the horizon point and/or diffraction point, and prop.dl[0] is made equal to sa, the path length between the tx and elevation point being studied, now the transmitter horizon/first diffraction point.

d. Angle prop.the is limited to a maximum of 1.569 radians.
e. Prop.hht, the transmitter side obstruction height, is reset to be equal to pfl[j+2].
f. The Boolean argument *wq* is set to "false", indicating that the entire path will NOT be a line-of-sight study.

Line 1091:
```
if (q>0.0)
{
    prop.los=0;
    prop.the[0]+=q/sa;
    prop.dl[0]=sa;
    prop.the[0]=mymin(prop.the[0],1.569);
    prop.hht=pfl[j+2];
    wq=false;
}
```

In the ITM *hzns* subroutine, for the purposes of calculating the receiver grazing angle, if a horizon or obstruction was found, the same terrain point found to be the transmitter horizon peak or the highest obstruction visible from the transmitter, was assumed to be the peak of the receive horizon or highest obstruction visible from the receiver. This is a very poor simplification.

In the ITWOM *hzns2* subroutine, the grazing angle to the receiver horizon, or highest obstruction "visible" from the receiver, is calculated separately, if a transmitter horizon or obstruction is found:

(2). If wq is not true, (i.e. if wq is "false", indicating that the path includes a horizon or at least one obstacle), a ***for*** loop is initiated to calculate the receiver take off (a.k.a. grazing) angle for a receiver beyond the obstruction or horizon. The counter is set to 1 to start at the last interval point before the receive site, and the interval points considered will step from the one nearest the receiver, toward the transmitter.

Line :
```
If (!wq)
{
    for(i=1; i<np i++)
    {
```

For each pass through the for loop:

a. The distance sb has an interval's width subtracted from it.
b. *q* will represent the difference between the height of the profile elevation point height being considered on this loop, and the height represented by the theoretical height of a theoretical point at the same distance from the receive antenna as the profile elevation point, calculated from the current receive take off (grazing) angle value held in prop.the[1]. It is calculated by taking *pfl[np+2 – i]*, the elevation point height being studied on this loop, and subtracting the combination of:
    iv. the effective curvature angle in radians of the earth (actual + refraction) between the receiver and the point being studied, (qc*(prop.dist - sb)), added to the receiver antenna take-off angle prop.the[1], all multiplied by:
    v. The distance between the receiver and the profile elevation point, (prop.dist-*sb)*. NOTE: This is not a precise calculation; multiplying the angle by the distance is a polar, not Cartesian, solution, and is, for this purpose, an adequate approximation, not a rigorous trigonometric solution.
    vi. Finally, the receiver elevation above mean sea level (*zb)* is also subtracted.

If the value held by *q* is greater than zero, there is an obstruction at this location that blocks the receiver antenna's "view" of the transmit antenna, and is higher than any other obstruction found so far; and the vertical distance *q* represents the vertical distance change at the obstruction that must be added to the receive horizon take-off angle to compute the take off angle from the receive terminal to the peak of the most recently found candidate to be the highest "visible" receive obstruction found, during the reverse sweep along the radial occurring in this loop.

Line : sb= – xi;
q=pfl[np+2-i] – (qc*(prop.dist-sb)+prop.the[1])*(prop.dist-sb) – zb;

If the value of q is greater than zero, indicating that a new horizon or new "highest obstruction visible from the receiver" has been found, then:

a. The receive antenna "look up", or grazing, angle, prop.the[1], is set to equal the existing value of the look up angle, plus the additional new angle increase approximated by dividing the value of q by the distance from the receiver to the horizon or obstacle.
b. The value of prop.the[1] is then limited to be between 1.57 and –1.568 radians, to avoid computer calculation errors.
c. The value of the horizon height for the receiver, prop.hhr, is set to equal the terrain height at the obstacle, in meters AMSL, read from the profile at pfl(np+2-i).
d. The value of prop.dl[1] is made equal to prop.dist less sb, and limited to be no less than zero, (which would indicate that the receiver horizon

is at the obstruction peak), and the receive horizon distance is the distance from the receive site to the last obstruction peak visible from the receive antenna.

e. At the end of the if loop, prop.the[0] should then indicate the flat-earth take-off angle to the most distant obstruction peak "visible" (to the RF signal) from the transmitter site, and prop.dl[1] should indicate the distance to the most distant obstruction peak visible from the receive site. Because of rounding errors, this distance is limited to a minimum of 0.0 meters.

```
Line    :       If (q>0.0)
                {
                        prop.the[1] +=q/(prop.dist – sb);
                        prop.the[1]=mymin(prop.the[1],1.57);
                        prop.the[1]=mymax(prop.the[1], – 1.568);
                        prop.hhr=pfl[np+2 – i];
                        prop.dl[1]=mymax(0.0,prop.dist – sb);
                }
            }
```

19. The grazing angles are, for speed, calculated using an approximation that may slightly miscalculate the final value of the grazing angles. Now that that is done, before exiting the if loop, if(!wq), that is executed when obstacles are found, we recalculate the grazing angles for maximum accuracy, all in units of radians, using the transmitter RC-AMSL, *sa*, the receiver RC-AMSL, **sb**, and the newly calculated values for the transmitter obstacle height AMSL, *prop.hht*, the receiver obstacle height AMSL, *prop.hhr*, the transmitter to obstacle distance in meters, prop.dl[0], and the receiver to obstacle distance in meters, prop.dl[1].

```
Line    :       prop.the[0]=atan((prop.hht-sa)/prop.dl[0])-0.5*gme*prop.dl[0];
                Prop.the[1]=atan((prop.hhr-sb)/prop.dl[1])-0.5*gme*prop.dl[1];
        }
    }
```

20. Returning to the calculation of coefficients for ***alos2*** and **mpath**, it is now possible to reasonably estimate the reflection point for 2-ray calculations for line-of-sight or post-obstruction scenarios. If one or more obstacles are found in the path, indicated in an **if** statement by the receive site horizon path, *prop.dl[1]*, being shorter than the total path distance, *prop.dist*, then:

```
Line(new):      if((prop.dl[1])<(prop.dist))
                {
```

21. Here we calculate coefficients for use in subroutine ***mpath***, for 2-ray calculations after an obstruction. The last obstruction peak is treated as if it were a secondary transmitter site. To obtain the location (in terms of the database interval number) for the reflection point *rp* in ***mpath*** after a single or 2nd obstruction, and then retrieve this terrain height value from pfl and store this height in *prop.rph2*, we define argument *dr* as the distance in meters from the last obstruction peak, which we treat as a pseudo-transmitter site, to the reflection point. For a reflection off of the earth's surface, we simplify to say that the grazing angle of the line from the transmit antenna to the reflection point, with respect to a horizontal reflecting surface at the reflection point, is equal to the grazing angle of the line from the receive antenna to a horizontal reflecting surface at the reflection point. Therefore, using the receive obstruction height AMSL in meters, *prop.hhr*:

prop.hhr/dr = tangent of the grazing angle = zb/(prop.dl[1] – dr)

Where: za is the transmitter site radiation center height above mean sea level (RCAMSL), in meters.
zb is the receive site antenna center RCAMSL in meters.
prop.dl[1] is the distance from the highest obstruction peak visible from the receive site, to the receive site.

This does assume that the reflection point height is near sea level. A better approximation is to assume that the reflection point height is near the height of the receiver ground level, a reasonable assumption when the receive height is 10 meters or less. Recalculating, then:

(prop.hhr-pfl[np+2])/dr = tangent of the grazing angle = prop.hg[1]/(prop.dl[1] – dr)

From this, we get: (prop.dl[1] – dr)/dr = prop.hg[1]/(prop.hhr-pfl[np+2])
Then: prop.dl[1] = dr(prop.hg[1]/(prop.hhr-pfl[np+2]) + dr
prop.dl[1]=dr(1+(prop.hg[1]/(prop.hhr-pfl[np+2]))

So: dr = prop.dl[1]/(1+ prop.hg[1]/(prop.hhr-pfl[np+2]))

To which we add the distance from the transmitter to the last obstruction peak, derived from the total path distance, *prop.dist*, less the receive site horizon distance, *prop.dl[1]*, redefining dr as the distance from the main transmit site to the 2-ray reflection point, and we have:

dr = prop.dist – prop.dl[1] + prop.dl[1]/(1+ prop.hg[1]/(prop.hhr-pfl[np+2]))

or: dr = prop.dist – prop.dl[1](1 – 1/(1+ prop.hg[1]/(prop.hhr-pfl[np+2]))

Line new: dr= prop.dist – (1 – 1/(1+ prop.hg[1]/(prop.hhr-pfl[np+2])))* prop.dl[1];
}

22. An else statement follows, so for an unobstructed, or line-of-sight scenario:

Line new: else
{

23. To obtain the location (in terms of the database interval number) for the reflection point *rp*, for a line-of-sight path, as used in subroutine ***alos2***, and then retrieve this terrain height value from pfl and store this height in *prop.rph*, we define *dr* as the distance in meters from the transmitter site to the reflection point. For a reflection off of the earth's surface, we simplify to say that the grazing angle of the line from the transmit antenna to the reflection point, with respect to a horizontal reflecting surface at the reflection point, is equal to the grazing angle of the line from the receive antenna to a horizontal reflecting surface at the reflection point. Therefore:

Za/dr = tangent of the grazing angle = zb/(prop.dist – dr)

Where: za is the transmitter site radiation center height above mean sea level (RC-AMSL), in meters.
zb is the receive site antenna center RC-AMSL in meters.
prop.dist is the total path distance from transmitter site to receive site.

This does assume that the reflection point height is near sea level. A better approximation is to assume that the reflection point height is near the height of the receiver ground level, a reasonable assumption when the receive height is 10 meters or less. Recalculating, then:

(za-pfl[np+2])/dr = tangent of the grazing angle = prop.hg[1]/(prop.dist – dr)

From this, we get: (prop.dist – dr)/dr = prop.hg[1]/(za-pfl[np+2])
Then: prop.dist = dr(prop.hg[1]/(za-pfl[np+2]) + dr
prop.dist=dr(1+(prop.hg[1]/(za-pfl[np+2]))

So: dr = (prop.dist)/(1+ prop.hg[1]/(za-pfl[np+2]))

Line new: dr=(prop.dist)/ (1+ prop.hg[1]/(za-pfl[np+2]));
}

24. The else statement ends, and dr is used to compute *rp*, the interval value in the pfl array corresponding to the reflection point. The argument dr, the distance in meters to the reflecting point, is divided by the width of a database interval, in meters, resulting in a value for the distance in intervals from the transmitter to the reflection point for a line-of-sight (l-o-s) calculation. The value of the argument *rp* is then stored in *prop.rpl*:

rp = 2+**int**(**floor**(0.5+*dr*/*xi*));
prop.rpl = rp;

Note: floor, a math.h subroutine not previously used in this set of subroutines, returns the largest integral value less than the value of: (0.5+*dr*/*xi*). The 0.5 added in, causes it to operate like a rounding function. The int causes the answer to be an integer, for which *rp* is declared. The integer value two is added to *rp* because the elevation values in the c++ pfl array start at pfl[2]. For fortran, this is 3, as the elevation values in the fortran pfl array start at pfl[3]. The integer value of *rp* will then be used to retrieve the elevation height above mean sea level (AMSL) from the elevation database array *pfl*, storing it in prop.rph:

Line new:
```
rp=2+int(floor(0.5+dr/xi));
prop.rpl = rp;
prop.rph=pfl[rp];
prop.rpd=dr;
```

25. ***hzns*** returns:

Outputs to prop_type structure:

a.	*prop.the[0]*	horizon take-off angle from transmitter
b.	*prop.the[1]*	horizon take-off angle from receiver
c.	*prop.dl[0]*	distance from transmitter to horizon (or 1st obst.)
d.	*prop.dl[1]*	distance from receiver to horizon (or last obstacle)
e.	*prop.rpl*	location of 2-ray reflection point in array pfl[]
f.	*prop.rph*	height of 2-ray reflection point in meters AMSL
g.	*prop.rpd*	distance from transmitter to 2-ray reflection point

And the subroutine has completed its run.

Finally, a note regarding a famous protest:

Hammett and Edison, in their October of 2004 comments submitted to the Federal Communications Commission (FCC) in CS Docket 98-201, regarding the use of Longley-Rice in calculating Grade B TV Signal Coverage, stated in paragraph 20:

"This ongoing work has convinced us that the implementation of the L-R model is even more flawed than had been originally suspected. For example is has come to light that the OET-69 software calculates the depression angle to a calculation point using the sources height above ground, not its height above sea level. This coding mistake by itself will introduce errors of perhaps 10-20 dB in the calculation results."

The code studied here does not, in and of itself, contain this error. In step 3 above, *za*, the transmitter height used for the angle calculations, is the transmit antenna height above sea level, determined by adding the transmitter site ground elevation height, *pfl[2]*, to *prop.hg[0]*, the transmitter site radiation center height above ground level (RCAGL). The same is true of *zb*, the receive antenna height above sea level. The original claimed source of the error spoken of by Hammett and Edison still exists, it exists in the original data preparatory subroutines written for OET-69 to prepare the information for the ITM subroutines, not in the ITM 1.2.2 code, including the new version 7, publicly released by the NTIA since 2003.

If an FCC engineer requests it, a new updated subroutine command allows for "corrected" calculation to be made.

SUBROUTINE LRPROP: A functional explanation, by Sid Shumate.

Last Revised: July 26, 2007.

Longley-Rice Propagation subroutine ***lrprop***.

Note: Used with both point-to-point and area modes. For point-to-point mode, called at end of ***qlrpfl.*** Calls subroutines ***adiff, alos, ascat, mymin,*** and ***mymax***.

Derived from the Irregular Terrain Model description by George Hufford, 2002, (ITMD), compared to the ITM.cpp prepared by J. D. McDonald and John Magliacane for compilation on unix and linux systems. "Line" numbers refer to the ITM.cpp as line numbered by Bloodshed Software's DevC++ print function. "Alg" numbers refer to the algorithm formula in "The ITS Irregular Terrain Model, version 1.22, the Algorithm" by G. A. Hufford, 1995. "ITS67" numbers refer to the algorithm formulas in "ESSA Technical Report ERL 79-ITS 67, Prediction of Tropospheric Radio Transmission Over Irregular Terrain, A Computer Method – 1968" by A.G.Longley and P.L.Rice.

From ITMD Sections 4, 5 to 9, 15; 16, and 17 with 18 and 19, and 20 with 21, 22 and 23:

The Longley-Rice propagation program. This is the basic program; it returns the reference attenuation *aref.* This is the "PaulM" version of lrprop; "Freds lrprop", found in version 1.2.2, was removed from version 7.0 when it was released on June 26, 2007.

Call inputs:

d	path distance
Prop_type	
&prop	array *prop* with array elements:
propa_type	
&propa	array *propa* with array elements:

defines private, or local, arguments:

wlos	static boolean argument; true if line-of-sight coefficients have been calculated.
wscat	static boolean argument; true if troposcatter coefficients have been calculated.
dmin	static double argument; minimum acceptable path distance length in meters
xae	static double argument; value calculated in Step 11.

prop_zgnd	sub array zgnd (average ground impedance) with elements:		
		prop.zgndreal	resistance element of ground impedance
		prop.zgndimag	reactive element of ground impedance
a0			
a1			
a2			
a3			
a4			
a5			
a6			
d0			
d1			
d2			
d3			
d4			
d5			
d6			
wq	Boolean argument; indicates whether general case 1 or general case 2 applies in calculating line of sight coefficients.		
q	working variable; holds various values during several operations		
j	either 0 (1 in Fortran), for transmit terminal, or 1 (2 in Fortran) for receive terminal		

This subroutine:

Uses *d*, prop_type, propa_type, and other information in arrays prop and propa in order to calculate aref, the reference attenuation (radio signal strength loss) along the path between a transmit site and a receive location.

1. An **if** statement is initiated; it operates from lines 675 to 714. If prop.mdp, the mode of the propagation model, is not equal to zero, (zero would have indicated that the area prediction mode has initiated and is continuing), then the mode of the propagation model is either 1, which would initialize the area prediction mode, or –1, which indicates the program is running in the point to point mode. If prop.mdp = 0, the program proceeds to the **for** loop on line 677. If prop.mdp is not zero, then:

Line 675: if (prop.mdp!=0)

2. A **for** statement is initiated with two loops, j=0 and j=1.

a. The first loop sets dls[0], the distance from the transmitter site to the smooth earth horizon, to be equal to the square root of 2 times prop.he[0] divided by prop.gme.

The algorithm formula comes from: "ESSA Technical Report ERL 79-ITS 67, Prediction of Tropospheric Radio Transmission Over Irregular Terrain, A Computer Method – 1968" by A.G.Longley and P.L.Rice, where it states on page 12;
"When individual path profiles are not available, median values of the horizon distances $d_{L1,2}$ are estimated as functions of the median effective antenna heights h_{e1} and h_{e2} determined above, the terrain irregularity factor Δh, and the smooth-earth horizon distances D_{Ls1} and D_{Ls2}. The smooth earth distance from each antenna to its horizon over a <u>smooth</u> <u>earth</u> is defined as:

$$D_{Ls1,2} = (.002 * a * h_{e1,2})^{.5} \text{ in km.} \qquad \text{[ITS67 (5a)]}$$

where the effective antenna heights $h_{e1,2}$ are in meters and the effective earth's radius a is in kilometers, as defined by (1)."

NOTE: Note the first phrase of the quote above, authored by Longley and Rice. Ms. Longley and Mr. Rice did not intend this formula to be use where individual path profiles are available; therefore, this subroutine is eligible for revision and correction. For more information, please see the notes in the chapter on subroutine *qlrpfl*, which also uses these formulas for point-to-point calculations.

Converting ITS67 (5a) for meters instead of km, we obtain:

$$D_{Ls1,2} = (2 * a * h_{e1,2})^{.5} \text{ in meters.}$$

As derived in the chapter on subroutine ***qlrpfl***, in step 11:

$$a = 1/ gme.$$

Where: *a* is the earth's effective radius, in meters, and
g*me* is the earth's effective curvature, in units of 1/meters.

The formula then becomes:

$$D_{Ls1,2} = (2 * h_{e1,2}/gme)^{.5} \text{ in meters.} \qquad \text{[Alg. 3.5]}$$

And in the computer, he[0] is h_{e1}, *gme* is stored in array *prop* at *prop.gme*; and the result, $D_{Ls1,}$ is stored in array *propa* at *propa.dls[0]*.

b. Similarly, the second loop sets dls[1], the distance from the receive site to the smooth earth horizon, equal to the square root of 2 times prop.he[1] divided by prop.gme.

Line 677:
```
for (j=0; j<2; j++)
        propa.dls[j]= sqrt(2.0*prop.he[j]/prop.gme);
```

3. The program then proceeds to:

a. sets propa.dlsa, the sum of the smooth-earth horizon distance, equal to the sum of propa.dls[0] and propa.dls[1], which were calculated in step 2 above, based on:

The sum of the smooth-earth horizon distance is

$$D_{Ls} = D_{Ls1} + D_{Ls2} \qquad \text{[ITS67 (5b) or Alg. 3.6]}$$

b. sets propa.dla, the total distance between the antennas and their horizons, equal to the sum of propa.dl[0] and propa.dl[1], based on:

The total distance, $d_{L,}$ between the antennas and their horizons is

$$d_L = d_{L1} + d_{L2} \qquad \text{[ITS67 (5d) or Alg. 3.7]}$$

c. sets propa.tha to be equal to the greater of either (1) the sum of the theta angles stored in prop.the[0] and prop.the[1], or (2) the result of multiplying (– propa.dla (calculated in step 3(b)), times prop.gme, the effective earth's curvature):

tha= greater of: (the1+the1) or (-dla*gme) [Alg. 3.8]

d. sets the Boolean value of *wlos* and *wscat* to be false, as per instruction in ITMD Section 6:

Line 680:
```
propa.dlsa=propa.dls[0]+propa.dls[1];
propa.dla=prop.dl[0]+prop.dl[1];
propa.tha=mymax(prop.the[0]+prop.the[1],-propa.dla*prop.gme);
wlos=false;
wscat=false;
```

<u>In Steps 4 through 9, the program checks the parameter ranges of the input values, as per instructions in ITMD Section 7.</u>

4. An **if** statement is initiated to check if the frequency is within range; the wave number, prop.wn, which is derived from the frequency in step 2 of subroutine ***qlrps***, is checked to see if it is less than .838 (equivalent to a frequency of 40 MHz) or greater than 210 (equivalent to a frequency of 10 GHz). If prop.wn is outside of the range, prop.kwx, the error marker, is set to equal the greater of: the existing value of prop.kwx; or 1.

Line 686:
```
if (prop.wn<0.838 || prop.wn>210.0)
prop.kwx=mymax(prop.kwx,1);
```

5. A **for** statement is initiated with two loops, j=0 and j=1.

a. An **if** statement is initiated to check if hg[0], the transmitter antenna height above ground level, is within range; if hg[0] is less than one meter or greater than one kilometer, prop.kwx, the error marker, is set to equal the greater of: the existing value of prop.kwx; or 1. A value of kwx = 0 indicates: no warning; kwx = 1 indicates: caution; parameters are close to limits.

b. An **if** statement is initiated to check if hg[1], the receiver antenna height above ground level, is within range; if hg[1] is less than one meter or greater than one kilometer, prop.kwx, the error marker, is set to equal the greater of: the existing value of prop.kwx; or 1.

Line 689:
```
for (j=0; j<2; j++)
if (prop.hg[j]<1.0 || prop.hg[j]>1000.0)
prop.kwx=mymax(prop.kwx,1);
```

6. A **for** statement is initiated with two loops, j=0 and j=1.

a. A three-way **if** statement is initiated to check if the[0], the transmitter antenna take off angle theta, is within range; if either:
   (1) the absolute value of the[0] is greater than 0.2,
   (2) prop.dl[0], the distance from transmitter to horizon,, is < less than 1/10 of propa.dls[0], smooth earth distance from transmitter to horizon,
   (3) prop.dl[0] is > 3.0 * propa.dls[0]

   Then prop.kwx, the error marker, is set to equal the greater of: the existing value of prop.kwx; or 3, a value of kwx = 3 indicating that internal calculations show parameters out of range.

b. A three-way **if** statement is initiated to check if the[1], the receiver antenna take off angle theta, is within range; if either:

(1) the absolute value of the[1] is greater than 0.2,
(2) prop.dl[1], the distance from transmitter to horizon,, is < less than 1/10 of propa.dls[1], the smooth earth distance from transmitter to horizon,
(3) prop.dl[1] is > 3.0 * propa.dls[1]

Then prop.kwx, the error marker, is set to equal the greater of: the existing value of prop.kwx; or 3.

Line 693:
```
for (j=0; j<2; j++)
    if (abs(prop.the[j]) >200e-3 || prop.dl[j]<0.1*propa.dls[j] ||
    prop.dl[j]>3.0*propa.dls[j] )
        prop.kwx=mymax(prop.kwx,3);
```

7. A seven-way **if** statement is initiated to check the ranges of *ens, gme, zgnd,* and *wn*. If either:
   a. *prop.ens,* the surface refractivity of the atmosphere, is less than 250.0 or greater than 400;
   b. *prop.gme,* the effective earth's curvature, is less than 75e–9 or greater than 250e–9;
   c. *prop.zgnd.real,* the surface transfer impedance real, (or resistance) component is less or equal to the absolute value of *prop.zgnd.imag*, the imaginary (or reactance) component;
   d. *prop.wn,* the wave number, is less than 0.419 (equal to a frequency of 20 Mhz) or greater than 420 (equal to a frequency of 20 Mhz);

Then prop.kwx, the error marker, is set to 4, indicating parameters out of range.

Line 697:
```
if (prop.ens < 250.0 || prop.ens > 400.0 || prop.gme < 75e-9 || prop.gme >
250e-9 || prop_zgnd.real() <= abs(prop_zgnd.imag()) || prop.wn < 0.419 ||
prop.wn > 420.0)
    prop.kwx=4;
```

8. A **for** statement is initiated with two loops, j=0 and j=1.

   a. An **if** statement is initiated to check if hg[0], the transmitter antenna height above ground level, is within its maximum range; if hg[0] is less than one-half meter or greater than three kilometers, prop.kwx, the error marker, is set to equal 4.

   b. An **if** statement is initiated to check hg[1], the receiver antenna height above ground level, as in (a.) above.

Line 700:
```
for (j=0; j<2; j++)
if (prop.hg[j]<0.5 || prop.hg[j]>3000.0)
    prop.kwx=4;
```

9. The value of *dmin* is set to be equal to five times the absolute value of [(prop.he[0] – prop.he[1])], i.e. equal to five times the value of the difference in height between the effective height of the transmit antenna and the receive antenna. The *abs* command ignores any negative sign in the result, causing the result to always be a positive value.

Line 704: dmin=abs(prop.he[0]-prop.he[1])/200e-3;

<u>From steps 10 through 20, the coefficients for the Diffraction Range are calculated:</u>

10. The program calls ***adiff*** with inputs (0.0,prop,propa) .

The subroutine ***adiff*** returns *adiff*, the "diffraction attenuation" at the distance *d*.

The value of *q* is set to be equal to the returned value *adiff*. (Note: since the input d = 0.0, the returned value of *adiff* will be 0.0 for point-to-point mode. See subroutine ***adiff***.

Line 705: q=adiff(0.0,prop,propa);

11. *xae* is set to be equal to: $(prop.wn*(prop.gme*prop.gme)^{-1/3}$ [Alg. 4.2]
Where:
prop.wn is the wave number, equal to the frequency in MHz/47.7.
prop.gme is the effective earth's curvature.

Line 707: xae=pow(prop.wn*(prop.gme*prop.gme),-THIRD);

12. *d3* is set to be equal to the greater of *propa.dlsa* or (1.3787 * *xae* + *propa.dla*): [Alg. 4.3]
where:
Propa.dlsa is the distance value set at line 680, the total smooth earth horizon distance.
xae value was set at line 707.
Propa.dla is the total horizon distance.

Line 708*:* d3=mymax(propa.dlsa,1.3787*xae+propa.dla);

13. *d4* is set to be equal to *d3* plus 2.7574 times *xae* [Alg. 4.4]

Line 709*:* d4=d3+2.7574*xae;

14. The program calls ***adiff*** with inputs (*d3*,prop,propa) . [Alg. 4.6]

The subroutine ***adiff*** returns *adiff*, the "diffraction attenuation" at the distance *d*. *a3* is set to be equal to *adiff;*

Line 710: a3=adiff(d3,prop,propa);

15. The program calls ***adiff*** with inputs (*d4*,prop,propa) . [Alg. 4.6]
The subroutine ***adiff*** returns *adiff*, the "diffraction attenuation" at the distance *d*. *a4* is set to be equal to *adiff*

Line 711: a4=adiff(d4,prop,propa);

16. *propa.emd* is set to be equal to: (*a4–a3)/(d4*–d*3*) [Alg. 4.7]

Line 712: propa.emd=(a4-a3)/(d4-d3);

17. *propa.aed* is set to be equal to: (*a3 – propa.emd*d3*) [Alg. 4.8]

Line 713: propa.aed=a3-propa.emd*d3;
}

18. The first **if** statement has run its course. A new **if** statement is initiated, as per Section 5 of the ITMD, stating that if prop.mdp is greater than or equal to zero, then:

a. prop.mdp is set to be equal to zero, indicating the area mode has initiated and will continue, and;
b. prop.dist is set to be equal to *d*, the path distance.

Line 716:
```
if (prop.mdp>=0)
{
prop.mdp=0;
prop.dist=d;
}
```

19. A third **if** statement is initiated. It has three embedded **if** statements that check the path distance [see Section 8 of the ITMD]; so if *prop.dist*, the value of *d* the path distance, is greater than zero, and:

a. if *prop.dist* is greater than 1,000 kilometers, then:
prop.kwx, the error value, is set to be equal to the higher value of *prop.kwx* or 1;

b. if *prop.dist* is less than *dmin*,then;
prop.kwx is set to be equal to the higher value of *prop.kwx* or 3;

c. if *prop.dist* is less than 1000 meters, or
prop.dist is greater than 2000 kilometers, then:
prop.kwx is set to be equal to 4;

Line 722:
```
if (prop.dist>0.0)
{
        if (prop.dist>1000e3)
                prop.kwx=mymax(prop.kwx,1);

        if (prop.dist<dmin)
                prop.kwx=mymax(prop.kwx,3);

        if (prop.dist<1e3 || prop.dist>2000e3)
                prop.kwx=4;
}
```

20. The third **if** statement, and its three embedded **if** statements, have run their course. A fourth primary **if** statement is initiated, stating that if prop.dist is less than prop.dlsa, then:

Line 734:
```
if (prop.dist<propa.dlsa)
{
```

<u>In steps 21 through 37, the coefficients for the Line-of-Sight Range, including the line-of-sight path loss, are calculated:</u>

21. The fourth primary **if** statement is follow by a series of embedded **if** and **else** statements; the first of these **if** statements states that: if (*wlos*) is a boolean false, indicating that the line-of-sight coefficients have not yet been calculated, then:

a. Subroutine ***alos*** is called with input (*0.0,prop,propa*). The subroutine ***alos*** returns *alosv*, the value of the line of sight attenuation, and *q* is set to be equal to *alosv*.

b. *d2* is set to be equal to *propa.dlsa*, the sum of the two smooth earth horizon distances;

c. *a2* is set to be equal to the sum of *propa.aed* and (*d2 * propa.emd*);

where

propa.aed	is defined in step 17, above	[Alg 4.8]
propa.emd	is defined in Step16, above	[Alg. 4.7]

d. d0 is set to be equal to: (1.908**prop.wn*prop.he[0]*prop.he[1]*;
[Alg. 4.28]
where

prop.wn is the wave number, = frequency /47.7 MHz*meters
prop.he[0] is the effective height of the transmit antenna
prop.he[1] is the effective height of the receive antenna

Line 736:
```
if (!wlos)
{
        q=alos(0.0,prop,propa);
        d2=propa.dlsa;
        a2=propa.aed+d2*propa.emd;
        d0=1.908*prop.wn*prop.he[0]*prop.he[1];
```

22. The first embedded **if** statement following the fourth primary **if** statement, states that if *propa.aed* is greater than, or equal to, zero, then:
    a. *d0* is set to be equal to the lesser of: *d0* or ½ of *propa.dla*; [Alg. 4.28] where *propa.dla* is the sum of the two terminal to horizon distances, and:
    b. d1 is set to be equal to: *d0*+0.25*(*propa.dla – d0*); [Alg. 4.29]

Line 743:
```
        if (propa.aed>=0.0)
        {
                d0=mymin(d0,0.5*propa.dla);
                d1=d0+0.25*(propa.dla-d0);
        }
```

23. An ***else*** statement follows, so if *propa.aed* is less than zero, then:
    d1 is set to be the greater of: [-propa.aed/propa.emd] or [0.25*propa.dla] [Alg. 4.39]

Line 749:
```
else
        d1=mymax(-propa.aed/propa.emd,0.25*propa.dla);
```

24. Subroutine ***alos*** is called with input (*d1*,*prop*,*propa*). [Alg. 4.31]

The subroutine ***alos*** returns *alosv*, the value of the line of sight attenuation, and *a1* is set to be equal to *alosv*.

Line 752:
```
a1=alos(d1,prop,propa);
```

25. *wq* is then set to be equal to Boolean false.

Line 753:
```
wq=false;
```

26. The second embedded **if** statement following the fourth primary **if** statement, states than if *d0* is less than *d1*, then:

a. Subroutine ***alos*** is called with input (*d0,prop,propa*). The subroutine ***alos*** returns *alosv,* the value of the line of sight attenuation, and *a0* is set to be equal to *alosv.* [Alg. 4.30]

b. q is set to be equal to: log(*d2*/*d0*)

c. propa.ak2 is set to be equal to:

((*d2-d0*)*(*a1-a0*)-(*d1-d0*)*(*a2-a0*))/((*d2-d0*)*log(*d1/d0*)-(*d1-d0*)**q*))

or zero, whichever is greater; [Note: there is an error in Alg. 4.32 here that leaves out the log function. Correct in the code.]

d. if *propa.aed*>=0.0 or *propa.ak*2>0.0
wq is set to be equal to Boolean true;

```
Line 755:  if (d0<d1)
                {
                    a0=alos(d0,prop,propa);
                    q=log(d2/d0);
                    propa.ak2=mymax(0.0,((d2-d0)*(a1-a0)-(d1-d0)*(a2-a0))/((d2-
                    d0)*log(d1/d0)-(d1-d0)*q));
                    wq=propa.aed>=0.0 || propa.ak2>0.0;
```

27. A second **if** statement is embedded within the **if** statement on line 736. If *wlos* is not a Boolean "true", and if *wq* is boolean "true", then: *propa.ak1* is set to be equal to: (*a2-a0-propa.ak2***q*)/(*d2-d0*). [Alg. 4.33]

```
Line 762:       if (wq)
                {
                    propa.ak1=(a2-a0-propa.ak2*q)/(d2-d0);
```

28. An **if** statement is embedded within the **if** statement on line 762. So:

a. If *wlos* is not a Boolean "true", and;
b. if *d0* is less than *d1*;
c. if *wq* is boolean "true", and;
d. if *propa.ak1* is less than zero;
e. then: (1.) *propa.ak1* is set to be equal to zero, and: [Alg. 4.36]
(2.) *propa.ak2* is set to be equal to (*a2 – a0)/*q if *a2* is greater than *a0;* if *a2* is not greater than *a0,* the FORTRAN_DIM function returns zero, and *propa.ak2* is set to be equal to 0.0/*q,* i.e. zero. [Alg. 4.35]

```
Line 766:           if  (propa.ak1<0.0)
                    {
                        propa.ak1=0.0;
```

```
propa.ak2=FORTRAN_DIM(a2,a0)/q;
```

29. An **if** statement is embedded within the **if** statement on line 766, So:
    a. if *wlos* is not a Boolean "true", and;
    b. if *d0* is less than *d1*;
    c. if *wq* is boolean "true", and
    d. if *propa.ak1* is less than zero, and
    e. if *propa.ak2* is equal to zero, then:
    f. *propa.ak1* is set to be equal to *propa.emd.* [Alg. 4.37]

```
Line 771:                   if (propa.ak2==0.0)
                                    propa.ak1=propa.emd;
                    }
          }
```

30. At this point, the **if** statements at Lines 771, 766, and 762 have completed their run. The **if** statements at Lines 755 and 736 are still active.

31. An ***else*** statement follows, providing an alternative path to the **if** statement on line 755. Therefore:
    a. if *wlos* is not a Boolean "true", and;
    b. if *d0* is equal to or greater than *d1*, then:
    c. *propa.ak2* is set to be equal to zero, and:
    d. *propa.ak1* is set to be equal to (*a2-a1*)/(*d2-d1*);

```
Line 776:       else
                    {
                            propa.ak2=0.0;
                            propa.ak1=(a2-a1)/(d2-d1);
```

32. An **if** statement is embedded within the **else** statement on line 766, so:
    a. if *wlos* is not a Boolean "true", and;
    b. if *d0* is equal to or greater than *d1*, and;
    c. if *propa.ak1* is less than or equal to zero,
    d. *propa.ak1* is set to be equal to *propa.emd*.

```
Line 781:                   if (propa.ak1<=0.0)
                              propa.ak1=propa.emd;
                    }
          }
```

33. The **else** statement from line 776 ends its run; the **if** statements on line 755 and 736 are still active. A new ***else*** statement follows on line 786, providing an alternative path to the **if** statement on line 755. So:

a. if *wlos* is not a Boolean "true", and;
b. if *d0* is equal to or greater than *d1*, and;
c. *propa.ak1* is set to be equal to: (*a2-a1*)/(*d2-d1*); [Alg. 4.41]
d. *propa.ak2* is set to be equal to 0.0; [Alg. 4.40]

Line 791:
```
else
{
        propa.ak1=(a2-a1)/(d2-d1);
        propa.ak2=0.0;
```

34. An **if** statement is embedded within the **else** statement on line 786, so:

a. if *wlos* is not a Boolean "true", and;
b. if *d0* is equal to or greater than *d1*, and;
c. if *propa.ak1* is less than or equal to zero,
d. *propa.ak1* is set to be equal to *propa.emd*.

Line 791:
```
        if (propa.ak1<=0.0)
                propa.ak1=propa.emd;
}
```

Note: In the ITM version 7.0 released June 26, 2007, the else and if statements in Steps 33 and 34 were removed, and the else statement in Step 33 becomes an if statement. The modification performs the same actions as the old code, in 10 fewer lines;

Alternate to Steps 31 to 34;

The **else** statement from line 776 ends its run; the **if** statements on line 755 and 736 are still active. A new ***if !****wq* statement follows, leading to a FORTRAN_DIM call; So:

e. if *wlos* is not a Boolean "true", and;
f. if *d0* is equal to or greater than *d1*, and;
g. *propa.ak1* is set to be equal to (*a2-a1*)/(*d2-d1*) if a2 is greater than a1; if a2 is equal to or less than a1, *propa.ak1* is set to be equal to zero. [Alg. 4.41]
h. *propa.ak2* is set to be equal to 0.0; [Alg. 4.40]

Alternate code:
```
if (! wq)
{
        propa.ak1=FORTRAN_DIM(a2,a1)/(d2-d1);
        propa.ak2=0.0;
```

An **if** statement is embedded within the **if (!** *wq***)** statement, so:

i. if *wlos* is not a Boolean "true", and;
j. if *d0* is equal to or greater than *d1*, and;
k. if *propa.ak1* is equal to zero,
l. *propa.ak1* is set to be equal to *propa.emd*.

Alternate Code: if (propa.ak1= =0.0) propa.ak1=propa.emd;
}

35. The **else** statement on line 786 has now completed its run. Here:

a. *propa.ael* is set to be equal to *a2 – propa.ak1 * d2 – propa.ak2 ** log(*d2*), and:
b. *wlos* is set to be equal to: Boolean" true", indicating completion of the calculation of the line-of-sight coefficients.

Line 795: propa.ael=a2-propa.ak1*d2-propa.ak2*log(d2);
wlos=true;
}

<u>The next step calculates the reference attenuation, *aref*, for the line of sight range.</u>

36. An **if** statement is initiated. The **if** statement on line 736 is still active, so:

a. if *wlos* was not a Boolean "true" when checked by the **if** statement in Step 21, and;
b. if *prop.dist* is greater than zero, then:
c. *prop.aref* is set to be equal to:

*propa.ael + propa.ak1 * prop.dist + propa.ak2 ** log(*prop.dist*)
[Alg. 4.1]

Line 799: if(prop.dist>0.0)
prop.aref=propa.ael+propa.ak1*prop.dist+propa.ak2*log(prop.dist);

37. The **if** statement on line 736 ends its run. We have finished calculating the coefficients for the Line of Sight range, and have calculated the value of aref if the path is line-of-sight from the transmit to the receive terminals.

Line 802: }

<u>In Steps 38 to 41, coefficients are calculated for the Troposcatter (scatter) range.</u>

38. The last primary **if** statement is initiated at line 804. It has an embedded **if** statement immediately following; so if:

a. *prop.dist*, the path distance, is less than or equal to zero, or:

b. *prop.dist*, is greater than *propa.dlsa*, the sum of the calculated distances to the smooth earth horizons. This is the point, for a smooth earth condition, where diffraction mode takes over from line of sight mode.
c. and;
d. if *wscat* is not Boolean *true* (i.e. is Boolean *false),* then:
    (1) subroutine ***ascat*** is called with inputs (0.0, *prop*,*propa*). Subroutine ***ascat*** returns *ascatv*, the value of the "scatter attenuation", and *q* is reset to be equal to *ascatv*
    (2) *d5* is set to be equal to *propa.dla* + 200,000 meters. [Alg. 4.52]
    (3) *d6* is set to be equal to *d5* + 200,000 meters, i.e. = *propa.dla* + 400,000 meters. [Alg. 4.53]
    (4) subroutine ***ascat*** is called with inputs (*d6*, *prop*,*propa*). Subroutine ***ascat*** returns *ascatv*, the value of the "scatter attenuation", and *a6* is reset to be equal to *ascatv* [Alg. 4.54]
    (5) subroutine ***ascat*** is called with inputs (*d5*, *prop*,*propa*). Subroutine ***ascat*** returns *ascatv*, the value of the "scatter attenuation", and *a5* is reset to be equal to *ascatv* [Alg. 4.55]

```
Line 804:  if  (prop.dist<=0.0 || prop.dist>=propa.dlsa)
            {
                if(!wscat)
                {
                        q=ascat(0.0,prop,propa);
                        d5=propa.dla+200e3;
                        d6=d5+200e3;
                        a6=ascat(d6,prop,propa);
                        a5=ascat(d5,prop,propa);
```

39. An **if** statement, embedded under the **if** statement at line 806, which is embedded under the primary **if** statement at line 804, is initiated. So if:
a. *prop.dist*, the path distance, is less than or equal to zero, or:
b. *prop.dist*, is greater than *propa.dlsa*, and:
c. if *wscat* is not Boolean *true* (i.e. is Boolean *false),*and;
d. if *a5* is less than 1000, then:
    (1) *propa.ems* is set to be equal to: (*a6*-*a5*)/(200000 meters) [Alg. 4.57]
    (2) *propa.dx*, the distance where diffraction mode gives way to scatter mode, is set to be equal to the greater of: [*propa.dlsa*] or [the greater of (*propa.dla* + 0.3 * *xae* * log(47.7 * *prop*.wn), or ((*a5*-*propa.aed*-*propa.ems**d5)/(*propa.emd*-*propa.ems*))]; [Alg. 4.58]
    (3) *propa.aes* is set to be equal to:
        (*propa.emd* -*propa.ems*) * *propa.dx* + *propa.aed* .
    [Alg. 4.59]

```
Line 814:       if (a5<1000.0)
                {
                        propa.ems=(a6-a5)/200e3;
                        propa.dx=mymax(propa.dlsa,mymax(propa.dla+0.3*xae*log(47.7*
                        prop.wn),(a5-propa.aed-propa.ems*d5)/(propa.emd-propa.ems)));
                        propa.aes=(propa.emd-propa.ems)*propa.dx+propa.aed;
                }
```

40. An **else** statement provides an alternate path to the **if** statement immediately above. So if:
    a. *prop.dist*, the path distance, is less than or equal to zero, or:
    b. *prop.dist*, is greater than *propa.dlsa*, and;
    c. if *wscat* is not Boolean *true* (i.e. is Boolean *false),*and;
    d. if *a5* is equal to or greater than 1000, then:
        (1) *propa.ems* is set to be equal to: *propa.emd.*
        (2) *propa.aes* is set to be equal to *propa.aed.*
        (3) *propa.dx* is set to be equal to: 10,000,000. [Alg. 4.56]

```
Line 821:               else
                        {
                                propa.ems=propa.emd;
                                propa.aes=propa.aed;
                                propa.dx=10.e6;
                        }
```

41. The value of *wscat* is then set to be equal to a Boolean "*true;*" The **if** statement at line 806 then ends its run.

```
Line 828:               wscat=true;  (Scatter coefficients calculated and ready)
                }
```

<u>The coefficients for the Troposcatter (scatter) range have now been calculated. In steps 42, 43, and 44, aref, the reference attenuation, will be computed as per [Alg. 4.1] if the path ends in the scatter (Step 43) or diffraction (Step 42) ranges.</u>

42. An **if** statement, embedded within the **if** statement at line 804, is initiated; so if:
    a. *prop.dist*, the path distance, is less than or equal to zero, or:
    b. *prop.dist*, is greater than *propa.dlsa*, and;
    c. if *prop.dist* is greater than *propa.dx*, indicating that we are in the troposcatter (scatter) range, then:
    d. *prop.aref* is set to be equal to: *propa.aes* + *propa.ems* * *prop.dist*;

```
Line 831:               if (prop.dist>propa.dx)
                        prop.aref=propa.aes+propa.ems*prop.dist;
```

43. An **else** statement provides an alternative path to the **if** statement directly above, which is still embedded within the the **if** statement in Step 38; so if:

a. *prop.dist*, the path distance, is less than or equal to zero, or:
b. *prop.dist*, is greater than *propa.dlsa*, indicating we are past the line-of-sight range for a smooth earth situation, and;
c. if *prop.dist* is equal to or less than *propa.dx*, indicating that we have not yet arrived at the distance where diffraction dominance rolls over to (tropo)scatter dominance, (the combination of b. and c. therefore indicating that we are in the diffraction dominant area) then:
d. *prop.aref* is set to be equal to: *propa.aed* + *propa.emd* * *prop.dist*;

```
Line 833:       else
                        prop.aref=propa.aed+propa.emd*prop.dist;
        }
```

Here we have a problem; an omission. The else statement in Step 38 only allows the else statement on Line 833 to act to switch the mode from line-of-sight to diffraction mode for a smooth earth situation, at the point where the path length exceeds the sum of the calculated smooth earth horizon distances. There is no provision to switch line-of-sight mode to diffraction mode, at the point when an obstruction blocks the horizon for the transmitter site. Therefore, the string mode propagation status printout from point_to_point, lies in outputting "Diffraction Mode" after passing the first obstacle, as the line-of-site mode actually continues until the path length exceeds the sum of the smooth earth horizon distances.

To make this subroutine operate to match the status printout from point_to_point, and correctly switch from line of sight mode calculation of aref to diffraction mode calculation of aref at the first obstruction, it is only necessary to add the following if statements, just after the last bracket for the if statement that starts on line 804:

```
        if (prop.dist<propa.dx);
                {
                if (prop.dist>propa.dla);
                        prop.aref=propa.aed+propa.emd*prop.dist;
                }
```

44. The subroutine ***lrprop*** then sets *prop.aref*, the reference attenuation, to be equal to the greater of *prop.aref* or zero, and then returns the value of *prop.aref*..

```
Line 837:       prop.aref=mymax(prop.aref,0.0);
}
```

Chapter XX: Lrprop2

Longley-Rice Propagation subroutine ***lrprop2***.

Last Modified Sept 21, 2008 – itwom1.0t.cpp.

Note: Used with both point-to-point and area modes. For point-to-point mode, called at end of ***qlrpfl.*** Calls subroutines ***adiff, alos2, ascat, mymin,*** and ***mymax***.

From ITMD Sections 4, 5 to 9, 15; 16, and 17 with 18 and 19, and 20 with 21, 22 and 23:

The Longley-Rice propagation program. This is the basic program; it returns the reference attenuation *aref.* This is the replacement for the "PaulM" version of lrprop2 found in the ITM version 7.0, released on June 26, 2007.

Many of the earliest found descriptions of the original equations in this subroutine are in ESSA Technical Report ERL 79 – ITS 67, "Prediction of Tropospheric Radio Transmission Over Irregular Terrain, A Computer Method – 1968" (ITS67) by A. G. Longley and P. L. Rice.

A Quick Summary of the (original) Lrprop Attenuation Calculation Function:

Therefore, the calculation of radio path loss starts out in the line of sight range, using 2-ray multipath calculations, added to diffraction calculations projected from the results of diffraction calculations in the diffraction range. The calculation fades to a diffraction calculation beyond the transmitter horizon; past distance dlsa, the sum of the theoretical smooth earth horizons, to distance dx, only diffraction mode calculation is used. Beyond the distance dx, troposcatter, or scatter, computations are used to determine the losses.

Why was it necessary to write ***lrprop2*** to replace ***lrprop***?

The original version of the ITM is designed to use either a very granular terrain database in the point-to-point mode, or no terrain database in the area mode. For no terrain database, or terrain databases that are granular enough to require interpolation of ground height between data points in the line-of-sight to early diffraction range, the subroutine ***lrprop*** does not trust a calculation made at just one point. It also has to show a smooth transition from the line of sight range to the diffraction range, where the line of sight ends and diffraction over the curve of the earth takes over. The surface where the signal grazes the curve of the earth at the horizon can be smooth, or it can be slightly "irregular terrain"; the calculation is designed to handle either situation. If the earth is smooth, the "terrain irregularity factor" is low, and the calculations simplify to a smooth earth calculation. As the terrain irregularity increases, the effect of the two-ray interfering signal is increasingly muted as the interfering signal is scattered by the terrain. Therefore, for the line-of-sight-to-just-past-the-horizon range, ***lrprop*** picked out five points along

the path, d0 through d4, and fits three each of these points to two geometric path loss curves of the form y = a + bx + c*ln(x), one for the line-of-sight range, and one for the diffraction range. The reference attenuation is then calculated by solving for a point on either the line-of-sight curve or the diffraction curve.

The original procedure, or "building plans" followed by another , earlier "contractor" is described in detail in ITS 67, starting in section 3.1, page 16, describing the line-of-sight curve creation:

> "These three values of attenuation, A_o, A_1 and A_{Ls}, computed at the distances d_o, d_1, and d_{Ls}, respectively, are used to determine the slopes k_1 and k_2 of a smooth curve of A_{cr} versus distance for the range $1 \leq d \leq d_{Ls}$."

This concept is expanded and modified in practice; The procedure used by the original subroutine ***lrprop***, which was assembled from the instructions found in several subroutines in ITS 67, is described in detail in ITS 67, starting in section 3.1, page 16, describing the line-of-sight curve creation:
in ITS67, Annex 3, Subsection 3-1, we find that:

> " The two-ray optics formulas (3.2) to (3.15) are used to compute values of attenuation A_{ot} and A_{1t} at distances d_o and d_1, respectively."
>
> "In addition to the [Ed.; line-of-sight] two-ray theory estimates A_{ot} and A_{1t} of attenuation at the distances d_o and d_1, estimates of diffraction attenuation A_{od}, A_{1d}, and A_{Ls} are also computed at d_o, d_1, and d_{Ls}."
>
> "The estimates of attenuation A_o and $A_{1,}$ at the distances d_o and d_1, are then computed as weighted averages of the two-ray theory and the diffraction estimates." [Ed. this, and the previous paragraph's actions occurs in calls to subroutine ***alos***.]
>
> "For distances less than the smooth-earth horizon distance d_{Ls}, the calculated reference value A_{cr} is defined by a smooth curve fitted to the three values of attenuation below free space, A_o, A_1, and A_{Ls}, at the distances d_o, d_1, and d_{Ls}."
>
> For $0 < d < d_{Ls}$;
>
> $$A_{cr} = A_o + k_1(d - d_o) + k_2\log_{10}(d/d_o) \quad \text{dB} \qquad (3.19)$$
>
> The constants k_1 and k_2 in (3.19) are evaluated as follows. First estimates $\hat{k}_1$, $\hat{k}_2$ of the slopes k_1 and k_2 in (3.19) are computed as:
>
> $$\hat{k}_2 = ((A_{Ls} - A_o)(d_1 - d_o) - (A_1 - A_o)(d_{Ls} - d_o))/ ((d_1 - d_o)\log_{10}(d_{Ls}/d_o) - (d_{Ls} - d_o)\log_{10}(d/d_o) \quad \text{dB},$$

or 0, whichever is larger algebraically, (3.20)

$$\hat{k}_1 = [(A_{Ls} - A_o) - \hat{k}_2 \log_{10}(d_{Ls}/d_o)]/(d_{Ls} - d_o) \text{ dB/km}, \quad (3.21)$$

if $\hat{k}_1 < 0$ set $k_1 = 0$ and

$$k_2 = (A_{Ls} - A_o)/\log_{10}(d_{Ls}/d_o). \quad (3.22)$$

If the reference attenuation A_{cr} computed from (3.19) is less than zero at any distance $0 < d < d_{Ls}$, let $A_{cr} = 0$ for that distance."

The abbreviation A_{cr}, for the reference attenuation, is later replaced by aref and the value is held by propa.aref. The formulas are also changed to derive, and later function, using natural logarithms, ln, or base e, where in the original version from ITS-67 above, common logarithms, a.k.a. base 10 logarithms, or $\log_{10}$, functions are used. Please be aware that the use of log, without the 10 subscript, may refer to either ln, the old and computer coding convention, or $\log_{10}$, the current common usage and calculator convention, and must be considered usage with an indeterminate base until verified, despite the statement in TN101 to the effect that all use of log refers to base 10 logarithms.

Answer: In ITS67, on page 15, there is a paragraph that starts with "The reference attenuation". It is a overview of the summary of the procedure followed by the program code. There is no mention of the word "obstruction". The concept used to design the original code was to present calculations that could account for "irregular terrain" at the horizon, as compared to a "smooth earth" scenario. No provision was made in the computations to account for the effect of interrupting the radio path with a significant, major obstruction.

No subroutine LRPROP exists in the FORTRAN code found in Annex 3-5 to ITS67. The original code that implements these procedures is found in unfamiliar, obsolete subroutines such as DIFF, SCATT, and LOS. When the functions of these subroutines were combined into LRPROP, however, it appears that they were combined hurriedly, without proper review, and without an understanding of the fact that the concept of Irregular Terrain had changed from "a little Irregular Terrain found at the horizon" to mean "all kinds of Irregular Terrain, including Obstructions." As a result, while the subroutines (some of which are imperfect early versions) necessary to support obstruction calculations are in place, no provision was made in *lrprop* to accommodate the changes required in the computational procedure that must occur when the line-of-sight and diffraction ranges no longer gently merge, but are, instead, abruptly separated by a major obstruction. Therefore, subroutine ***lrprop*** *was never completed, despite the fact that a slightly upgraded version, as part of ITM version 7.0, was released to the public as late as June 26, 2007.*

This book is not the first time these problems have been noted, but the previous reference does not exhibit a realization of the true reason for the problem, nor does it offer a

solution. In the book "Fixed Broadband Wireless System Design", by Harry R. Anderson, Ph.D., P.E., ©2003 John Wiley & Sons, Ltd., in the chapter on Physical Models, subsection 3.5.3, "Longley-Rice model", Dr. Anderson, the President and CEO of EDX Wireless, LLC, correctly summarizes the path loss slope line methodology utilized in this subroutine, and notes that:

> "At distances from the transmitter to the horizon, the path loss actually includes a weighted portion of the diffraction loss *beyond* the horizon. Having the path loss to a receiver location be affected by terrain obstacles beyond that receiver location is clearly noncausal and violates physical reasoning for a single path two-dimensional model."

The Quick Summary of the new Lrprop2 Attenuation Calculation Function:

The calculation of radio path loss starts out in the line of sight range, using 2-ray multipath attenuation calculations, the results of which are added to average ground clutter attenuation calculations to predict the line-of-sight attenuation to be added to free space dispersion. At the peak of the highest obstruction visible from the transmitter, if any, or distance dlsa, the sum of the theoretical smooth earth horizons, a diffraction-only calculation determination takes over. Beyond the distance dx, troposcatter, or scatter, computations are used to determine the losses.

Changes from ***lrprop*** to ***lrprop2***:

This subroutine ***lrprop2*** is a major rewrite of ***lrprop***, a low level supervisory, or "foreman" subroutine. The specialized attenuation calculations subroutines called by ***lrprop2*** are the "skilled worker" subroutines ***adiff, alos2,*** (which calls ***saalos***), and ***ascat***. While subroutine ***alos2*** is a major rewrite of ***alos***; ***saalos*** is an entirely new subroutine.

The subroutine ***alos2*** calculates the additional attenuation for line-of-sight paths that comes from two-ray multipath cancellation, the out-of-phase cancellation caused by the primary signal reflecting off of the ground and combining with the direct line-of-sight signal. The two-ray effect quickly fades away as increasing terrain irregularity, and two passes through the absorbing ground clutter layer, reduce the amount of ground-reflected signal arriving at the reception point. Subroutine ***alos2*** then calls ***saalos*** to determine the attenuation due to ground clutter, summing it with the multipath attenuation and reporting the total as the line-of-sight attenuation. Subroutine ***adiff*** calculates the diffraction attenuation in the diffraction range. Subroutine ***ascat*** calculates the diffraction attenuation in the troposcatter range.

The result-averaging line-of–sight calculation methodology used for area mode or widely-spaced terrain databases, is modified to use data points that are before the horizon or first obstacle. For databases with 3 arc-second or less spacing, the calculation methodology in the line of sight and diffraction range is simplified to a single calculation

per point method that is appropriate when one-calculation-per-pixel methodology is used in the wrap-around input-output software.

For a line-of-sight calculation, the lrprop2 subroutine, working with the alos subroutine, now calculates, in a single pass, a full consideration of the non-contiguous and non-cumulative wavefront-wavelet primary and multipath (formerly 2-ray) signal combination effects, and adds in consideration of average clutter attenuation, using Radiative Transfer Theory methodology, in the line of sight range. The line-of-sight range now also rolls over to the diffraction range at the peak of the tallest obstruction visible from the transmitter site along the rf path, if any exist; an obstruction being defined as a elevation point in the rf path that rises above an imaginary line between the transmitter antenna center and the horizon, as identified in subroutine ***hzns***.

A Detailed Overview:

The ***lrprop2*** subroutine calculates up to seven distances, d0 to d6, here starting with d2 to d6. Distance d2 is set to be the distance from the transmitter site to the terrain data point representing the top of the highest obstacle visible from the transmitter, the start of an obstructed diffraction range. If no such obstacles exist in the path, d2 is set to equal distance *dlsa*, the sum of the two calculated smooth-earth horizon distances, used as the maximum distance considered for a line-of-sight range.

Distance d3 is set to be equal to the minimum of: (1) the maximum range of free space calculation, distance *dlsa*, the sum of the estimated smooth earth horizon distances, or (2) 1.3787*xae+dla, whichever is greater. For an obstructed path, dla is the sum of the actual horizon distances. For a line-of-sight path, dla is a calculated theoretical estimate. Distance d3 is therefore at or past distance dlsa. Distance d4 is well inside of the diffraction range, between distances *dlsa* and *dx*. Distances d5 and d6 are beyond distance *dx*, in the troposcatter range. Distance d6 is the farthest from the transmitter. The distance *dlsa* represents the maximum distance where the line-of-sight calculations can be used; beyond this point, the diffraction calculations are used until the distance *dx* is reached.

If the radial interval length *xi*, stored in pfl[1], is greater than 500 meters, it indicates that a terrain database with data intervals greater than 3 arc-seconds is in use. When *xi* > 500 meters, subroutine ***lrprop2*** operates with a corrected and updated version of the result-averaging system originally employed in the legacy ITM 1.2.2 to 7.0 version computer implementations.

Revised Calculations for a Coarse Terrain Database or No Terrain Database:

If *xi* > 500, subroutine ***lrprop2***

In the section on setting line-of-sight coefficients, the old ***lrprop*** subroutine then set distance d2 = distance dlsa, and calculated a2, the diffraction attenuation at distance d2 = dlsa, using the diffraction loss formula calculated in the paragraph above. In ***lrprop2***, distance d2 is set to be at the top of the highest obstacle visible from the transmitter, the start of an obstructed diffraction range. If no obstacles exist in the path, d2 is set to equal distance *dlsa*, the sum of the two calculated smooth-earth horizon distances, used as the maximum distance considered for a line-of-sight range.

Subroutine ***lrprop2*** then calls subroutine ***adiff*** to calculate the diffraction loss, a2, at distance d2, and the diffraction loss, d3, at distance d3. Subroutine ***adiff*** performs a diffraction calculation based on a combined estimate of Fresnel-Kirchhoff knife-edge diffraction theory and Vogler smooth bulge theory, described in section 3.2 of the ITS67. Subroutine ***lrprop2*** then calculates a straight-line diffraction loss formula, of the form *a*(#) = *aed* + *emd**(path location distance *d(#)*).

If d is less than the actual transmitter horizon, dl[0], the distances d0 and d1 are then calculated. Subroutine ***lrprop2*** then calls subroutine ***alos2*** three times, to calculate the line of sight attenuation a0, a1, and a2a, at locations d = d0, d1, and d2.

Subroutine ***alos2*** calculates the line-of-sight attenuation, a0, at distance d0, using a two-ray multipath calculation, and then adds the Radiative Transfer Engine-calculated clutter loss obtained from a call to subroutine ***saalos***. The sum of the multipath and clutter attenuation, the line-of-sight attenuation to be added to free space dispersion, is stored as attenuation a0. On the second call, it does the same at distance d1. On the third run, Subroutine ***lrprop*** compares the output of ***alos2*** to the existing diffraction-calculated value of a2; the lowest value of the two values is stored as the attenuation value for a2. Over smooth, bare earth, the diffraction mode may have taken over at distance d2 as the path of least attenuation of the signal from the transmitter to the receiver; over cluttered ground, the combination of Snell's law geometry and Radiative Transfer Engine (RTE) scatter mode I3 (See Shumate's Approximations) can extend the line-of-sight mode up to .02 radians (1/2 degree) past the point where the grazing angle becomes horizontal.

We now have attenuation values, a0, a1, and a2, calculated for the distances d0, d1, d2, from near the transmitter site (d0) to the maximum distance for line-of-sight calculations, dlsa (d2). These values are used to generate a new, master line-of-sight attenuation formula, a log-geometric curve formula of the form:

$$A_{cr} = a0 + k_1(d - d0) + k_2\log_{10}(d/d0) \qquad \text{[ITS-67 3.19]}$$

The above curve equation is used in ITS-67; in the ITM 1.2.2, the curve equation is modified, and the logarithm function changes its base. The documentation is consistent about this; the ITMDLL.cpp source code, the Algorithm, and the FORTRAN source code in the Guide agree that the logarithmic function changes from a common logarithm, or $\log_{10}$ function, in the ITS-67 source code, to a natural logarithm, or ln function ("ALOG" in the FORTRAN and "log" in the c++ code) in the curve equation used in the ITM 1.2.2. to 7.0. This new curve equation is:

$$A_{ref} = a2 - k_1*d2 - k_2*\ln(d2) + + k_1*d + k_2*\ln(d/d_{Ls}),$$

Which is split into:

$$A_{ref} = A_{el} + k_1*d + k_2*\ln(d/d_{Ls}) \qquad \text{[Alg. 4.1a]}$$

where:

$$A_{el} = a2 - k_1*d2 - k_2*\ln(d2) \qquad \text{[Alg. 4.42 with the omission corrected]}$$

Where A_{ref} is the line-of-sight attenuation to be summed with free space loss attenuation to obtain the full attenuation value in the line of sight range from the transmitter out to distance dlsa (the sum of the estimated smooth earth horizon distances), or the horizon, or the tallest obstruction visible from the transmitter site, whichever comes first. This value, which will be reported out as the reference attenuation if *prop.dist*, the distance to the receive terminal, is more than zero and less than or equal to *dl[0]*, is then stored in *prop.aref*.

To cut through the massive mathematical obfuscation, for any Δh greater than 4 meters (90 meters is average), the 2-ray calculation attenuation fades quickly out of the picture. For most cases, from mildly irregular terrain through average terrain and on to very rough terrain, and for paths covered with ground clutter, most of the line-of-sight attenuation to be added to free space attenuation is now based on Radiative Transfer Engine theory, in the form of Shumate's Approximations.

The surface where the signal grazes the curve of the earth at the horizon can be smooth, or it can be slightly "irregular terrain"; the calculation is designed to handle either situation. If the earth is smooth, the "terrain irregularity factor" is low, and the calculations simplify to a smooth earth calculation. As the terrain irregularity increases, the effect of the two-ray interfering signal is increasingly muted as the interfering signal is scattered by the terrain.

For the diffraction range for area mode or terrain databases with more than 3-arc second spacing, ***lrprop2*** calls subroutine ***adiff*** to calculate the diffraction loss at an additional points along the path, d4, and fits attenuation values a2, a3 and a4 to a geometric path loss curve of the form y = a + bx + c*ln(x), for the diffraction range. If the path distance is equal to or greater than dl[0], and less than dx, the reference attenuation is then calculated by solving for a point on this diffraction curve.

New Calculations for a Terrain Database with 3-arc second or smaller pixel size.

If the radial interval length *xi*, stored in pfl[1], is equal to or less than 500 meters, it indicates that a terrain database with a maximum of 3 arc-second terrain data intervals is in use. When *xi* < 500 meters, subroutine ***lrprop2*** assumes that the wrap-around software is operating in a one-calculation per terrain data point mode, i.e. a point-to-all-points mode, eliminating the need for interpolation of height between terrain data points. Subroutine ***lrprop2*** changes its methodology accordingly, abandoning the line-of-sight

and diffraction range averaging, and uses a one-calculation per terrain data point methodology in the line-of-sight and diffraction ranges to speed calculation and improve the accuracy of the prediction at each specified terrain data point.

Therefore, if $xi \leq 500$, and if d < d2, the minimum of: (1) distance dlsa, or (2) the distance to the top of the highest obstacle visible from the transmitter, then ***lrprop*** calls ***alos2*** once, to calculate the line of sight attenuation at the terrain database point represented by distance d. Subroutine ***alos*** calculates the multipath cancellation, and calls subroutine ***saalos*** to determine the attenuation due to ground clutter, summing the two values and reporting out the "attenuation from line-of-sight value" *alosv*. The array value *prop.aref* is set to be equal to *alosv*.

If $xi < 500$, and if the path distance d $\geq$ distance d2, but less than dx, ***lrprop*** calls subroutine ***adiff*** once, to calculate the diffraction at the terrain database point represented by distance d, and the array value *prop.aref* is set to be equal to *adiffv*.

A_{ref} , or *prop.aref*, is the line-of-sight attenuation to be summed with free space loss attenuation to obtain the full attenuation value in the line of sight range.

To cut through the mathematical obfuscation, for any Δh greater than 4 meters (90 meters is average), the 2-ray calculation attenuation fades quickly out of the picture. For most cases, from mildly irregular terrain through average terrain and on to very rough terrain, most of the line-of-sight attenuation to be added to free space attenuation is now based on the Radiative Transfer Engine calculations from Shumate's Approximations, a deterministic and deterministic-based set of approximation equations derived from a study of ITU Recommendation ITU-R P.1546-2 line-of sight empirical data curves, developed for use in the ITWOM Longley-Rice computer implementation.

Beyond the first obstruction or the horizon, up to distance dx, diffraction mode takes over, and the diffraction attenuation is determined using a single call to ***adiff***, which performs a two-knife-edge and rounded top diffraction calculation.

Beyond distance dx; the tropopscatter mode range.

Beyond distance *dx*, troposcatter mode determines the attenuation. The coefficients are then calculated for the Troposcatter (scatter) range. The distances *d5* and *d6* are calculated; *d5* = sum of the horizon distances (*dla*, not *dlsa*) plus 200 km. Distance *d6* is set to be 200 km beyond *d5*. Subroutine ***ascat*** is called twice to calculate the attenuation due to scatter at distances *a6* and *a5*. The distance *dx* is calculated. The slope, *ems*, for a straight-line scatter attenuation formula, of the form *a*(#) = *aes* + *ems**(path location distance *d(#)*), is derived from the results *a5* and *a6*. The intercept, *aes*, is derived from the difference in value between the diffraction and scatter slopes, (*emd* – *ems*), multiplied by the sum of distance *dx* and the diffraction intercept value *aed*.

In the final steps, if the path distance, *prop.dist*, is greater than *dx*, i.e. in the troposcatter range, then the reference attenuation value, *prop.aref*, to be added to free space attenuation in the ***point_to_point*** subroutine, is recalculated using the straight-line scatter attenuation formula, set to path distance d, stored in *prop.dist*. If the path distance is between dl[0] and *dx*, *prop.aref* is taken from the diffraction range calculation at distance *prop.dist*. If the path distance is above zero, and less than or equal to dl[0], *prop.aref* remains at the value set by the line-of-sight calculation above. The value of *prop.aref* is then set to be equal to the greater of *prop.aref* or zero, and the subroutine ends.

This is what happens in subroutine ***lrprop2***, where the work actually gets done. The mode of operation status information reported out by the ***point_to_point_two*** subroutine now matches up with what is actually happening in ***lrprop2***, which is now operating on the same instruction set built into "***point_to_point***", the ITM "middle manager" subroutine added prior to April, 1982.

Call inputs:

d — path distance set to 0.0 to set coefficients. (prop.dist carries the actual active path distance.)

&pfl — array *pfl* with terrain data coefficients and heights array

&prop — array *prop* with array elements:

&propa — array *propa* with array elements:

defines private, or local, arguments:

wlos — static boolean argument; true if line-of-sight coefficients have been calculated.

wscat — static boolean argument; true if troposcatter coefficients have been calculated.

dmin — static double argument; minimum acceptable path distance length in meters

xae — static double argument; a calculated distance term, with units in meters, based on the cube root of: a rf-signal illuminated square area of the earth's surface, with each side's length defined as the effective radius of the earth, with this area multiplied by the wave number (i.e. $2\pi/\lambda$, or the rotational frequency) of the signal. Used to determine distances d3 and d4, at which diffraction is calculated, et al; derived from [ITM67 3.24].

prop_zgnd — sub array zgnd (average ground impedance) with elements:

- prop.zgndreal — resistance element of ground impedance
- prop.zgndimag — reactive element of ground impedance

a0 — line-of-sight attenuation at distance d0

a1 line-of-sight attenuation at distance d1
a2 diffraction, or line-of-sight attenuation, whichever is less, at the peak of the highest obstruction visible from the transmitter site; or at distance dlsa if no obstruction exists in the path.
a3 diffraction attenuation at distance d3
a4 diffraction attenuation at distance d4
a5 troposcatter attenuation at distance d5
a6 troposcatter attenuation at distance d6

d0 1 of 2 distances at which line-of-sight attenuation is calculated
d1 2nd of 2 distances at which line-of-sight attenuation is calculated
d2 distance to peak of the highest obstruction visible from the transmitter site; or at distance dlsa if no obstruction exists in the path.
d3 2nd of 3 distances at which diffraction attenuation is calculated
d4 3rd of 3 distances at which diffraction attenuation is calculated
d5 1 of 2 distances at which troposcatter attenuation is calculated
d6 2nd of 2 distances at which troposcatter attenuation is calculated
pd1 the active path distance in meters, set to equal prop.dist,
wq Boolean argument; indicates whether general case 1 or general case 2 applies in calculating line of sight coefficients.
q working variable; holds various values during several operations
j either 0 (1 in Fortran), for transmit terminal, or 1 (2 in Fortran) for receive terminal
iw terrain interval width in meters, from prop.tiw; calculated in ***hzns***.

This subroutine:

Uses *d*=0.0, to set up coefficients, and information in arrays *prop* and *propa*, in order to calculate aref, the reference attenuation (radio signal strength loss) along the path between a transmit site and a receive location.

1. *iw*, the terrain interval width in meters, is set to be equal to *prop.tiw;* pd1, the active path location, is set to equal *prop.dist*, and dx is preset to 500,000 meters until the path distance exceeds dlsa, where the troposcatter coefficients computation section resets it to the working value. The argument dx, held in propa.dx, is preset to 2000 km until the path distance exceeds distance *dlsa*.

Line (new): iw=prop.tiw;
pd1=prop.dist;
dx=2000000.0;

2. An **if** statement is initiated. If prop.mdp, the mode of the propagation model, is not equal to zero, (zero would have indicated that the area prediction mode has initiated and is continuing), then the mode of the propagation model is either 1, which would initialize the area prediction mode, or –1, which indicates the

program is running in the point to point mode. If prop.mdp = 0, the program proceeds to the **for** loop on line 677. If prop.mdp is not zero, then:

Line 675: if (prop.mdp!=0)

3. A **for** statement is initiated with two loops, j=0 and j=1.

a. The first loop sets dls[0], the distance from the transmitter site to the smooth earth horizon, to be equal to the square root of 2 times prop.he[0] divided by prop.gme.

The algorithm formula comes from: "ESSA Technical Report ERL 79-ITS 67, Prediction of Tropospheric Radio Transmission Over Irregular Terrain, A Computer Method – 1968" by A.G.Longley and P.L.Rice, where it states on page 12:

"When individual path profiles are not available, median values of the horizon distances $d_{L1,2}$ are estimated as functions of the median effective antenna heights h_{e1} and h_{e2} determined above, the terrain irregularity factor Δh, and the smooth-earth horizon distances D_{Ls1} and D_{Ls2}. The smooth earth distance from each antenna to its horizon over a <u>smooth</u> <u>earth</u> is defined as:

$$D_{Ls1,2} = (.002 * a * h_{e1,2})^{.5} \text{ in km.} \qquad \text{[ITS67 (5a)]}$$

where the effective antenna heights $h_{e1,2}$ are in meters and the effective earth's radius a is in kilometers, as defined by (1)."

NOTE: Ms. Longley and Mr. Rice did not intend this formula to be utilized in the point-to-point mode, where individual path profiles are available. In the area mode, an individual path profile is not available. In point_to_point mode, it is still used to establish a point beyond which the possibility of line-of-sight mode has ceased, and only diffraction or troposcatter modes will be allowed.

Converting ITS67 (5a) for meters instead of km, we obtain:

$$D_{Ls1,2} = (2 * a * h_{e1,2})^{.5} \text{ in meters.}$$

As derived in the chapter on subroutine ***qlrpfl***, in step 11:

$$a = 1/ gme.$$

Where: *a* is the earth's effective radius, in meters, and
g*me* is the earth's effective curvature, in units of 1/meters. gme's value will nominally be calculated to be in the neighborhood of 0.00000012/meters.

The formula then becomes:

$$D_{Ls1,\,2} = (2 * h_{e1,\,2}/gme)^{.5} \text{ in meters.} \qquad \text{[Alg. 3.5]}$$

And in the computer, he[0] is h_{e1}, *gme* is stored in array *prop* at *prop.gme*; and the result, $D_{Ls1,}$ is stored in array *propa* at *propa.dls[0]*.

b. Similarly, the second loop sets dls[1], the distance from the receive site to the smooth earth horizon, equal to the square root of 2 times prop.he[1] divided by prop.gme.

Line 677:

```
for (j=0; j<2; j++)
        propa.dls[j]= sqrt(2.0*prop.he[j]/prop.gme);
```

4. The program then proceeds to:

   a. set propa.dlsa, the sum of the smooth-earth horizon distance, equal to the sum of propa.dls[0] and propa.dls[1], which were calculated in step 2 above, based on:

      The sum of the smooth-earth horizon distance is

$$D_{Ls} = D_{Ls1} + D_{Ls2} \qquad \text{[ITS67 (5b) or Alg. 3.6]}$$

   b. set propa.dla, the total distance between the antennas and their horizons, equal to the sum of propa.dl[0] and propa.dl[1], based on:

      The total distance, $d_{L,}$ between the antennas and their horizons is

$$d_L = d_{L1} + d_{L2} \qquad \text{[ITS67 (5d) or Alg. 3.7]}$$

The distances d_{L1} and $d_{L2,}$ the horizon distances, the value of which is stored in prop.dl[0] and prop.dl[1], are actual distances to the tallest visible obstruction from the transmitter (dl[0]) or receiver (dl[1]), or the estimated actual horizon. These distances have been determined as actual distances to obstruction peaks by subroutine ***hzns*** when it is called by subroutine ***qlrpfl***. For unobstructed line-of-sight paths, the horizon distances are estimated in subroutine ***qlrpfl*** by shortening the estimated smooth-earth horizon distances using an exponential factor based on the ratio of the terrain irregularity factor Δh, and the transmitter effective height.

   c. sets propa.tha to be equal to the greater of either (1) the sum of the theta angles stored in prop.the[0] and prop.the[1], or (2) the result of multiplying (– propa.dla (calculated in step 3(b)), times prop.gme, the effective earth's curvature). The effect on the subroutine is that propa.tha is set to be equal to the sum of the theta angles, which are generally positive (looking above the horizon) for an obstruction or multiple

obstructions, but for smooth-earth horizons, or small obstructions far away, one or both of the theta angles can be negative (looking down); and if the sum of the two angles is negative, the negative value is limited to propa.dla times prop.gme, the effective curvature of the earth. The argument tha can therefore be positive or negative.

NOTE: An attempted computation of infinity "inf" discontinuity in the adiff subroutine computation of *a* can occur when tha is equal to –d*propa.gme, causing th=0.0, and a = ds/th to report "inf" by attempting to divide by zero. This condition occurs where the sum of the theta angles represents a straight line from the transmitter to the receiver that just grazes the effective earth curvature.

tha= greater of: (the1+the1) or (-dla*gme) [Alg. 3.8]

d. sets the Boolean value of *wlos* and *wscat* to be false, as per instruction in ITMD Section 6:

Line 680:
```
propa.dlsa=propa.dls[0]+propa.dls[1];
propa.dla=prop.dl[0]+prop.dl[1];
propa.tha=mymax(prop.the[0]+prop.the[1],-propa.dla*prop.gme);
wlos=false;
wscat=false;
```

<u>In Steps 4 through 9, the program checks the parameter ranges of the input values, as per instructions in ITMD Section 7.</u>

5. An **if** statement is initiated to check if the frequency is within range; the wave number, prop.wn, which is derived from the frequency in step 2 of subroutine ***qlrps***, is checked to see if it is less than .838 (equivalent to a frequency of 40 MHz) or greater than 210 (equivalent to a frequency of 10 GHz). If prop.wn is outside of the range, prop.kwx, the error marker, is set to equal the greater of: the existing value of prop.kwx; or 1.

Line 686:
```
if (prop.wn<0.838 || prop.wn>210.0)
prop.kwx=mymax(prop.kwx,1);
```

6. A **for** statement is initiated with two loops, j=0 and j=1.

a. An **if** statement is initiated to check if hg[0], the transmitter antenna height above ground level, is within range; if hg[0] is less than one meter or greater than one kilometer, prop.kwx, the error marker, is set to equal the greater of: the existing value of prop.kwx; or 1. A value of kwx = 0 indicates: no warning; kwx = 1 indicates: caution; parameters are close to limits.

b. An **if** statement is initiated to check if hg[1], the receiver antenna height above ground level, is within range; if hg[1] is less than one meter or greater than one kilometer, prop.kwx, the error marker, is set to equal the greater of: the existing value of prop.kwx; or 1.

Line 689:
```
for (j=0; j<2; j++)
if (prop.hg[j]<1.0 || prop.hg[j]>1000.0)
prop.kwx=mymax(prop.kwx,1);
```

7. A **for** statement is initiated with two loops, j=0 and j=1.

   a. A three-way **if** statement is initiated to check if the[0], the transmitter antenna take off angle theta, is within range; if either:
      (1) the absolute value of the[0] is greater than 0.2,
      (2) prop.dl[0], the distance from transmitter to horizon,, is < less than 1/10 of propa.dls[0], smooth earth distance from transmitter to horizon,
      (3) prop.dl[0] is > 3.0 * propa.dls[0]

   Then prop.kwx, the error marker, is set to equal the greater of: the existing value of prop.kwx; or 3, a value of kwx = 3 indicating that internal calculations show parameters out of range.

   b. A three-way **if** statement is initiated to check if the[1], the receiver antenna take off angle theta, is within range; if either:
      (1) the absolute value of the[1] is greater than 0.2,
      (2) prop.dl[1], the distance from transmitter to horizon,, is < less than 1/10 of propa.dls[1], the smooth earth distance from transmitter to horizon,
      (3) prop.dl[1] is > 3.0 * propa.dls[1]

   Then prop.kwx, the error marker, is set to equal the greater of: the existing value of prop.kwx; or 3.

Line 693:
```
for (j=0; j<2; j++)
if (abs(prop.the[j]) >200e-3 || prop.dl[j]<0.1*propa.dls[j] ||
prop.dl[j]>3.0*propa.dls[j] )
prop.kwx=mymax(prop.kwx,3);
```

8. A seven-way **if** statement is initiated to check the ranges of *ens*, *gme*, *zgnd*, and *wn*. If either:
   a. *prop.ens*, the surface refractivity of the atmosphere, is less than 250.0 or greater than 400;
   b. *prop.gme*, the effective earth's curvature, is less than 75e–9 or greater than 250e–9;

c. *prop.zgnd.real*, the surface transfer impedance real, (or resistance) component is less or equal to the absolute value of *prop.zgnd.imag*, the imaginary (or reactance) component;
d. *prop.wn*, the wave number, is less than 0.419 (equal to a frequency of 20 Mhz) or greater than 420 (equal to a frequency of 20 Mhz);

Then prop.kwx, the error marker, is set to 4, indicating parameters out of range.

Line 697:
```
if (prop.ens < 250.0 || prop.ens > 400.0 || prop.gme < 75e-9 || prop.gme > 250e-9 || prop_zgnd.real() <= abs(prop_zgnd.imag()) || prop.wn < 0.419 || prop.wn > 420.0)
        prop.kwx=4;
```

9. A **for** statement is initiated with two loops, j=0 and j=1.
   a. An **if** statement is initiated to check if hg[0], the transmitter antenna height above ground level, is within its maximum range; if hg[0] is less than one-half meter or greater than three kilometers, prop.kwx, the error marker, is set to equal 4.
   b. An **if** statement is initiated to check hg[1], the receiver antenna height above ground level, as in (a.) above.

Line 700:
```
for (j=0; j<2; j++)
if (prop.hg[j]<0.5 || prop.hg[j]>3000.0)
        prop.kwx=4;
```

10. The value of *dmin* is set to be equal to five times the absolute value of [(prop.he[0] – prop.he[1])], i.e. equal to five times the value of the difference in height between the effective height of the transmit antenna and the receive antenna. The *abs* command ignores any negative sign in the result, causing the result to always be a positive value.

Line 704:
```
dmin=abs(prop.he[0]-prop.he[1])/200e-3;
```

<u>From steps 10 through 20, the coefficients for the Diffraction Range are calculated; including a straight line diffraction value formula based on a straight line drawn through the diffraction attenuation values a3 and a4:</u>

11. The program calls ***adiff*** with inputs (0.0,prop,propa) .

The subroutine ***adiff*** returns *adiff*, the "diffraction attenuation" at the distance *d*.

The value of *q* is set to be equal to the returned value *adiff*. (Note: since the input d = 0.0, the returned value of *adiff* will be 0.0 for point-to-point mode. See subroutine ***adiff***.

Line 705: q=adiff(0.0,prop,propa);

12. *xae* is set to be equal to: $(prop.wn*prop.gme*prop.gme)^{-1/3}$ This is a distance term, with units in meters, based on the cube root of: a rf-signal illuminated square area of the earth's surface, with each side's length equal to the effective radius of the earth (thereby representing an area defined by a square radian of the earth's surface), with this area multiplied by the wavelength per radian of the rf signal (in this case, represented by division by the wave number, (i.e. wn $=2\pi/\lambda$) of the signal. It is used to determine distances d3 and d4, at which diffraction is calculated. From: [Alg. 4.2]; this term is also a partial term derived from equation [ITS67 3.24, with conversion from km to meters, and from frequency to wn]. Where:

    prop.wn is the wave number, equal to the frequency in MHz/47.7.
    prop.gme is the effective earth's curvature, = 1/a.
    a is the earth's effective radius

Line 707: xae=pow(prop.wn*(prop.gme*prop.gme),-THIRD);

13. d3 is set to be equal to the greater of *propa.dlsa* or (1.3787 * *xae* + *propa.dla*): [Alg. 4.3], and the rest of [ITS67 3.24], again, converting km to meters, f to wn.

    where:
    propa.dlsa is the distance value set at line 680, the total smooth earth horizon distance.
    xae value was set at line 707.
    propa.dla is the total horizon distance; the sum of the transmit and receive horizon distances.

The diffraction attenuation is calculated using three locations; distance *d2*, distance *d3* and distance *d4*. Distance d3 is set to be equal to the greater of dlsa or 1.3787*xae+propa.dla. For a transmitter effective height of 300 meters, a receive height of 10 meters, at an FM broadcast band frequency of 100 MHz, distance dlsa calculates to be 37.4 km; xae calculates to be 11km, so if d3 is 37.4 km, d4 is 67.8 km.

Line 708*:* d3=mymax(propa.dlsa,1.3787*xae+propa.dla);

14. *d4* is set to be equal to *d3* plus 2.7574 times *xae* [Alg. 4.4], [ITS67 3.24]

Line 709*:* d4=d3+2.7574*xae;

15. The program calls ***adiff*** with inputs (*d3*,prop,propa) . [Alg. 4.6]

    The subroutine ***adiff*** returns *adiff*, the "diffraction attenuation" at the distance *d = d3*, then *a3* is set to be equal to *adiff*.

Line 710: a3=adiff(d3,prop,propa);

16. The program calls ***adiff*** with inputs (*d4*,prop,propa) . [Alg. 4.6]
The subroutine ***adiff*** returns *adiff*, the "diffraction attenuation" at the distance *d*. = *d4*, then *a4* is set to be equal to *adiff*.

Line 711: a4=adiff(d4,prop,propa);

17. *propa.emd*, a.k.a. m_d, the slope of a straight line formula of form $y = A_{ed} + m_d d$, is set to be equal to: (*a4–a3)/(d4*–d*3*) [Alg. 4.7], [ITS67. 3.38b]

Line 712: propa.emd=(a4-a3)/(d4-d3);

18. *propa.aed*, the intercept of the straight line formula, $y = A_{ed} + m_d d$ is set to be equal to the solution to the straight line formula (*a3 – propa.emd*d3*) [Alg. 4.8], [ITS67. 3.38b]

Line 713: propa.aed=a3-propa.emd*d3;
}

19. The first **if** statement has run its course. A new **if** statement is initiated, as per Section 5 of the ITMD, stating that if prop.mdp is greater than or equal to zero, then:
    a. prop.mdp is set to be equal to zero, indicating the area mode has initiated and will continue, and;
    b. prop.dist is set to be equal to *d*, the path distance.

Line 716:
```
if (prop.mdp>=0)
{
prop.mdp=0;
prop.dist=d;
}
```

20. A third **if** statement is initiated. It has three embedded **if** statements that check the path distance [see Section 8 of the ITMD]; so if *prop.dist*, the value of *d* the path distance, is greater than zero, and:

    a. if *prop.dist* is greater than 1,000 kilometers, then:
        prop.kwx, the error value, is set to be equal to the higher value of *prop.kwx* or 1;

    b. if *prop.dist* is less than *dmin*,then;
        prop.kwx is set to be equal to the higher value of *prop.kwx* or 3;

    c. if *prop.dist* is less than 1000 meters, or
        prop.dist is greater than 2000 kilometers, then:
        prop.kwx is set to be equal to 4;

```
Line 722:       if (prop.dist>0.0)
                {
                        if (prop.dist>1000e3)
                                prop.kwx=mymax(prop.kwx,1);

                        if (prop.dist<dmin)
                                prop.kwx=mymax(prop.kwx,3);

                        if (prop.dist<1e3 || prop.dist>2000e3)
                                prop.kwx=4;
                }
```

21. The third **if** statement, and its three embedded **if** statements, have run their course. A fourth primary **if** statement is initiated, stating that if prop.dist is less than prop.dlsa, then:

```
Line 734:       if (prop.dist<propa.dlsa)
                {
```

Note: This "fourth primary **if** statement" will control what happens, and which mode of calculation is used, from the transmitter site to the distance *dlsa*, the value of which is stored in *(propa.dlsa),*the sum of the two calculated smooth earth horizon distances.

In steps 22 through 38, the coefficients for the Line-of-Sight Range, and the line-of-sight path loss, are calculated:

For additional description and theoretical background on the calculations below, see Section 3-1, page 3-2, of ESSA Technical Report ERL 79-ITS67.

22. The fourth primary **if** statement is follow by a series of embedded **if** and **else** statements; the third of these **if** statements states that: if (*wlos*) is a boolean false, indicating that the line-of-sight coefficients have not yet been calculated, then:
    a. Subroutine ***alos2*** is called with input (*0.0,prop,propa*). The subroutine ***alos2*** returns *alosv,* the value of the line of sight attenuation, and *q* is set to be equal to *alosv.*
    b. *d2* is set to be equal to *propa.dlsa*, the sum of the two smooth earth horizon distances;
    c. *a2* is set to be equal to a value solved from a diffraction straight line formula, aed +d2*propa.emd, the zero intercept (aed) and slope (emd) coefficients of which were derived from the attenuation value a3 at distance d3, and attenuation a4, at distances d4, in steps 16 and 17.
    d. Distance d0, the distance closest to the transmitter at which ***alos2*** will be called to determine the line-of-sight attenuation, is estimated and preset to:

d0 =1.908*wave number*transmit effective height* receive effective height.

For example: for he[0] = 300 m., he[1] = 10 m. , f = 100 MHz, (wn = 2.1), d0 would be preset to 12 km.

Line 736:
```
if (!wlos)
{
        q=alos2(0.0,prop,propa);
        d2=propa.dlsa;
        a2=propa.aed+d2*propa.emd;
        d0=1.908*prop.wn*prop.he[0]*prop.he[1];
```

23. We add, in ITWOM, to the "old" methodology, an "if" and "else" statement pair to switch the mode from line-of-sight to diffraction mode at the point when an obstruction blocks the horizon for the transmitter site, or the estimated actual transmitter horizon is reached; at distance *dl[0]*. This will function up to where the path length reaches *dlsa*, where a later set of commands will allow only diffraction mode to be used at distances at or past the distance *dlsa*.

Line (new)
```
if(prop.dist>(prop.dl[0]))
{
        prop.aref=propa.aed+propa.emd*prop.dist;
}
else
{
```

24. The fourth embedded **if** statement following the fourth primary **if** statement, states that if *propa.aed*, the slope of the diffraction straight line formula that was determined in Step 17 based on two diffraction values, a3 and a4, is greater than, or equal to, zero, then:
    a. *d0*, the distance nearest to the transmitter at which ***alos*** will be called to determine line-of-sight attenuation, is reset to be equal to the lesser of: *d0* , as preset in the previous step) or ½ of *propa.dla*; [Alg. 4.28]
    where *propa.dla* is the sum of the two terminal to horizon distances, and:
    b. d1, the second distance at which alos will be called to determine line of sight attenuation, is then set to be equal to: *d0*+0.25*(*propa.dla – d0*); [Alg. 4.29]

Line 743:
```
if (propa.aed>=0.0)
{
        d0=mymin(d0,0.5*propa.dla);
        d1=d0+0.25*(propa.dla-d0);
}
```

25. An ***else*** statement follows, so if *propa.aed* is less than zero, then:

d1 is set to be the greater of: [-*propa.aed*/*propa.emd*] or [0.25**propa.dla*]
[Alg. 4.39]

where:

propa.aed is the slope of the beyond-horizon straight line diffraction value formula solved for diffraction attenuation values a3 and a4, (see Step 17).

p*ropa.emd* is the intercept of the straight line diffraction value formula, (see Step 16).

Prop.dla is the sum of the two actual horizon distances.

Line 749:

```
else
{
        d1=mymax(-propa.aed/propa.emd,0.25*propa.dla);
{
```

26. Subroutine ***alos2*** is called with input (*d1*,*prop*,*propa*). [Alg. 4.31]

The subroutine ***alos2*** returns *alosv*, the value of the line of sight attenuation due to (two-ray) multipath cancellation and clutter, and *a1* is set to be equal to *alosv*.

Line 752:

```
a1=alos2(d1,prop,propa);
```

27. *wq* is then set to be equal to Boolean false.

Line 753:

```
wq=false;
```

28. The fifth embedded **if** statement following the fourth primary **if** statement, states than if *d0* is less than *d1*, then:

a. Subroutine ***alos2*** is called with input (*d0*,*prop*,*propa*). The subroutine ***alos2*** returns *alosv*, the value of the line of sight attenuation, and *a0* is set to be equal to *alosv*. [Alg. 4.30]

b. Subroutine ***alos2*** is called with input (*d2*,*prop*,*propa*). The subroutine ***alos2*** returns *alosv*, the value of the line of sight attenuation, and *a2*, which was previously set to the diffraction attenuation at d2 by a call to ***adiff***, is reset to be equal to the lesser value of a2 or *alosv*. *Note: This is new in* ***lrprop2***.

c. *q* is set to be equal to: ln(*d2*/*d0*).

d. A first estimate of slope k_2 is calculated; *propa.ak2* is set to be equal to:

$$((d2-d0)*(a1-a0)-(d1-d0)*(a2-a0))/((d2-d0)*\ln(d1/d0)-(d1-d0)*q))$$

or zero, whichever is greater;

Note: there are significant confusion and documentation errors here and in several of the following steps, regarding the logarithmic functions. Here:

<u>Re: Calculation of slopes propa.ak1 and propa.ak2 (a.k.a. AK1 and AK2, or K1 and K2)</u>:

In ESSA Technical Report ERL 79-ITS 67, [1967 with a 1970 errata sheet attached] equation 3.20 shows both of the log functions found in step b. and c. above to be $\log_{10}$ functions. The errata pages for this report do not note any changes to be made to this page. The computer source code listing on page 3-40 shows the use of the FORTRAN ALOG10 function for this equation; the errata pages for this report do not note any changes to be made to this line. Since the argument q, which contains one of these log functions, is utilized in step 27 below for the calculation of slope propa.ak1 (a.k.a. k_1, AK1 or k1), where ITS 67 equation 3.21 shows the use of a $\log_{10}$ function, and in step 28, for the recalculation of slope propa.ak2 (a.k.a. k_2, AK2 or k2), where ITS 67 equation 3.22 shows the use of a $\log_{10}$ function. This confusion also affects these subsequent steps. The source code on page 3-40, starting at 10, also shows that an ALOG10 function is used for these equations.

However, in "A Guide to the Use of the ITS Irregular Terrain Model in the Area Prediction Mode",[April 1982] Appendix A, pages 78, and 79, subroutine LRPROP(D), following "42", and "44", shows all of these functions to be ALOG functions without the .4343 correction factor applied, therefore indicating a natural log, or ln, function.

In the "Irregular Terrain Model" [August 1, 2002 Revision], we find in section 16 that q, and ak2, are calculated using a log function without a .4343 correction, suggesting a natural logarithm, or ln function. In an earlier, pre August 1, 2002 version on hand, we find that the log function is missing entirely from the denominator of the ak2 equation, between (d2-d0)* and (d1/d0); therefore the opportunity to clearly state whether this should be a $\log_{10}$ or ln function was missed when this equation was corrected. This section references the "Algorithm", equation 4.32.

In the ITS Irregular Terrain Model, version 1.2.2. The Algorithm, (date unspecified, but bibliography lists a 1985 reference to a Memorandum to users of the ITS Irregular Terrain Model); we find:

a. **In equation 4.32, the use of ln, the natural logarithm, is specified for calculating K'_2, as in steps a and b above.**
b. **In equation 4.33 the use of ln, the natural logarithm, is specified for calculating K'_1 as in step 27 below.**
c. **In equation 4.35, we find for the case of $K'_1 < 0$, $K''_2 = (A_2 - A_0) / \ln(d_2/d_0)$; the use of ln, the natural logarithm, is specified for recalculating K''_2 in step 28 below.**

In the ITMDLL.cpp source code, we find the use of the *log* function (ln, or natural logarithm) for all of the steps discussed above. (In c++, the *log* function is a natural logarithm; the *log10* function is a logarithm to the base 10). The specific source code lines are:

```
q=log(d2/d0);

propa.ak2=mymax(0.0,((d2-d0)*(a1-a0)-(d1-d0)*(a2-a0))/((d2-d0)*log(d1/d0)-(d1-d0)*q));

propa.ak1=(a2-a0-propa.ak2*q)/(d2-d0);

propa.ak2=FORTRAN_DIM(a2,a0)/q;
```

For now, it appears that the functions to calculate the line formulas have been deliberately changed to a natural logarithm function, as all instances appear to have been changed. See the discussion in the section on correcting the Irregular Terrain Model.

And then;

d. if *propa.aed*>=0.0 or *propa.ak*2>0.0

wq is set to be equal to Boolean true;

```
Line 755:  if (d0<d1)
           {
               a0=alos2(d0,prop,propa);
               a2=mymin(a2,alos2(d2,prop,propa))
               q=log10(d2/d0);
               propa.ak2=mymax(0.0,((d2-d0)*(a1-a0)-(d1-d0)*(a2-a0))/((d2-d0)*log10(d1/d0)-(d1-d0)*q));
               wq=propa.aed>=0.0 || propa.ak2>0.0;
```

29. A second **if** statement is embedded within the **if** statement above. If *wlos* is not a Boolean "true", and if *wq* is boolean "true", then a first estimate of slope k_1 is calculated; *propa.ak1* is set to be equal to: (*a2-a0-propa.ak2**log*(d2/d0)*)/(*d2-d0*), since *q*=log(*d2*/*d0*). [Alg. 4.33], [ITS67 3.21]

```
Line 762:  if (wq)
           {
               propa.ak1=(a2-a0-propa.ak2*q)/(d2-d0);
```

30. An **if** statement is embedded within the **if** statement on line 762. So:

a. If *wlos* is not a Boolean "true", and;
b. if *d0* is less than *d1*;
c. if *wq* is boolean "true", and;
d. if *propa.ak1* is less than zero;

e. then: (1.) *propa.ak1* (slope k_1) is set to be equal to zero, and: [Alg. 4.36], [ITS67. 3.22]

(2.) *propa.ak2* is set to be equal to (*a2 – a0)/*q if *a2* is greater than *a0;* if *a2* is not greater than *a0,* the FORTRAN_DIM function returns zero, and *propa.ak2* is set to be equal to 0.0/*q,* i.e. zero. [Alg. 4.35]

Line 766:

```
if  (propa.ak1<0.0)
{
        propa.ak1=0.0;
        propa.ak2=FORTRAN_DIM(a2,a0)/q;
```

31. An **if** statement is embedded within the above **if** statement, So:
    a. if *wlos* is not a Boolean "true", and;
    b. if *d0* is less than *d1*;
    c. if *wq* is boolean "true", and
    d. if *propa.ak1* is less than zero, and
    e. if *propa.ak2* is equal to zero, then:
    f. *propa.ak1* is set to be equal to *propa.emd,* the intercept of the diffraction straight line formula . [Alg. 4.37]

Line 771:

```
        if (propa.ak2==0.0)
                propa.ak1=propa.emd;
    }
}
}
```

32. At this point, the **if** statements at Lines 771, 766, and 762 have completed their run. The **if** statements at Lines 755 and 736 are still active.

Note: Here, in the ITM version 7.0 released June 26, 2007, else and if statements were removed, and an else statement became an if statement. The modification performs the same actions as the old code, in 10 fewer lines. These changes are incorporated below:

33. The **else** statement from line 776 ends its run; the **if** statements on line 755 and 736 are still active. A new ***if !****wq* statement follows, leading to a FORTRAN_DIM call; So:
    d. if *wlos* is not a Boolean "true", indicating the line-of-sight coefficients have not yet been calculated, and;
    e. if *wq* is not a Boolean "true", indicating that the other case applies, then:
        (1) *propa.ak1* is set to be equal to (*a2-a1*)/(*d2-d1*) if a2 is greater than a1; if a2 is equal to or less than a1, FORTRAN_DIM returns a zero value for (a2-a1), and *propa.ak1* is set to be equal to zero. [Alg. 4.41]
        (2) *propa.ak2* is set to be equal to 0.0. [Alg. 4.40]

Line (Alternate code): if (! wq)

```
{
        propa.ak1=FORTRAN_DIM(a2,a1)/(d2-d1);
        propa.ak2=0.0;
```

34. An **if** statement is embedded within the **if (!** *wq***)** statement, so:
    a. if *wlos* is not a Boolean "true", and;
    b. if *propa.ak1* is equal to zero,
    c. *propa.ak1* is set to be equal to *propa.emd*.

Line (Alternate Code):

```
        if (propa.ak1= =0.0)
        {
                propa.ak1=propa.emd;
        }
}
```

35. The **else** statement on line 786 has now completed its run. Here:
    a. *propa.ael*, the estimated line of sight attenuation, is set to be equal to *a2 – propa.ak1 * d2 – propa.ak2 ** ln(*d2*), [ITS67. 11, p.16]
    and:
    b. *wlos* is set to be equal to: Boolean" true", indicating completion of the calculation of the line-of-sight coefficients.

Re: <u>Use of logarithms in computing propa.ael, the estimated line-of-sight attenuation</u> :

A logarithmic function is used in this subroutine for the calculation of propa.ael.

In ESSA Technical Report ERL 79-ITS 67, [1967 with a 1970 errata sheet attached] equation 11, on page 16, shows the formula used to calculate the value of propa.ael; the logarithmic function shown is a $\log_{10}$ function. The text notes that it is a simplified version of equation 10. The errata pages for this report do not note any changes to be made to this page. The FORTRAN computer source code listing on page 3-34 shows: AE = AOG – K1*D0-K2*ALOG10(D0), using the base 10 logarithmic function, ALOG10, for this equation. The errata pages for this report do not note any changes to be made to this line.

However, changes are made to this set of equations between ITS-67 and ITM 1.2.2. In "A Guide to the Use of the ITS Irregular Terrain Model in the Area Prediction Mode",[April 1982] Appendix A, page 79, subroutine LRPROP(D), at "46", shows the use of an ALOG function without the .4343 correction factor applied; therefore showing a natural log, or ln, function.

In the Irregular Terrain Model, section 16, we find ael calculated with a log function without an associated .4343 correction. This line references the Algorithm, equation 4.42.

In "The Algorithm", At equation 4.42 we find:

$$A_{el} = A_2 - K_1 d_2$$

which appears to be missing the last term; - K_2ln(d2), as the source code states:

```
propa.ael=a2-propa.ak1*d2-propa.ak2*log(d2);
```

We also find that the full reference attenuation equation, which adds in ael as a term, also uses the ln function; we will assume that the correct log function for the ITM 1.2.2 variable ael is, therefore, a natural, or base e, ln function, which is coded in c++ as a "log" function. Returning to the main discussion:

```
Line 795:                propa.ael=a2-propa.ak1*d2-propa.ak2*log(d2);
                         wlos=true;
                 }
        }
```

<u>The next step calculates the reference attenuation, aref, for the line of sight range, if the old methodology is being used for a terrain database with intervals of 500 meters or more:</u>

36. An **if** statement is initiated. The **if** statement on line 736 is still active, so:
    - *a.* if *wlos* was not a Boolean "true" when checked by the **if** statement in Step 21, and;
    - *b.* if *prop.dist* is greater than zero, then:
    - c. An additional **if** statement is initiated. If in addition to the above, the path distance *prop.dist* is greater than *prop.dl[0]*, the actual distance to the first obstacle or estimated actual transmitter horizon distance, then diffraction mode applies, and:
    - *d.* *prop.aref* is set to be equal to:

      ```
      prop.aref=propa.aed+propa.emd*prop.dist;
      ```

      The same as when prop.dist is between distance dlsa and dx.

    - *e.* An ***else*** statement follows, so if the path distance *prop.dist* is less than *prop.dl[0]*, then: *prop.aref* is set to be equal to:

      propa.ael + *propa.ak1* * *prop.dist* + *propa.ak2* * log(*prop.dist*)
      [Alg. 4.1]

where:
propa.ael is the attenuation for line-of-sight
propa.ak1 is the slope k1
propa.ak2 is the slope k2
propa.dist is the total radio path distance

Note that this line-of-sight calculation is, at this point, eligible to be applied to all path locations that are greater than zero.

Re: <u>Use of logarithm is calculating the reference attenuation, prop.aref</u>:

A logarithmic function is used in this subroutine for the calculation of the reference attenuation, prop.aref.

In ESSA Technical Report ERL 79-ITS 67, [1967 with a 1970 errata sheet attached] equation 10, on page 16, shows the formula used to calculate the value of propa.ael; (a.k.a. A_{cr}). The logarithmic function shown is a $\log_{10}$ function. The errata pages for this report do not note any changes to be made to this page. The FORTRAN computer source code listing on page 3-40 shows at 50: AG = AO + K1*(D-D0) + K2*ALOG10(D/D0), using the base 10 logarithmic function, ALOG10, for this equation; three lines later, at 53, we find: ACR=AG. Again, the errata pages for this report do not note any changes to be made to these two lines.

However, change are made to this master curve equation for the line of sight range between ITS-67 and ITM 1.2.2, and one of the changes is that both log functions change from base 10 common logarithms ($\log_{10}$), to base e natural logarithms (ln);

In "A Guide to the Use of the ITS Irregular Terrain Model in the Area Prediction Mode",[April 1982] Appendix A, page 79, subroutine LRPROP2(D), following "48", shows this function to be an ALOG function without the .4343 correction factor applied; therefore showing a natural log, or ln, function.

In the Irregular Terrain Model, section 15, we find aref calculated with a log function without an associated .4343 correction. It states "if (dist >0.)
***aref = ael +ak1*dist + ak2* log(*dist*)" and then references the Algorithm, equation 4.1.**

In "The Algorithm", At equation 4.1 we find the use of "ln":

Thus indicating the use of a natural logarithm function. And in the ITMDLL.cpp we find:

```
prop.aref=propa.ael+propa.ak1*prop.dist+propa.ak2*log(prop.dist);
```

Therefore, since all instances were changed, we will assume that the correct log functions here are natural, or base e, ln functions, which are coded in c++ as "log".

```
Line  799:      if(prop.dist>0.0)
                {
                    if(prop.dist>(prop.dl[0]))
                   {
                    propa.aref=propa.aed+propa.emd*prop.dist;
                    }
                    else
                    {
                    prop.aref=propa.ael+propa.ak1*prop.dist+propa.ak2*log(prop.dist);
                     }
                }
```

37. Here we find the ***else*** statement that provides an alternate path to the ***if*** statement far above that splits the path between the old ITM method and the ITWOM method. So for this else statement, if:
    a. the path distance is less than distance dlsa;
    b. the interval width, iw, is more than zero and less than 150 meters, then the new ITWOM method applies.

```
Line(new): else
                {
```

38. An if statement is initiated; if *wlos* is not a Boolean true, indicating the line-of-sight coefficients have not been prepared, then subroutine ***alos2*** is called with inputs (*0.0*,*prop*,*propa*) to set up the coefficients for a full run of ***alos2***, and the meaningless output *alosv* is stored in the utility storage argument q.

39. An if statement is initiated; if the integer value of prop.dist-propa.dla, the path length, *prop.dist*, less the sum of the actual horizon distances, *dla*, is less than 0.0, indicating that the horizon distances overlap and therefore indicating a line of sight condition exists, then subroutine ***alos2*** is called with inputs (*pd1*,*prop*,*propa*). The argument *pd1* is equal to the path distance; the output, *alosv*, the attenuation value for a line of sight path, is stored in *prop.aref*.

40. An else statement provides an alternative path to the if statement directly above it, so if the path length less the sum of the actual horizon distances is equal to or greater than zero, one or more obstructions exist in the path, or the path has reached the transmitter site horizon. The path terminates at the first obstacle peak or horizon, or is beyond the line of sight.

```
Line(new):      else
                {
```

41. An if statement is initiated; if the integer value of prop.dist-propa.dla, the path length, *prop.dist*, less the sum of the actual horizon distances, *dla*, is equal to 0.0,

indicating that a single obstacle or mutual horizon condition exists, and if the integer value of the receive site horizon distance is equal to zero, indicating that the receive site is at the transmitter obstacle peak, or that is at a diffracted mutual horizon point, then the line of sight computation subroutine ***alos2*** is called with inputs (*pd1*,*prop*,*propa*). The argument *pd1* is equal to the end-of-line-of-sight path distance to the peak or transmitter horizon; the output, *alosv*, the attenuation value for a line of sight path, is summed with 5.75 dB, the diffraction loss at a peak, and stored in prop.aref.

```
Line(new):      if (int(prop.dist-propa.dla)==0.0)  && (int(prop.dl[1])==0.0)
                {
                        prop.aref=5.75+alos2(pd1,prop,propa);
                }
```

42. The if statement controlling what happens if prop.dist, the path distance is less than distance dlsa, ends its run, and an else statement provides an alternative path to compute what happens when prop.dist is equal to or greater than distance dlsa.

```
Line(xxx): }
            else
           {
```

43. Here we again split the path between the old and the new. We retain an improved version of the old path; a completed and corrected version of the old methodology, appropriate to use with no database (area mode) or old, granular databases (with data points farther apart than 3 arc-second), and also add a new, completed methodology for use with precise, point to all terrain point computations. In the last section above, the coefficients for the diffraction range and line-of-sight range, and computations for the old and new path computations for distances less than dlsa were done.

    To split the path, we initiate the below if statement, to make a decision based on the distance between terrain data points. This is set at 130 meters, to accommodate the distance between the centers of two adjacent 3-arc-second pixels at the equator, located so that only the corners of the pixels touch.

    So if xi is zero, indicating no terrain database information, or is 130 meters or more, the improved old computer methodology is utilized.

```
Line (new):     if ((iw=0.0) || (iw>130))
                {
```

44. An else statement is initiated, providing an alternate path to the if (iw==0.0 || iw=>130) statement in step 22; so if 0<iw<130, where iw=pfl[1], distance per increment, (i.e. distance between elevation height data points) in meters, the new

actual-points line-of-sight and diffraction methodology is followed. This "new" methodology is derived from the original, manual calculation methods in Tech Note 101. This original procedure allowed the terrain height at an obstacle or the horizon to be precisely determined and obtained from a hand-drawn radial on a paper terrain map. Failures to extend the Tech Note 101 horizon distance-based equations to properly consider obstacles have been corrected. So if the path distance prop.dist is less than dlsa, and if iw is between 0.0 and 130.0 meters, then:

Line (new):
```
else
{
```

45. An ***if*** statement is initiated, so if:
    a. the path distance prop.dist is less than dlsa, and:
    b. *iw* is between 0.0 and 130.0 meters, indicating an ITWOM database, and:
    c. Boolean argument *wlos* is not true, meaning that the initial setup run of alos2 with input 0.0 has not been made, then:
    d. subroutine ***alos2*** is called with input (0.0,*prop*,*propa*) to set up the coefficients within alos2.
    e. The boolean argument *wlos* is set to be equal to true.

Line (new):
```
if (!wlos)
{
        q=alos2(0.0,prop,propa);
        wlos=true;
}
```

46. An ***if*** statement is initiated, so if:
    a. the path distance prop.dist is less than dlsa, and:
    b. *iw* is between 0.0 and 130.0 meters, indicating an ITWOM database, and:
    c. the integer value of prop.dist, the path distance, subtracted from dla, the distance to the peak of the transmitter tallest visible obstacle or horizon distance, is less than zero, (i.e. the path distance is less than the distance to the obstacle or horizon), indicating a location within the line-of-sight range; then:
    d. subroutine ***alos2*** is called with input (*pd1*,*prop*,*propa*). The subroutine ***alos2*** returns *alosv*, the value of the line of sight attenuation due to (two-ray) multipath cancellation and clutter. The variable *prop.aref* , representing the <u>ref</u>erence <u>a</u>ttenuation, is set to be equal to *alosv.*

Line(new):
```
if ((int(prop.dist-prop.dla)==0.0  && (int(prop.dl[1]==0.0))
{
        prop.aref=alos2(pd1,prop,propa);
```

47. An else statement provides that if:
    a. the path distance prop.dist is less than dlsa, and:

b. *iw* is between 0.0 and 130.0 meters, and:
c. prop.dist, the path distance, is equal to or greater than dla, the sum of the transmitter and receiver horizon distances, indicating a location at or past the peak of a first obstacle, and therefore at the end of, or past the line-of-sight range and in a diffraction range with one obstacle, then:

Line(new): else
{

48. An ***if*** statement is initiated, so if:
    a. the path distance prop.dist is less than dlsa, and:
    b. *iw* is between 0.0 and 130.0 meters, indicating an ITWOM database, and:
    c. the integer value of prop.dist, the path distance, subtracted from dla, the distance to the peak of the transmitter tallest visible obstacle or horizon distance, is equal to zero, (i.e. the path distance is equal to the distance to the obstacle or horizon), indicating a location at or near the peak of the first obstacle or horizon, and the integer value of prop.dl[1], the length of the path from the obstacle to the receiver site is equal to zero, indicating that the receive antenna is at the peak of the first major obstacle or horizon, then:
    d. subroutine ***alos2*** is called with input (*pd1*,*prop*,*propa*). The subroutine ***alos2*** returns *alosv*, the value of the line of sight attenuation due to (two-ray) multipath cancellation and clutter between the transmit site and the receive point on the peak of the obstacle.
    e. . The variable *prop.aref*, representing the reference attenuation, is set to be equal to the value of *alosv*, plus 5.8 db to account for the knife-edge diffraction loss at the peak of the first major obstacle.

```
Line(new):    if ((int(prop.dist-prop.dla)==0.0  && (int(prop.dl[1]==0.0))
              {
                      propa.aref=5.8+alos2(pd1,prop,propa);
```

In the above step, we have a special condition; the end of a line of sight path terminating in a diffraction-affected receive point at a horizon point or the top of an obstacle. At the top of an obstacle, the $\Delta r = (r_1 + r_2 - r_0)$ in the computation of $v = 2(\Delta r / \lambda)^{1/2}$ for a diffraction attenuation computes to be zero, as r_1 , equal to *prop.dl[0]*, the transmitter horizon distance, is equal to = r_0 , the path distance *prop.dist* or *pd1*, and r_2 , the receive horizon distance *prop.dl[1]*, is zero. Therefore v and v2 compute to be zero; the diffraction equation attempts to compute the log10(v) where v=0.0, and defaults to not-a-number (nan). There is diffraction attenuation here, coming from the cancellation of the incoming signal by reflections off of the leading side of the knife (i.e.the incoming signal side of the obstruction). From Figure 7.1, Tech Note 101, it can be seen that the value of attenuation for v=0 is approximately 5.8 dB, and is frequency insensitive as the frequency compensation is included in v. Since we have limited information as to the shape of the obstruction in a 3-arc-second database, we conservatively assume a knife-edge equivalent

at this point, and ignore additional attenuation that may occur due to rounding at this location. The equivalent adjustment for a receive point at the top of a second obstruction is incorporated into subroutine ***alos2***, where a receive point at the top of the second obstruction is treated as a single obstruction path with 5.8 dB added to account for the knife edge diffraction at the knife point.

49. An if statement is initiated. If:
    a. the path distance prop.dist is less than dlsa, and:
    b. *iw* is between 0.0 and 130.0 meters, and:
    c. prop.dist, the path distance, is greater than dl[0], the distance from the transmitter to the peak of the first major obstacle or horizon, and
    d. prop.dl[1] is greater than zero, indicating a receive location past the line-of-sight range,
    e. then subroutine ***adiff2*** is called twice, once with inputs (*0,0,prop,propa*) to set the coefficients, and a second time with inputs (*pd1,prop,propa*), to compute the diffraction at the path distance *pd1*. The argument prop.aref is set to be equal to the output adiffv of the subroutine, the attenuation from diffraction:

```
Line(new):      if  ((int(prop.dist-prop.dl[0])>0.0) && (int(prop.dl[1]>0.0))
                {
                        q=adiff2(0.0,prop,propa);
                        prop.aref=adiff2(pd1,prop,propa);
                }
```

50. An else statement provides an emergency alternative to the above:

```
                else
                {
                        prop.aref = 1.0;
                }
            }
        }
    }
```

The if statement “if (prop.dist<propa.dlsa)” has now run its course. We have finished calculating the coefficients for the Line of Sight range, and have calculated the value of *aref* if the path is line-of-sight from the transmit terminal to the receive terminal, or has switched to diffraction mode prior to reaching the distance *dlsa*.

<u>In the following steps, coefficients are calculated for the Troposcatter (scatter) range:</u>

51. The last primary **if** statement is initiated at line 804.
    a. If *prop.dist*, the path distance, is less than or equal to zero, or:
    b. *prop.dist*, is greater than *propa.dlsa*, the sum of the calculated distances to the smooth earth horizons. This is the point, for a smooth earth condition, where diffraction mode takes over from line of sight mode. The following operations, in steps 38 to 43, therefore apply only at two places on the path; the transmitter site location, and then for all points beyond the distance *dlsa*, the sum of the calculated horizon distances over smooth earth.

```
Line 804:  if  (prop.dist==0.0 || prop.dist>=propa.dlsa)
               {
```

52. Here we again provide a split path for the old ITM procedure and the new ITWOM methodology. An if statement is initiated; if *iw* is zero, indicating no terrain data, or greater than 130 meters, indicating a terrain database with larger than 3-arc-second data intervals, the old ITM system proceeds. If not, the program path jumps to the else statement below and follows the ITWOM methodology.

```
Line (new):     if ((iw==0.0) || (iw>130.0))
                {
```

53. If the interval width is zero or greater than 130.0 meters, then there is an additional embedded **if** statement immediately following; so:

    a. if *wscat* is not Boolean *true* (i.e. is Boolean *false),*indicating that the scatter coefficients have not been calculated; then subroutine ***ascat*** is called with inputs (0.0, *prop,propa*).
        1) Subroutine ***ascat*** returns *ascatv*, the value of the "scatter attenuation", and the utility value storehouse argument *q* is reset to be equal to *ascatv*.
        2) *d5* is set to be equal to *propa.dla* + 200,000 meters. [Alg. 4.52]
        3) *d6* is set to be equal to *d5* + 200,000 meters, i.e. = *propa.dla* + 400,000 meters. [Alg. 4.53]
        4) subroutine ***ascat*** is called with inputs (*d6*, *prop,propa*). Subroutine ***ascat*** returns *ascatv*, the value of the "scatter attenuation", and *a6* is reset to be equal to *ascatv* [Alg. 4.54]
        5) subroutine ***ascat*** is called with inputs (*d5*, *prop,propa*). Subroutine ***ascat*** returns *ascatv*, the value of the "scatter attenuation", and *a5* is reset to be equal to *ascatv* [Alg. 4.55]

Line(changed):

```
if(!wscat)
{
        q=ascat(0.0,prop,propa);
        d5=propa.dla+200e3;
        d6=d5+200e3;
        a6=ascat(d6,prop,propa);
        a5=ascat(d5,prop,propa);
```

54. An **if** statement, embedded under the **if** statement at line 806, which is embedded under the primary **if** statement at line 804, is initiated. So if:
    a. *prop.dist*, the path distance, is less than or equal to zero, or:
    b. *prop.dist*, is greater than *propa.dlsa*, and:
    c. if *wscat* is not Boolean *true* (i.e. is Boolean *false),*and;
    d. if *a5* is less than 1000, then:
        (1) *propa.ems* is set to be equal to: (*a6*-*a5*)/(200000 meters) [Alg. 4.57]
        (2) *propa.dx*, the distance where diffraction mode attenuation is equal to scatter mode attenuation, and where diffraction mode dominance actually gives way to scatter mode dominance, is set to be equal to the greater of: [*propa.dlsa*] or [the greater of (*propa.dla* + 0.3 * *xae* * log(47.7 * *prop*.wn),[ITS67 3.44b) or ((*a5*-*propa.aed*-*propa.ems**d5)/(*propa.emd*-*propa.ems*))]; [Alg. 4.58] and [ITS67. 3.44a]

Re: <u>Use of logarithm is calculating the distance dx</u>:

A logarithmic function is used in this subroutine for the calculation of distance *dx*. ITS67 documentation, equation 3.44a, shows this function to be a $\log_{10}$ equation, and the source code shows a ALOG10 function on the line performing this function, after 30 in subroutine DIFF.

In "A Guide to the Use of the ITS Irregular Terrain Model in the Area Prediction Mode",[April 1982] Appendix A, page 79, subroutine LRPROP(D), following "51", shows this function to be an ALOG function without the .4343 correction factor applied; therefore showing a natural log, or ln, function.

In the Irregular Terrain Model, section 21, we find dx calculated with a log function without an associated .4343 correction. This line references the Algorithm, equation 4.58.

In "The Algorithm", At equation 4.58 we find the use of "log" with no .4343 correction factor:

$$d_x = \max[d_{Ls}, d_L + X_{ae}\log(k*H_s), (A_5 - A_{ed} - m_s d_5)/m_d - m_s)]$$

Thus indicating the use of a natural logarithm function. And in the ITMDLL.cpp we find:

```
propa.dx=mymax(propa.dlsa,mymax(propa.dla+0.3*xae*log(47.7*prop.w
n),(a5-propa.aed-propa.ems*d5)/(propa.emd-propa.ems)));
```

Which suggests that this function should be a natural, or ln function, which c++ implements using the *log* subroutine. Returning to the main discussion:

(3) propa.aes is set to be equal to:
(*propa.emd -propa.ems*) * *propa.dx* + *propa.aed* .
[Alg. 4.59] and [ITS67. 3.44c]

Line 814:
```
if (a5<1000.0)
{
    propa.ems=(a6-a5)/200e3;
    propa.dx=mymax(propa.dlsa,mymax(propa.dla+0.3*xae*log(47.7*
    prop.wn),(a5-propa.aed-propa.ems*d5)/(propa.emd-propa.ems)));
    propa.aes=(propa.emd-propa.ems)*propa.dx+propa.aed;
}
```

55. An **else** statement provides an alternate path to the **if** statement immediately above. So if:
    a. *prop.dist*, the path distance, is less than or equal to zero, or:
    b. *prop.dist*, is greater than *propa.dlsa*, and;
    c. if *wscat* is not Boolean *true* (i.e. is Boolean *false),*and;
    d. if *a5* is equal to or greater than 1000, then:
        (1) propa.ems is set to be equal to: *propa.emd.*
        (2) propa.aes is set to be equal to *propa.aed.*
        (3) propa.dx is set to be equal to: 10,000,000. [Alg. 4.56]

Line 821:
```
else
{
    propa.ems=propa.emd;
    propa.aes=propa.aed;
    propa.dx=10.e6;
}
```

56. The value of *wscat* is then set to be equal to a Boolean "*true;*" The **if** statement at line 806 then ends its run.

Line 828:
```
    wscat=true;  (Scatter coefficients calculated and ready)
}
```

<u>The coefficients for the Troposcatter (scatter) range for the ITM method have now been calculated. In the following steps the reference attenuation, will be computed as per [Alg. 4.1] for the ITM method, if the path ends in the scatter or diffraction ranges.</u>

57. An **if** statement, embedded within the **if** statement in Step 38, is initiated; so if:
    a. *prop.dist*, the path distance, is less than or equal to zero, or:
    b. *prop.dist*, is greater than *propa.dlsa*, and;
    c. if *prop.dist* is greater than *propa.dx*, indicating that we are in the troposcatter (scatter) range, then:
    d. *prop.aref* is set to be equal to: *propa.aes* + *propa.ems* * *prop.dist*;

Line 831:
```
if (prop.dist>propa.dx)
{
prop.aref=propa.aes+propa.ems*prop.dist;
}
```

58. An **else** statement provides an alternative path to the **if** statement directly above, which is still embedded within the the **if** statement in Step 38; so if:
    a. *prop.dist*, the path distance, is less than or equal to zero, or:
    b. *prop.dist*, is greater than *propa.dlsa*, indicating we are past the line-of-sight range for a smooth earth situation, and;
    c. if *prop.dist* is equal to or less than *propa.dx*, indicating that we have not yet arrived at the distance where diffraction dominance rolls over to (tropo)scatter dominance, (the combination of b. and c. therefore indicating that we are in the diffraction dominant area) then:
    d. *prop.aref* is set to be equal to: *propa.aed* + *propa.emd* * *prop.dist*;

Line xxx:
```
else
{
        prop.aref=propa.aed+propa.emd*prop.dist;
}
```

<u>In the following steps, coefficients are calculated for the Troposcatter (scatter) range for the ITWOM procedure; for a database with intervals 3-arc-second width or smaller:</u>

59. An **else** statement follows; if the path increment distance is greater than zero and less than 130 meters, the ITWOM methodology is followed, and:

60. An **if** statement immediately follows; so:

    a. if *wscat* is not Boolean *true* (i.e. is Boolean *false),*indicating that the scatter coefficients have not been calculated; then subroutine ***ascat*** is called with inputs (0.0, *prop,propa*) to prepare coefficients. Subroutine

ascat returns *ascatv*, the value of the "scatter attenuation", and *q* is reset to be equal to *ascatv.*

b. The subroutine ***ascat*** is called again, this time with inputs (*pd1*, *prop,propa*). Subroutine ***ascat*** returns *ascatv*, the value of the "scatter attenuation", and *a5* is reset to be equal to *ascatv.*

Line(changed):

```
if(!wscat)
{
        q=ascat(0.0,prop,propa);
        a6=ascat(d5,prop,propa);
```

61. Then ***adiff*** is called twice:

a. once with inputs (0.0, prop,propa) to set up coefficients;

b. and then the second time, this time with inputs (pd1,prop,propa) to calculate the diffraction attenuation at path distance pd1.

c. The subroutine ***adiff*** returns *adiff*, the "diffraction attenuation" at the path distance *d*.

Line 833:

```
q=adiff(0.0,prop,propa);
a5=adiff(pd1,prop,propa);
```

62. An ***if*** statement is initiated, embedded within the above ***if*** statement. If the result from the diffraction attenuation calculation is less than or equal to the scatter attenuation result, then propa.dx is set to 1,000 km. and prop.aref is set equal to the result of the diffraction attenuation, *a5*.

Line(new):

```
if (a5<=a6)
{
        propa.dx=10000000;
        propa.aref=a5;
{
```

63. An else statement provides an alternative path; if the scatter attenuation calculation result, *a6*, is less than the diffraction attenuation, then propa.dx, the switchover distance to scatter attenuation, is set to be equal to the distance dlsa, and propa.aref is set to be equal to *a6*:

Line(new):

```
else
{
prop.dx=propa.dlsa;
prop.aref=a6;
}
```

64. The coefficients having been set for the troposcatter range, as well as the calculation of attenuation, then wscat is set to be equal to a Boolean true:

```
Line(new):                    wscat=true;
                              }
                    }
          }
```

65. The subroutine ***lrprop2*** then sets *prop.aref*, the reference attenuation, to be equal to the greater of *prop.aref* or zero, and then returns the value of *prop.aref*.

```
Line 837:       prop.aref=mymax(prop.aref,0.0);
        }
```

SUBROUTINE POINT_TO_POINT_TWO: A functional explanation, by Sid Shumate.

Last Revised: January 20, 2008.

Longley-Rice Point-to-Point Profile

Note: This is the primary subroutine called by the commercial, freeware, or custom wrap-around software for point-to-point calculations. In version 7.0 of the ITMDLL.cpp, released in June of 2007, there are two alternative subroutines, ***point_to_pointDH*** and ***point_to_pointMDH***, that provide improvements. This subroutine initiates the point-to-point mode calculation of signal loss between two points, or terminals, over a single irregular terrain path profile. The two terminals are the transmit terminal, normally the transmitting antenna; and the receive terminal, normally the receiving antenna or location. The subroutine reports out a single value of loss, *aref*, the "reference attenuation" in decibels (dB), of the radio signal between the two terminals. Calls ***qlrps*** and then ***lrprop,*** and then calls ***avar*** to calculate additional loss due to statistical variation before reporting out *errnum*, the error code, and *dbloss*, (a.k.a. *aref*, the reference attenuation) the path loss.

Derived from the Irregular Terrain Model description by George Hufford, 2002, (ITMD), and from "A manual for ITM, "Irregular Terrain Model", a manual for the FORTRAN user, found at: http://flattop.its.bldrdoc.gov/itm/itm_man.txt, compared to the ITM.cpp prepared by J. D. McDonald and John Magliacane for compilation on unix and linux systems. "Line" numbers refer to the ITM.cpp as line numbered by Bloodshed Software's DevC++ print function. "Alg" numbers refer to the algorithm formulas in "The ITS Irregular Terrain Model, version 1.22, the Algorithm" by G. A. Hufford, 1995. "ITS67" numbers refer to the algorithm formulas in "ESSA Technical Report ERL 79-ITS 67, Prediction of Tropospheric Radio Transmission Over Irregular Terrain, A Computer Method – 1968" by A.G.Longley and P.L.Rice.

Other references include: documentation for SPLAT!, an RF Signal Propagation, Loss and Terrain analysis tool.

This subroutine incorporates improvements from ***point_to_pointDH*** and ***point_to_pointMDH***, and changes to support new or modified subroutines saalos, alospo, and lrprop2.

From ITMD Sections: 4 and 5.

Call inputs:

The inputs to this subroutine were originally prepared and placed on IBM punch cards for input into a 1970's mainframe computer that used the 1966, and, later, 1977 ANSI standard version of the FORTRAN computer language. Today, for this c++ code, the user must provide either a commercially written version (ComStudy, PROBE, RFCAD, and TAP being examples), a freeware version (NTIA's ITMsetup.exe, Radio Mobile,

SPLAT, or my own SPLAT with PLOP), or write their own wrap-around input and output processing software to collect and process the following input data for the point_to_point subroutine. If you wish to "roll your own", as a learning tool or as a basis to build on under the GNU GPL license, the full source code for the wrap around software is only freely available for SPLAT and SPLAT with PLOP; both of these are Linux command line programs.

This data comes from a combination of direct data input from the program user, from standard preset value tables, and from a terrain elevation database [such as GLOBE, the USGS National Elevation Database (NED) or the data from the Shuttle Radar Terrain Mission (SRTM)]. At this time, only the experimental SPLAT with PLOP provides the option of using a combination of two of the databases, one as a ground height database (NED) and one as a radio signal reflection height database (SRTM).

elev[ ] a.k.a. pfl, or array pfl, prepared for use in either version 1.2.2. or version 7.0 of the ITM compilied from source code written in c++. As this array is primarily referred to in other parts of the ITM, and in the NTIA documentation, as the pfl array, so will we. This array of elevation values is prepared by the calling program or subroutine. This array contains the values of terrain elevation heights (in meters), equally spaced along a path starting at the transmit terminal, and ending at the receive terminal, and following great circle path, with:

pfl[0] = *enp*, the number of increments between elevation data points, (also one less that the number of data points)

pfl[1] = *xi*, distance per increment, (i.e. distance between elevation height data points) in meters

pfl[2] = *z(0)*, the transmitter tower base AMSL, or elevation height in meters.

pfl[[np+2] = *z(np)*, the receive location AMSL, the last elevation height in meters.

Tht_m a.k.a. hg (0); transmitter antenna center of radiation height in meters, above ground level (RCAGL).

Rht_m a.k.a. hg(1); receive antenna center of reception height in meters, above ground level (RCAGL).

Note: tht_m and rht_m also referred to in the documentation collectively as HG, heights above ground.

Eps_dielect Earth's dielectric constant, a.k..a eps; relative permittivity.
Customary default setting: 15.000
Typical values:

Salt water	80
Fresh water	80
Good ground	25
Farmland, forest	15

Average ground	15
Mountain, sand	13
Marshy land	12
City	5
Poor Ground	4

Sgm_conductivity

Earth's conductivity; a.k.a. sgm.
Customary default setting: 0.005 Siemens per meter
Typical values:

Salt water	5.000
Good ground	0.020
Fresh water	0.010
Marshy land	0.007
Farmland, forest	0.005
Average ground	0.005
Mountain, sand	0.002
City	0.001
Poor Ground	0.001

Eno_ns_surfref

Atmospheric bending constant, (eno); eno varies with elevation above ground; here it is set to equal ens, the refractivity of the atmosphere as it approaches the surface of the earth.
Customary default: ens = 301.000 N-units (parts per million).

Enc_ncc_clcref

(enc) Average clutter canopy refractivity constant. Customary default setting: encc = 1,000 N-units (parts per million) for compatibility with ITU-R P.1546-2 data. Added for ITWOM.

Clutter_height

(cch) Average clutter canopy height. Customary default setting; 25.3 meters above ground level (for compatibility with ITU-R P.1546-2 data). Added as part of ITWOM.

Frq_mhz

Frequency of the transmitter in MHz, Range: 20 to 20000 MHz.

Radio_climate

(klim), the radio climate code; set by the user or obtained from a preset list. Customary Default value is 5. The climate codes are:

1. Equatorial; (Africa, along the equator)
2. Continental Subtropical; (Sudan region)
3. Subtropical (a.k.a. Maritime Subtropical (West Coast of Africa);
4. Desert (Death Valley, NV; Sahara);
5. Continental Temperate (usual general U.S. default);
6. Maritime Temperate Over Land (California to State of Washington; West Coast of Europe including U.K.),
7. Maritime Temperate, Over Sea.

pol polarity of the transmitted signal and receive antenna:
0 for Horizontal; used primarily for television broadcast and FM home reception.
1 for Vertical: used for FM automotive, cellular, cellular automotive, and other 2-way vehicular and handheld radios with vertical whip antennas.
2 for Circular: (new, in development) use only with a circularly polarized receive antenna receiving from a circularly polarized transmitting antenna. If the transmitting antenna is not circularly polarized, do not use. If the antenna is elliptically polarized, compute by individual polarity and combine results based on ratio of polarity. If the transmitting antenna is circularly polarized, and the receive antenna is either horizontal or vertical, use the polarity of the receive antenna.

conf confidence; a statistical percentage of confidence in the situation; set as .01 to .99 . In ***avar,*** the definition of confidence varies with the value of mdvar, the mode of variability; for mdvar = 0, for example, time, location, and situation variability are combined together. For point_to_point mode, mdvar = -1. See step 18 below, or the chapter on subroutine ***avar***, or itm.man. Usual default setting; 0.50 (50%) (Note: often used instead of location calculation, i.e. to approximate 50% of locations; however, ***avar*** has separate inputs for confidence and location, and point_to_point calls ***avar*** to calculate the reference attenuation with the location variable set at 0.0. See note at input loc below regarding optional subroutine ***point_to_pointMDH***.)

rel reliability; a statistical percentage of time availability; set as .01 to .99 Usual default setting; for NTSC (analog) TV, FM broadcast and most FM analog transmissions, set to 0.50 (50% for FCC 50,50); for Digital FM IBOC sidebands, set to .90 or .98, or for television (DTV), set to 0.97 (97% for FCC 50, 97).

NOTE: loc, immediately below, is a new option for point-to-point use; see optional alternative subroutine ***point_to_pointMDH*** released in ITMDLL.cpp version 7.0, June 2007:

loc location, a statistical percentage of location availability; set as .01 to .99. Internally fixed to zero in ***point_to_point***; as ***point_to_point*** calls subroutine ***avar*** with the location variable set at 0.0. In the new version 7, the optional alternative subroutine ***point_to_pointMDH*** allows this to be set by the user. Usual user default setting; 0.50 (50%) for 50% of locations.

mode_var (mdvar) mode of variability (operating mode); Range is 0 to 23; New to enable changes in version 7; see notes below.
Preset to 12.

NOTE: It is interesting to note that for TV or FM broadcast use, in point_to_point mode, the mdvar is set to Mobile mode, not Broadcast. The code for mdvar, the variability mode, which sets the mode of operation, is a tens and single digit code.

The tens digit code is: No tens; default to area mode; combined code is 0 to 3.

10 – For the point-to-point mode. Location variability is eliminated.

20 – For interference problems. Direct situation variability is eliminated. Note that there may be a small residual situational variability.

The single digits represent:

0 - Single message mode. Time, location and situation variability are combined together to give a confidence level.

1 – Accidental mode. Reliability is given by time availability. Confidence is a combination of location and situation variability.

2 – Mobile mode. Reliability is a combination of time and location variability. Confidence is given by the situation variability.

3 – Broadcast mode. Reliability is given by the statement of –at least- qT of the time in qL of the locations. Confidence is given by the situation variability.

Outputs:

&dbloss dbloss, a.k.a. *aref*, the reference attenuation, or RF path loss, in dB

strmode output string for use in printed report, indicating mode of operation of the calculation by printing out “Line of Sight Mode”, “Single Horizon”, or “Double Horizon”, and either “Diffraction Dominant”, or Troposcatter Dominant”.

NOTE: New Option for point-to-point use; see optional alternative subroutine ***point_to_pointMDH*** released in ITMDLL.cpp version 7.0, June 2007, which replaces strmode with a single numerical code to eliminate printing “Line of Sight Mode”, etc. hundreds or thousands of times on the reports.

&errnum errnum, a.k.a. kwx; the error indicator. Must be preset by user input to zero at beginning of run. Indicates:

0 = no warning

1 = Warning: Some parameters are nearly out of range. Results should be used with caution.

2 = Note: Default parameters have been substituted for impossible ones. This value indicates an effect on computations. Since errnum (kwx) is cumulative, this effect on computations by substitution may or may not be true when higher numbers (3, 4, etc.) are reported out.

3 = Warning: A combination of parameters is out of range. Results are probably invalid.

4 and higher = Warning: Some parameters are out of range. Results are

probably invalid.

Note: Users of the TAP software should check the documentation; TAP uses an expanded and modified set of kwx (errnum), definitions.

defines private, or local, arguments:

prop_type prop: array prop with elements:

prop.wn	wave number, = freq. in MHz/47.7 MHz*m; units in 1/meters
prop.ens	surface refractivity (refractivity of the atmosphere)
prop.gme	effective earth curvature
prop.zgnd	surface impedance
prop.zgndreal	real surface impedance (resistance component)
prop.zgndimag	imaginary surface impedance (reactive component)
prop.encc	average clutter canopy top refractivity
prop.cch	average clutter canopy top height AGL
prop.ptx	polarity of transmitted signal

prop_type propa; array with elements:

zsys =0	zsys; preset to zero; later calculated to be the average elevation height along a selected portion of the total RF path (between *ja* and *jb).*
zc	conf, or confidence level to be calculated, in percent
zr	rel, or reliability level to be calculated, in percent
eno	atmospheric bending constant
enso	atmospheric bending constant to be preset to zero.
q	utility value-holding variable
ja	a location along the RF path that is 1/10 of the total RF path length rounded to the nearest increment, plus three increments, away from the transmit terminal. Specified in units of increments.
jb	a location along the RF path that is 1/10 of the total RF path length rounded to the nearest increment, plus three increments, away from the receive terminal. Specified in units of increments.
i	incremented value in a **for** loop
np	number of points, the total number of increments between the elevation height measurement points from the profile array starting with the transmit terminal site and ending with the receive terminal.
dkm	total RF path distance, in kilometers
xkm	distance per path increment, in kilometers per increment
fs	free space path loss, in dB

This subroutine:

1. Sets *prop.hg[0]* , the height above ground level of the transmitting antenna center of radiation, to be equal to *tht_m*, and sets *prop.hg[1]*, the height above ground level of the receiving antenna center of radiation to be equal to *rht_m*;

Line 1418: prop.hg[0]=tht_m;
prop.hg[1]=rht_m;

2. Sets *propv.klim* to be equal to the *radio_climate* value set by the user; the customary default being 5, Continental Temperate;

Line 1420: propv.klim=radio_climate;

3. Sets *prop.encc* to be equal to the *enc_ncc_clcref* value set by the user; the customary default being 1000.000;

Line 1420: prop.encc=enc_ncc_clcref;

4. Sets *prop.cch* to be equal to the *clutter_height* value set by the user; the customary default being 25.3;

Line 1420: prop.cch=clutter_height;

5. Resets prop.kwx, the error indicator (a.k.a. errnum) to be equal to zero.

Line 1421: prop.kwx=0;

6. Presets *propv.lvar* to be equal to five.

Line 1422: propv.lvar=5;

7. Sets *prop.mdp* , the mode of propagation, to be –1, which indicates operation in the point-to-point mode;

Line 1423: prop.mdp=-1;

8. Sets *prop.ptx*, the polarity of the transmitted signal, to be 0 (horizontal.), 1 (vertical), or 2 (circular), going by the user input value *pol*.

Line 1423: prop.ptx=pol;

9. Sets *zc*, the confidence level to be calculated by ***avar***, to be equal to *qerfi(conf)*, and sets *zr*, the reliability level to be calculated by ***avar***, to be equal to *querfi(rel)*.

Line 1424: zc = qerfi(conf);
zr = querfi(rel);

10. Sets *np*, the number of increments (one less that the total number of elevation points in the path), equal to the value of *elev[0]* (a.k.a. pfl [0]). The SPLAT version, itm.cpp, defines the argument *np* to be **long**(*elev[0]*) to handle much more detailed (more increments per km) RF terrain paths than those originally anticipated by the ITMDLL.cpp.

Line 1426: np=(long)elev[0];

11. Sets *dkm* to be equal to the number of increments multiplied by the distance in meters between elevation points, divided by 1000 meters per kilometer. The result is the total RF path distance between the terminals, in kilometers.

Line 1427: dkm=(elev[1]*elev[0])/1000.0;

12. Sets xkm to be equal to the distance in meters between elevation points, divided by 1000 meters per kilometer. The result is the length of one increment, in kilometers.

Line 1428: xkm=elev[1]/1000.0;

13. Sets *eno*, the atmospheric bending constant (relative permittivity) equal to the value of eno_ns_surfref;

Line 1429: eno=eno_ns_surfref;

14. Sets *enc*, the average clutter canopy refractivity constant equal to the value of enc_ncc_clcref;

Line (new): enc=enc_ncc_clcref;

15. Sets cch, the average clutter canopy height above ground level, equal to the value of clutter_height.

Line (new) cch=clutter_height;

16. Presets *enso* equal to zero.

Line 1430: enso=0.0;

17. Presets *q* to be equal to *enso*, which in step 13, was preset to be equal to zero.

Line 1431: q=enso;

18. An **if** statement is initiated; if *q* is less than or equal to zero, (a given, in that *q* was set to be equal to zero in step 12), then:

a. *ja* is set to be equal to three increments, plus one-tenth of the total RF path distance in increments. The SPLAT version, itm.cpp, defines the argument *ja* to be equal to **long**(3+.1*elev[0]).

b. *jb* is set to be equal to the total number of increments, plus six, less *ja*.. This sets *jb* to be the same distance away from the receive terminal, along the rf path, that *ja* is from the transmit terminal.

```
Line 1433:      if (q<=0.0)
                        {
                        ja=(long)(3.0+0.1*elev[0]);  /* KD2BD added (long) */
                        jb=np-ja+6;
```

19. A **for** loop is initiated, embedded within the **if** statement. The loop operates from *i = ja-1*, until *i=jb*. As the loop cycles, the value of zsys, which was preset to zero when declared, reads and sums up the value of all elevations along the RF path from *ja* to *jb*.

```
Line 1438:      for (i=ja-1; i<jb; ++i)
                        zsys+=elev[i];
```

20. After the **for** loop has completed its cycle, then zsys is divided by the value of the total number of elevation points starting at *ja* and ending at *jb*. This results in *zsys* being equal to the average elevation height between *ja* and *jb*.

```
Line 1441:                              zsys/=(jb-ja+1);
```

:

21. *q* is reset to be equal to *eno*, the atmospheric bending constant.. *eno* varies with elevation above ground; here it is set to equal ens, the refractivity of the atmosphere as it approaches the surface of the earth, a user selected input with a customary preset value of 301.000 N-units. The **if** statement then ends its run.

```
Line 1442:              q=eno;
                }
```

22. *propv.mdvar*, the mode of variability, is preset to be equal to 12. This consists of a combination of modes 10 and 2. The value “10” is added for the point-to-point mode, which causes location variability to be eliminated. [However, this should not be true in version 7, which allows location variability to be set]. The value “2” indicates “Mobile” mode, where reliability is calculated as a combination of time and location variability. Confidence in mode 2 (or 12) is given by the situation variability.

So the setting of mdvar equal to 12, formerly hard-coded into the ITMDLL.cpp, meant that the version 7.0 of ITMDLL.cpp is never intended to be used for broadcast unless the value of mdvar is changed in the source code and the ITMDLL is re-compiled.

Note: Therefore, for broadcast reception prediction use, the ITMDLL has been modified to allow external resetting of the mdvar, using input mode_var = 12 (preset); this is especially critical in light of the new optional subroutine *point_to_pointMDP*, which allows for setting the percentage value for location.

Line 1445: propv.mdvar=mode_var;

23. Subroutine ***qlrps*** is then called with inputs:

a. frq_mhz, the frequency in MHz
b. zsys, the average terrain height along path *ja* to *jb*
c. q, most recently set to be equal to *eno*, the atmospheric bending constant/ atmospheric refractivity
d. pol, polarity of the transmitted and received RF signal
e. eps_dielect, earth's dielectric constant
f. sgm_conductivity, earth's conductivity
g. prop; the array *prop.*

From these inputs, subroutine ***qlrps*** calculates and processes the following data and inserts it into the prop_type structure (prop_type & prop) :

h. *Prop.wn* wave number (=freq. in MHz/47.7 MHz * meters) units in 1/meters
i. *prop.ens* surface refractivity
j. *prop.gme* effective earth curvature
k. *prop.zgnd* surface impedance
l. *prop.zgndreal* real surface impedance (resistance component)
m. *prop.zgndimag* imaginary surface impedance (reactive component)

Line 1446: qlrps(frq_mhz,zsys,q,pol,eps_dielect,sgm_conductivity,prop);

24. Subroutine ***qlrpfl*** is then called with inputs:

a. elev, array *elev (*a.k.a. array *pfl)*
b. propv.klim, the climate variable
c. propv.mdvar, the mode of variability (operating mode)
d. prop, array *prop*
e. propa, array *propa*
f. propv. array *propv*

Subroutine ***qlrpfl*** calls subroutines ***hzns, dlthx*** (which calls ***zlsq1*** and ***qtile***), after which ***qlrpfl*** may call ***zlsq1*** directly, then ends by calling ***lrprop***. Subroutine ***lrprop*** then places the value of *aref*, the value of the reference attenuation, or RF path loss, along the RF path, into array location *prop.aref*;

Line 1447: qlrpfl(elev,propv.klim,propv.mdvar,prop,propa,propv);

25. The free space path loss is then calculated by determining the amount of RF emitted from a point source, or isotropic antenna, at the center-point of a sphere, that would be received by a frequency-tuned antenna embedded in the surface of the sphere at a radius r equal to the RF path distance, all in free (open and well above the surface) space. The formula, for units of:

a. Frequency in MHz, and
b. Distance in kilometers (prop.dist/1000) is:

Free Space Loss, in dB = 32.45 + 20*$\log_{10}$(Distance) + 20.0*$\log_{10}$(Frequency)

Line 1448: fs=32.45+20.0*log10(frq_mhz)+20.0*log10(prop.dist/1000.0);

In steps 22 to 28, the subroutine utilizes string mode (strmode) to prepare printouts of the operating mode for use in a printed report.

26. Once again we press the utility variable *q* into service, setting it to be equal to prop.dist-propa.dla, subtracting the distance to the transmitter horizon (or highest visible obstruction), from the total path distance.

Line 1449: q=prop.dist-propa.dla;

NOTE: The action of q, here used to determine the point where the printout of "Line of Sight Mode" in the report switches to "Single Horizon", and then to "Double Horizon", only controls the printout, not the actual calculation. The actual calculation instruction set is incorporated into, and performed by, subroutine lrprop.

27. An **if** statement is initiated; if the integer value of *q* , which was calculated in step 22, is negative (less than zero), then the path is line-of-sight, and the string prepares to output "Line-Of-Sight Mode" to an output terminal.

```
Line 1451:    if (int(q)<0.0)
                    strcpy(strmode,"Line-Of-Sight Mode");
```

28. An **else** statement follows, so if *q* is greater than or equal to zero:

```
Line 1453:    else
              {
```

29. An **if** statement is embedded within the **else** statement, so if the integer value of *q* is equal to zero, then: the string prepares to output “Single Horizon” to an output terminal.

```
Line 1455:          if (int(q)==0.0)
                    strcpy(strmode,"Single Horizon");
```

30. An **else if** statement pair is embedded within the **else** statement in step 24, so if the integer value of q is greater than zero, then the string prepares to output “Double Horizon” to an output terminal.

```
Line 1458:    else if (int(q)>0.0)
                    strcpy(strmode,"Double Horizon");
```

31. The following **if** statement is also embedded within the **else** statement in step 24, so if:
    a. *q* is not less than zero, and;
    b. *prop.dist* is less than or equal to *propa.dlsa* (i.e., the total path length, prop.dist, is shorter than or equal to: the sum of the two smooth earth horizon distances) or;
    c. prop.dist is less than or equal to propa.dx, the distance beyond which troposcatter mode has dominance, then:
    d. the string prepares to concatenate (append) the phrase “, Diffraction Dominant” to the phrase preloaded in steps 25 or 26 above.

```
Line 1461:    if (prop.dist<=propa.dlsa || prop.dist <= propa.dx)
                    strcat(strmode,", Diffraction Dominant");
```

32. An **else if** statement pair follows; they are also embedded within the **else** statement in step 24, and counteroffer the **if** statement in step 27. So if prop.dist is greater than propa.dx, the distance beyond which troposcatter mode has dominance, then the string prepares to concatenate (append) the phrase “, Troposcatter Dominant” to the phrase preloaded in steps 25 or 26 above.

```
Line 1464: else if (prop.dist>propa.dx)
                    strcat(strmode, ", Troposcatter Dominant");
}
```

33. The subroutine ***avar*** is called with inputs:
    a. *zr*, the time reliability, (50% of the time) as a percentage decimal value between 0.01 and 0 .99, to be calculated.
    b. 0.0, the location percentage (at 00% of locations) to be calculated. (Location percentage is disabled for mdvar= 2 or 12).
    c. *zc*, the confidence, as a percentage decimal value between 0.01 and 0.99, to be calculated.

d. array *prop*,
e. array *propv*

Subroutine ***avar*** returns *avarv*, a value representing the reference attenuation *aref* adjusted for the additional loss to be included as a result of calculating for statistical variation as a percentage of time, location, and confidence as authorized by the setting of the mdvar code. The value of dbloss (a.k.a. aref, the reference attenuation in dB) is set to be equal to the value of *avarv* in dB returned by subroutine ***avar*** and the value of the free space attenuation.

```
Line 1468:    dbloss=avar(zr,0.0,zc,prop,propv)+fs;
```

34. The value of the output-accessible argument *errnum* is set to be equal to the value of *prop.kwx*, the error indicator.

```
Line 1469     errnum=prop.kwx;
        }
```

Subroutine ***point_to_point*** ends. The user-supplied wrap-around input-output program continues. For a single terrain profile, the wrap-around input-output program may execute subroutine ***point_to_point*** hundreds of times to calculate the RF signal loss for each terrain point (except the transmitter site) calculated along the terrain path. For a modern point-to-point all-points area mapping of signal loss, the wrap-around input-output program may run ***point_to_point*** thousands, hundreds of thousands, or even millions of times to calculate the RF signal loss for each terrain point. So it is important to keep the code “lean and tight” for purposes of speed of execution.

SUBROUTINE QLRPFL: A functional explanation, by Sid Shumate.

Last Revised: Sept. 26, 2008 to include receiver approach angle calculation.

Quick Longley-Rice Profile

Note: Used with point-to-point mode only. Called by point_to_point after calling ***qlrps***. Calls ***hzns, dlthx*** (which calls ***zlsq1*** and ***qtile***), after which ***qlrpfl*** may call ***zlsq1*** directly, then ends by calling ***lrprop***. One may then call ***avar*** for quantiles, if desired.

Derived from the Irregular Terrain Model description by George Hufford, 2002, (ITMD), compared to the ITM.cpp prepared by J. D. McDonald and John Magliacane for compilation on unix and linux systems. "Line" numbers refer to the ITM.cpp as line numbered by Bloodshed Software's DevC++ print function. "Alg" numbers refer to the algorithm formulas in "The ITS Irregular Terrain Model, version 1.22, the Algorithm" by G. A. Hufford, 1995. "ITS67" numbers refer to the algorithm formulas in "ESSA Technical Report ERL 79-ITS 67, Prediction of Tropospheric Radio Transmission Over Irregular Terrain, A Computer Method – 1968" by A.G.Longley and P.L.Rice.

Please note that the ***qlrpfl*** subroutine, and the ***dlthx, hzns*** and ***zlsq1*** subroutines that are called during ***qlrpfl,*** were intended to be experimental early versions, but are still in use today with few modifications or corrections. George Hufford, in The ITM Manual states:

> "It should be noted that the original ITM is silent on many of the details for defining some of the path parameters. This is particularly true of the effective heights HE, and, to some lesser degree, of the terrain irregularity parameter DH. The effective height, for example is defined as the height above the "effective reflecting plane," and in the past the investigator has been urged to use his own best judgement as to where that plane should be placed. The subroutine QLRPFL, in trying to automate the definition of all parameters, has been forced to define explicitly all missing details. It has done this in a way that seems reasonable and in full accord with the intent of the model. One should not, however, conclude that these efforts are completed. Hopefully, better results are obtainable."

From ITMD Section 43:

Call inputs:

pfl terrain elevation profile array, starting at tx, ending at rcvr, following great circle path, with:
pfl[0] =*enp*, the number of increments
pfl[1] = *xi*, distance per increment
pfl[2] = *z(0)*, the transmitter tower base AMSL, or elevation height
pfl[[np+2] = *z(np)*, the receive location AMSL, the last elevation.

klimx	a.k.a. propv.klim,	the climate code
mdvarx	a.k.a. propv.mdvar,	the mode of variability; preset to 12.0 in the ***point_to_point*** subroutine, and readjusted during ***qlrpfl***;

defines private, or local, arguments:

np	number of points
j	terminal, either 0 (1, or transmit site) or 1 (2, or receive site)
x1[0]	position on terrain path a short distance from transmitter site, in meters.
x1[1]	position on terrain path a short distance from receive site, in meters.
q	Δh(s), delta h adjusted for the path length
za	elevation value in meters of the average terrain height at the transmit site
zb	elevation value in meters of the average terrain height at the receive site
temp	temporary value holding variable used at line 1326 and line 1327.
rad	receive approach path start location, in meters from the transmitter site.
rae1	receive approach elevation at "rad" in meters.
rae2	receive approach elevation at receiver in meters.

This subroutine:

1. Uses *pfl [0]*, number of points, and *pfl [1]*, increment distance, to calculate path length *prop.dist*.

Line 1302: prop.dist=pfl[0] * pfl[1];

2. Defines *np*, number of points, to be equal to the value stored in *pfl [0]*.

Line 1303: np=(int)pfl (0);

3. Calls subroutine ***hzns***, forwarding as input, arrays *pfl* and *prop*.

Line 1304: hzns(pfl,prop); See separate description for subroutine ***hzns.***

4. ***hzns*** returns:
    a. *prop.the[0]* horizon elevation angle as seen from tx antenna center; specified as vertical units of increase or decrease per horizontal distance.
    b. *prop.the[1]* horizon elevation angle as seen from rcvr antenna center; specified as vertical units of increase or decrease per horizontal distance.
    c. *prop.dl[0]* distance from transmitter tower base to horizon
    d. *prop.dl[1]* distance from receive antenna ground point to horizon

5. A **for** loop is initiated to determine the values of x1 and x2. This is a short loop, 2 cycles, from j=0 to j<2, i.e. for j=0 and j=1. j refers to the terminals, j=0 represents the transmitter site terminal, j=1 represents the receive site terminal.

Line 1306: for (j=0; j<2; j++)

6. The loop performs the following operations:

a. On the first pass, sets the value of xl[0] (a.k.a. x1, or x-one,) equal to the lesser of (15 times the height of the transmit antenna, or 1/10 of the distance from tx tower base to horizon.

Line 1307: xl[j]=mymin(15.0*prop.hg[j],0.1*prop.dl[j]

b. On the second pass, sets the value of x1[1] (a.k.a. x2) equal to the lesser of 15 times the height of the receive antenna, or 1/10 of the distance from the receive site to the horizon.

7. The value of xl[1] is then set equal to the path distance less the existing value of xl[1]. This makes it equal to the distance from the transmitter site to the point near the receiver site.

Line 1309: xl[1]=prop.dist-xl[1]

8. The value of prop.dh, the terrain irregularity parameter (a.k.a. delta h) is then determined by calling subroutine ***dlthx***(pfl,xl[0],xl[1].

Line 1310: prop.dh=dlthx(pfl,xl[0],xl[1]);

Dlthx calls ***mymin, mymax, assert, zlsql*** and ***qtile.***

dlthx returns dlthxv, the Δh (a.k.a. delta h or *dh*) terrain irregularity parameter, which is stored in prop.dh

9. Next, an **if** statement is initiated; the first task of this **if** statement is to determine if the path is a line-of-sight path, or a trans-horizon path. If the sum of prop.dl[0} and prop.dl[1], the horizon distances from the transmitter to the horizon and the receiver to the horizon, is less than 1.5 times the total path distance prop.dist, then the path is determined to be a line-of-sight path.

For example, if the transmit to receive path distance is 10,000 meters (10 km), then the combined total of the distance to the horizon from the transmit site, and of the distance to the horizon from the receive site, must equal or be greater than 150% of 10 km, or 15,000 meters (15 km), overlapping each other by an average of 1/3 of each, for the RF path to be determined to be a line-of-sight path.

As a second example, if there is a single obstacle, then the combination of the distance to the horizon (the obstacle) from the transmit site and of the distance to

the horizon (the obstacle) from the receive site, would approximately equal the combination of the distance to the horizon (obstacle) from the transmit site and of the distance to the horizon (obstacle) from the receive site, and would therefore not meet or exceed the 150% of total path length overlap requirement necessary to determined to be a line-of-sight path. The default determination made would be that the RF path is a trans-horizon path, and the computer program would jump to the **else** statement on Line 1343.

Line 1312: **if** (prop.dl[0] +prop.dl[1]>1.5*prop.dist)

10. If the path is a line-of-sight path, the **if** statement then continues:

a. subroutine ***zlsql*** is called with inputs (*pfl, x1[0],xl[1*]), in order to calculate an average terrain line between points x1[0] and xl[1], and determine the average elevation height on that line at the location of x1[0] and x1[1] ;

b. ***zlsq1*** then returns:

za = z0, the elevation value of the average terrain line at the transmitter site.

zb = z1, the elevation value of the average terrain line at the receive site.

c. The effective height of the transmit site, *he(0),* is set to be equal to *prop.hg(0) + pfl(2) – za*, but only if *pfl(2) > za*. If *pfl(2)* is not > *za,* then *he(0)* is set to be equal to *prop.hg(0).*

Therefore *prop.he(0),* the effective height of the transmit site, is set to be equal to *prop.hg(0)*; and if the existing ground height of the transmit site, *pfl(2),* is above the average elevation height at the transmit site, *za,* (established by ***zlsq1***), then the difference in height between the average transmit site elevation height and the ground height is also added to prop.he(0).

d. the effective height of the receive site, *prop.he(1),* is set to be equal to *prop.hg(1) + pfl(np+2) – zb*, but only if *pfl(np+2) > zb*. If *pfl(np+2)* is not > *zb,* then *prop.he(1)* is set to be equal to *prop.hg(1).*

Therefore prop.he(1) is set to be equal to prop.hg(1), and if the existing ground height of the receive site, *pfl(np+2),* is above the average elevation height at the receive site, *zb,* the difference in height between the average receive site elevation height and the ground height is also added to prop.he(1).

NOTE: In October of 2004, Hammett & Edison, a well respected and highly regarded communications engineering consulting firm in San Francisco, CA, submitted comments to the Federal Communications Commission (FCC) in CS Docket 98-201, regarding the

use of Longley-Rice in calculating Grade B TV Signal Coverage. They stated in paragraph 20:

"This ongoing work has convinced us that the implementation of the L-R model is even more flawed than had been originally suspected. For example is has come to light that the OET-69 software calculates the depression angle to a calculation point using the sources height above ground, not its height above sea level. This coding mistake by itself will introduce errors of perhaps 10-20 dB in the calculation results."

Therefore, the calculation of the elevation height value in prop.hg(0) at the transmit site and the elevation height value in prop.hg(0), at the receive site, must be reviewed to determine if prop.hg(0) and prop.hg(1) are derived from the height above ground level, or are derived correctly from height above sea level. If from ground level, it is eligible for correction; One solution would be to add pfl(2), the ground height, to he(0) on line 1315, and add pfl(np+2) to he(1) on line 1316. This however, would make it more difficult to insert separate receive site elevations from a second, ground height database, in the input to the point_to_point subroutine call. The better solution would be to replace the input value of prop.hg(0) supplied in the point_to_point subroutine call, with the radiation center height above mean sea level (RCAMSL), in meters, of the transmit site antenna, and to replace the value of prop.hg(1) with the reception center height above mean sea level of the receive antenna. However, all other subroutines must be checked for necessary adjustments and corrections if they take input, directly or indirectly, from prop.hg(0) or prop hg(1).

First, however, we continue to describe the function of the software in its current form:

```
Line 1314:    zlsq1(zpfl,x1[0],x1[1],za,zb);
Line 1315:    prop.he[0]=prop.hg[0]+FORTRAN_DIM(pfl[2],za);
Line 1316:    prop.he[1]=prop.hg[1]+FORTRAN_DIM(pfl[np+2],zb);
```

11. A **for** loop, with two loops (j=0 and j=1), within the above **for** loop (started in Step 5.) is initiated here, to determine or re-determine the values of prop.dl[0], the distance from the transmitter site to the horizon, and prop dl[1], the distance from the receive site to the horizon, for a line of sight analysis.

NOTE: The Environmental Science Services Administration (ESSA) Technical Report ERL 79-ITS 67, "Prediction of Tropospheric Radio Transmission Loss Over Irregular Terrain, A Computer Method – 1968" by A.G. Longley and P. L. Rice, states on page 12, starting with paragraph 2:

> "When individual path profiles are not available, median values of the horizon distances $d_{L1,2}$ are estimated as functions of the median effective antenna heights h_{e1} and h_{e2} determined above, the terrain irregularity factor Δh, and the smooth-earth horizon distances D_{Ls1} and D_{Ls2}. The smooth earth distance from each antenna to its horizon over a <u>smooth</u> <u>earth</u> is defined as:

$$D_{Ls1,2} = (.002 * a * h_{e1,2})^{.5} \text{ in km.} \qquad \text{ITS67 (5a)}$$

where the effective antenna heights $h_{e1,2}$ are in meters and the effective earth's radius a is in kilometers, as defined by (1). The sum of the smooth-earth horizon distance is

$$D_{Ls} = D_{Ls1} + D_{Ls2}, \text{ in km.} \qquad \text{ITS67 (5b)}$$

Median values of horizon distances over <u>irregular</u> <u>terrain</u> are estimated as

$$D_{L1,2} = D_{Ls1,2} \exp(-.07 (\Delta h/h_e)^{.5} \text{ in km,} \qquad \text{ITS67 (5c)}$$

where

$$h_e = h_{e1,2} \text{ for } h_{e1,2} \geq 5 \text{ meters, or } 5 \text{ meters otherwise.}$$

The total distance, $d_{L,}$ between the antennas and their horizons is

$$d_L = d_{L1} + d_{L2}, \text{ in km".} \qquad \text{ITS67 (5d)}$$

To use these formulas in this subroutine, we convert from km to meters:

$$D_{Ls1,2} = (.002 * a * h_{e1,2})^{.5} \text{ in km.} \qquad \text{ITS67 (5a)}$$

becomes:

$$D_{Ls1,2} = (2 * h_{e1,2} * a)^{.5} \text{ in meters}$$

The effective earth's radius *a*, in kilometers, is defined by ITS67 (1). The term *gma*, representing earth's actual curvature, is a simplified approximation, as it treats the earth as a sphere, not a spheroid. It is established in subroutine ***qlrps*** to be equal to 157e–9 1/meter, and used in step 4 of ***qlrps*** to calculate the effective earth curvature, *gme*, which is then stored in array *prop* at *prop.gme.*

So what is the relationship between *a* and *gma?* One might reasonably assume that earth's actual curvature would be defined as the change per meter of circumference of the earth. If the actual earth's radius is r, then the earth's circumference is:

$$c_e = 2 * (PI) * r$$

and the actual earth's curvature might be defined, per meter, by 1divided by the circumference;

$$gma = 1/ c_e = 1/(2 * PI * r)$$

So for an actual earth radius of *r* = 6,370,000 meters, *gma* would be = *.0000000249 or 249e-10;* but it is not. T*he established value for gma is 157e9, equal to 1/6,370,000 meters.* Therefore, the actual relationship between *a* and *gma* is:

$$gma = 1/(\text{actual earth's radius, in meters})$$

The same relationship therefore applies between the effective earth's radius and the effective earth's curvature:

$$gme = 1/a, \text{ in units of 1/meters, and } a = 1/gme.$$

So then, $D_{Ls1,2} = (2 * h_{e1,2} * a)^{.5}$ m. becomes $D_{Ls1,2} = (2 * h_{e1,2} / gme)^{.5}$ m.

In converting from km to meters:

$$D_{L1,2} = D_{Ls1,2} \exp(-.07 (\Delta h/h_e)^{.5} \text{ in km,} \qquad \text{ITS67 (5c)}$$

Becomes:

$$D_{L1,2} = D_{Ls1,2} \exp(-.07 (\Delta h/h_e)^{.5} \text{ in meters,}$$

Substituting the formula for $D_{Ls1,2}$ derived above:

$$D_{L1,2} = ((2 * h_{e1,2} / gme) \exp(-.07 (\Delta h/h_e)^{.5})^{.5} \text{ in meters,}$$

where

$$h_e = h_{e1,2} \text{ for } h_{e1,2} \geq 5 \text{ meters, or } 5 \text{ meters otherwise.}$$

Which we can restate in the notation primarily used in this text as:

```
Prop.dl[j] =  ((2*prop.he[j]/prop.gme)^( –.07*(prop.dh/mymax(prop.he[j],5)^.5)
```

The formula for the total distance between the antennas and their horizons is the same as long as all units are in either km or meters:

$$d_L = d_{L1} + d_{L2}, \text{ in meters;}$$

A problem with this subroutine, with respect to both line-of-sight and trans-horizon paths, is that the Irregular Terrain Model code, ITMDLL.cpp, still uses a formula derived from ITS67 (5c) for the point_to_point mode, despite the fact that Anita Longley and Phil Rice flatly stated, at the beginning of paragraph 2, page 12 of the ESSA Technical Report ERL 79-ITS 67 quoted above, that it was to be used "only when individual path profiles are not available".

The point_to_point mode utilizes an individual path profile, input as array pfl. The use of the NTIA-released ITMDLL.cpp c++ software requires the use of additional code, a "wrap-around" package (an example being the open source program SPLAT, or its experimental, advanced concept development cousin SPLAT with PLOP) that is compiled with the irregular terrain model windows-compatible software (or in the case of SPLAT, the linux-friendly itm.cpp) that prepares the input, including deriving the pfl array from the raw elevation database data, and processes the output from the core itm.cpp subroutines,

The pfl array, especially when used in multiple runs to analyze signal loss and/or reception over a large area, usually contains far more elevation data, extending along the great circle path through the receive site, than is required to derive the distance to the actual horizon. The first two values stored in the array pfl, as sent to the point_to_point subroutine call, are pfl[0], the value of the number of intervals, and pfl[1], the value of the width, in meters, of an individual interval. The value of pfl[0] is set to indicate the number of intervals between the transmit site, and a receive site to be considered, and indicates the minimum number of elevation data values stored in the array pfl. The value of pfl[0] does not necessarily indicate the maximum number of elevation data values stored in the array pfl. The pfl array, especially when used in multiple runs to analyze signal loss and/or reception over a large area, usually contains far more elevation data, extending along the great circle path through the receive site, than is required to derive the distance to the receive site in question, as the wrap-around software will store elevation values in the pfl array extending out several tens of kilometers, in anticipation of repeating the point_to_point call to derive loss values at hundreds of receive locations along the rf path being studied. Therefore, the elevation data stored in the pfl array usually, if not always, represents elevation data along the great circle path extending far beyond any value of the horizon for a ground-mounted reception site. In the few cases where this data is not available, i.e. where the database from which the pfl array was derived does not extend to the horizon, this methodology could remain available as a default, and an additional kwx flag could be generated, to indicate that the pfl array does not extend to the radio horizon and that the distances to the radio horizon are estimated.

Therefore, with today's comprehensive elevation databases, including the SRTM and NED elevation data, there is little or no call to continue to use this approximation instead of deriving a more accurate result, where available, for the distances to the actual horizons, from the elevation database.

Therefore, this subroutine's code is eligible for review and revision in order to make today's ITM computer programs operate more in accordance with Longley and Rice's original concept, procedure, and instructions, by deriving the actual distance to the radio horizons from the transmit and receive sites, from additional terrain profile data in the pfl array.

12. The value of prop.dl[0] is estimated as a median value of a horizontal distance over irregular terrain using the formula:

    Prop.dl[0] = ((2*prop.he[0]/prop.gme)^(.07*(prop.dh/mymax(prop.he[0],5)^.5)
    ITS67 (5c)

    Where:

    prop.he[0] is the effective height of the transmitter site (from step 10)

prop.he[1] is the effective height of the receive site (from step 10)
prop.gme is the effective earth's curvature, input with the point_to_point subroutine call
prop.dh is the terrain irregularity parameter, Δh, or delta h, obtained from the ***dlthx*** subroutine call on line 1310.

Similarly, the value of prop dl[1] is estimated as:

Prop.dl[1] = ((2*prop.he[1]/prop.gme)^(–.07*(prop.dh/mymax(prop.he[1],5)^.5)
ITS67 (5c)

Line 1318: for (j=0; j<2; j++)
prop.dl[j]=
sqrt(2.0*prop.he[j]/prop.gme)*exp(–0.07*sqrt(prop.dh/mymax(prop.he[j],5.0)));

13. The variable *q* is then set to be equal to the combined total of the distance to the horizon from the transmit site, dl[0], and the distance to the horizon from the receive site dl[1], for a line of sight analysis.

Line 1321: q=prop.dl[0]+prop.dl[1];

14. An **if** loop is initiated; if q, as defined in step 12 above, is less than the total path distance from the transmit site to the receive site, then the variable *temp* is set to be equal to the total path distance, *prop.dist*, divided by *q*, and the value of *q* is then reset to be equal to the square of the value stored in *temp.*

A rare comment in the itm.cpp c++ code, referring to the earlier FORTRAN version, states: q=pow(prop.dist/q,2.0);

Line 1323:
```
if (q<=prop.dist)
{
        /* q=pow(prop.dist/q,2.0); */
        temp=prop.dist/q;
        q=temp*temp;
```

15. A **for** loop is initiated, again with two loops, j=0 and j=1, for the transmitting and receive site locations. This for loop:

a. changes the value of prop.he[0] to be equal to the existing value of prop.he[0], the effective height of the transmit antenna, multiplied by the value of q, and;

b. changes the value of prop.he[1] to be equal to the existing value of prop.he[1], the effective height of the receive antenna, multiplied by the value of q.
c. again resets the value of prop.dl[0], which represents the distance from the transmit site to the horizon, to be equal to:

dl[0]= sqrt(2.0*prop.he[0]/prop.gme)*exp(–0.07*sqrt(prop.dh/mymax(prop.he[0],5.0)));
ITS67 (5c)

Where, as in step 11 above:
prop.he[0] is the effective height of the transmitter site (from step 10)
prop.he[1] is the effective height of the receive site (from step 10)
prop.gme is the effective earth's curvature, input with the point_to_point subroutine call
prop.dh is the terrain irregularity parameter, Δh, or delta h, obtained from the ***dlthx*** subroutine call on line 1310.

d. And also again resets the value of prop.dl[1], which represents the distance from the receive site to the horizon, to be equal to:

dl[1]= sqrt(2.0*prop.he[1]/prop.gme)*exp(–0.07*sqrt(prop.dh/mymax(prop.he[1],5.0)));
ITS67 (5c)

Line 1329:
```
for (j=0; j<2; j++)
        {
   prop.he[j]*=q;
   prop.dl[j]=sqrt(2.0*prop.he[j]/prop.gme)*exp(–
0.07*sqrt(prop.dh/mymax(prop.he[j],5.0)));
        }
}
```

16. A **for** loop is initiated, again with two loops, j=0 and j=1, for the transmitting and receive site locations. This for loop determines the value of prop.the[0] and prop.the[1], the "theta" angle:

a. resets the value of *q* to be equal to 2 * prop.he[0]/prop.gme
b. sets the value of prop.the[1] equal to:
prop.the[0]=(0.65*prop.dh*(q/prop.dl[0] –1.0) –2.0*prop.he[0])/q;
c. resets the value of *q* to be equal to 2 * prop.he[1]/prop.gme
d. sets the value of prop.the[1] equal to:
prop.the[1]=(0.65*prop.dh*(q/prop.dl[1] –1.0) –2.0*prop.he[1])/q;

Line 1336:
```
        for (j=0; j<2; j++)
{
        q=sqrt(2.0*prop.he[j]/prop.gme);
        prop.the[j]=(0.65*prop.dh*(q/prop.dl[j] –1.0) –2.0*prop.he[j])/q;
}
}
```

17. Two new steps added for the ITWOM, computes the receiver approach (theta receive approach, or θ_{ra}), and the near receiver approach (theta near receiver, or θ_{nr}) angles.
    a. The receive approach path start location, *rad*, defined as the distance from the transmitter site to the receive approach path start location, is set to be equal to the maximum of the path distance, prop.dist, or prop.dist-450. This defines the receiver approach path to be the 450 meters (or path distance, if less) leading up to the receiver.
    b. The subroutine calls ***zlsq1*** with inputs (*pfl,rad,prop.dist*);
       Where *plf* is the elevation array;
       rad is the distance from the transmitter to a location 450 meters away from of the receive site, or zero (the transmitter site) if the receive site is less than 450 meters from the transmitter.
       prop.dist is the path distance from the transmitter site to the receiver site.

       Subroutine ***zlsq1*** calculates an average terrain line between *rad* and the receive site, and determines the average elevation height on that line at the locations *rad* and the receive site.

       zlsq1 then returns:
       rae1 = z0, the elevation value in meters of the receiver approach path average terrain line at *rad*, the receive approach path start location.
       rae2= z1, the elevation value in meters of the average terrain line at the receive site.
    c. The slope is then recovered and converted to an angle, θ_{ra}, theta receive approach, the receive approach angle in radians, to be stored in prop array argument *prop.thera*:

       $$prop.thera = \arctan((rae2 - rae1)/ pfl[1]);$$

       where *pfl[1]* is the interval width in meters.

    d. The angle θ_{nr}, theta near receiver, the slope angle of a line between the last terrain point elevation height before the receiver and the receiver terrain height, in radians, is computed from the respective terrain heights and the interval width, and stored in prop array argument *prop.thenr*:

       $$prop.thenr = \arctan((pfl[np+2]-pfl[np+1])/ pfl[1]);$$

       where *pfl[np+2]* is the elevation height of the receiver
       pfl[np+2] is the elevation height of the terrain database location one interval before the receiver.
       pfl[1] is the interval width in meters.

Line: rad=mymax(0.0,prop.dist-450.0);
zlsq1(pfl,rad,prop.dist,rae1,rae2);
prop.thera=atan((rae2-rae1)/pfl[1]);
prop.thenr=atan((pfl[np+2]-pfl[np+1])/pfl[1]);

18. If the path was determined to be line-of-sight, the program then ignores the **else** statement below, and proceeds to step 18. If the path determination defaulted to trans-horizon, the program proceeds to execute the **else** statement.

a. For a trans-horizon path, the **else** statement calls ***zlsq1*** with inputs (*pfl,xl[0]*,0.9**prop.dl[0]*);
Where *plf* is the elevation array;
xl[0] is the transmit site elevation, and
the term 0.9**prop.dl[0]* specifies a location that is at a point 9/10th of the distance from the transmit site toward the transmit site horizon.

Subroutine ***zlsq1*** calculates an average terrain line between points x1[0] and the point represented by (0.9**prop.dl[0]*), and determines the average elevation height on that line at the locations x1[0] and (0.9**prop.dl[0]*);

zlsq1 then returns:

za = z0, the elevation value of the average terrain line at the transmitter site.
q= z1, the elevation value of the average terrain line at the point 9/10th of the distance from the transmit site toward the transmit site horizon.

b. The subroutine ***zlsq1*** is called with inputs (*pfl, prop.dist*–0.9**prop.dl[1],xl[1]*);
Where: *plf* is the elevation array;
the term (*prop.dist*–0.9**prop.dl[1]*) specifies a location that is at a point 9/10th of the distance from the receive site toward the receive site horizon, and;
xl[1] is the receive site elevation

Subroutine ***zlsq1*** calculates an average terrain line between the point represented by (*prop.dist*–0.9**prop.dl[1]*), and the point represented by x1[1], and determines the average elevation height on that line at those two points;

zlsq1 then returns:

q = z0, the elevation value of the average terrain line at a point 9/10th of the distance from the receive site toward the receive site horizon.
zb = z1, the elevation value of the average terrain line at the receive site.

c. The effective height of the transmit site, *he(0)*, is set to be equal to *prop.hg(0) + pfl(2) – za*, but only if *pfl(2) > za*. If *pfl(2)* is not > *za*, then *he(0)* is set to be equal to *prop.hg(0).*

Therefore *prop.he(0)*, the effective height of the transmit site, is set to be equal to *prop.hg(0)*; and if the existing ground height of the transmit site, *pfl(2)*, is above the average elevation height at the transmit site, *za*, (established by ***zlsq1***), then the difference in height between the average transmit site elevation height and the ground height is also added to prop.he(0).

d. the effective height of the receive site, *prop.he(1)*, is set to be equal to *prop.hg(1) + pfl(np+2) – zb*, but only if *pfl(np+2) > zb*. If *pfl(np+2)* is not > *zb*, then *prop.he(1)* is set to be equal to *prop.hg(1).*

Therefore prop.he(1) is set to be equal to prop.hg(1), and if the existing ground height of the receive site, *pfl(np+2)*, is above the average elevation height at the receive site, *zb*, the difference in height between the average receive site elevation height and the ground height is also added to prop.he(1).

Note: The procedures in steps 15 and 16 do for a trans-horizon path what the procedure in step 10 does for a line-of-sight path. Therefore, the note in step 10 about the October, 2004 comments of Hammett & Edison to the FCC, regarding an error in the code, also applies to the code in steps 15 and 16.

```
else
{
        z1sq1(pfl,xl[0],0.9*prop.dl[0],za,q);
        z1sq1(pfl,prop.dist-0.9*prop.dl[1],xl[1],q,zb);
        prop.he[0]=prop.hg[0]+FORTRAN_DIM(pfl[2],za);
        prop.he[1]=prop.hg[1]+FORTRAN_DIM(pfl[np+2],zb);
}
```

19. The value of prop.mdp, the mode of the propagation model, is set to be equal to – 1, indicating point_to_point mode, and the value of propv.lvar, the level to which coefficients in AVAR must be redefined, is set to be equal to the greater value of either propv.lvar or 3.

```
Line:     prop.mdp=-1;
        propv.lvar=mymax(propv.lvar,3);
```

20. An **if** statement is initiated, stating that if mdvarx, a variable representing the mode of variability, is greater than zero, (zero representing the single message mode), then the value of prov.mdvar is set equal to the value of mdvarx, and the value of propv.lvar is set to the be equal to the greater value of propv.lvar, or 4.

```
Line:           if (mdvarx>=0)
        {
                propv.mdvar=mdvarx;
                propv.lvar=mymax(propv.lvar,4);
        }
```

21. An **if** statement is initiated, stating that if klimx, the climate variable, is greater than zero, the value of propv.klim, the climate code, is set to be equal to the value of klimx, and the value of propv.lvar, which may have been set in step 18 above, is reset to be equal to 5.

```
Line:       if (klimx>0)
        {
                propv.klim=klimx;
                propv.lvar=5;
        }
```

Note: for more information re: lvar, mdvar, and klimx, see “A manual for ITM, “Irregular Terrain Model”, available at http://flattop.its.bldrdoc.gov/itm/itm_man.txt.

22. Finally, we arrive at the climax; calculating the Longley-Rice path loss. The subroutine then calls ***lrprop*** with inputs 0.0, array prop, and array propa;

```
Line:           lrprop(0.0,prop,propa);
}
```

lrprop returns the reference attenuation, *aref*. This is the “answer”, the amount of loss, in db, in the rf signal level between the transmitter and the receiver.

SUBROUTINE QLRPFL2: A functional explanation, by Sid Shumate.

Last Revised: August 15, to catch up on documentation changes.
Aug. 1, 2010 to add prop.rch[0], prop.rch[2].
Previous known change as qlrpfl, Sept. 26, 2008 to include receiver approach angle calculation.

Quick Longley-Rice Profile

Note: Used with point-to-point mode only. Called by point_to_point after calling ***qlrps***. Calls ***hzns, dlthx*** (which calls ***zlsq1*** and ***qtile***), after which ***qlrpfl2*** may call ***zlsq1*** directly, then ends by calling ***lrprop***. One may then call ***avar*** for quantiles, if desired.

Derived from the Irregular Terrain Model description by George Hufford, 2002, (ITMD), compared to the ITM.cpp prepared by J. D. McDonald and John Magliacane for compilation on unix and linux systems. "Line" numbers refer to the ITM.cpp as line numbered by Bloodshed Software's DevC++ print function. "Alg" numbers refer to the algorithm formulas in "The ITS Irregular Terrain Model, version 1.22, the Algorithm" by G. A. Hufford, 1995. "ITS67" numbers refer to the algorithm formulas in "ESSA Technical Report ERL 79-ITS 67, Prediction of Tropospheric Radio Transmission Over Irregular Terrain, A Computer Method – 1968" by A.G.Longley and P.L.Rice.

Please note that the ***qlrpfl*** subroutine, and the ***dlthx, hzns*** and ***zlsq1*** subroutines that are called during ***qlrpfl,*** were intended to be experimental early versions, but are still in use today with few modifications or corrections. George Hufford, in The ITM Manual states:

> "It should be noted that the original ITM is silent on many of the details for defining some of the path parameters. This is particularly true of the effective heights HE, and, to some lesser degree, of the terrain irregularity parameter DH. The effective height, for example is defined as the height above the "effective reflecting plane," and in the past the investigator has been urged to use his own best judgement as to where that plane should be placed. The subroutine QLRPFL, in trying to automate the definition of all parameters, has been forced to define explicitly all missing details. It has done this in a way that seems reasonable and in full accord with the intent of the model. One should not, however, conclude that these efforts are completed. Hopefully, better results are obtainable."

From ITMD Section 43:

Call inputs:

pfl terrain elevation profile array, starting at tx, ending at rcvr, following great circle path, with:

pfl[0] =*enp*, the number of increments
pfl[1] = *xi*, distance per increment
pfl[2] = *z(0)*, the transmitter tower base AMSL, or elevation height
pfl[[np+2] = *z(np)*, the receive location AMSL, the last elevation.

klimx	a.k.a. propv.klim,	the climate code
mdvarx	a.k.a. propv.mdvar,	the mode of variability; preset to 12.0 in the ***point_to_point*** subroutine, and readjusted during ***qlrpfl2***;

defines private, or local, arguments:

np	number of points
j	terminal, either 0 (1, or transmit site) or 1 (2, or receive site)
x1[0]	position on terrain path a short distance from transmitter site, in meters.
x1[1]	position on terrain path a short distance from receive site, in meters.
q	Δh(s), delta h adjusted for the path length
za	elevation value in meters of the average terrain height at the transmit site
zb	elevation value in meters of the average terrain height at the receive site
temp	temporary value holding variable used at line 1326 and line 1327.
rad	receive approach path start location, in meters from the transmitter site.
rae1	receive approach elevation at "rad" in meters.
rae2	receive approach elevation at receiver in meters.

This subroutine:

1. Uses *pfl [0]*, number of points, and *pfl [1]*, increment distance, to calculate path length *prop.dist*.

Line 1302: prop.dist=pfl[0] * pfl[1];

2. Defines *np*, number of points, to be equal to the value stored in *pfl [0]*.

Line 1303: np=(int)pfl (0);

3. Calls subroutine ***hzns***, forwarding as input, arrays *pfl* and *prop.*

Line 1304: hzns(pfl,prop); See separate description for subroutine ***hzns.***

4. ***hzns*** returns:
    a. *prop.the[0]* horizon elevation angle as seen from tx antenna center; specified as vertical units of increase or decrease per horizontal distance.
    b. *prop.the[1]* horizon elevation angle as seen from rcvr antenna center; specified as vertical units of increase or decrease per horizontal distance.
    c. *prop.dl[0]* distance from transmitter tower base to horizon
    d. *prop.dl[1]* distance from receive antenna ground point to horizon

5. Argument *dlb* is set to be equal to the sum of the horizon path lengths, prop.dl[0] and prop.dl[1].

Line (new): dlb=prop.dl[0]+prop.dl[1];;

6.

7. For the use of the ***saalos*** subroutine on paths that are past an obstruction, we need actual transmitter and receiver antenna heights above mean sea level. So we establish a new set of variables, prop.rch[ ], to hold the actual transmitter and receiver radiation center heights above ground level (RC-AMSL).

   a. *Prop.rch[0]*, the actual transmit antenna radiation center height above mean seal level, RC-AMSL, is set to be equal to the transmitter antenna height above ground plus the transmitter ground height above mean sea level (AMSL).
   b. Prop.rch[1], the actual receive antenna radiation center height above mean seal level, RC-AMSL, is set to be equal to the transmitter antenna height above ground plus the transmitter ground height above mean sea level (AMSL).

Line (new): prop.rch[0]=prop.hg[0]+pfl[2];
prop.rch[1]=prop.hg[1]+pfl[np+2];

A two cycle **for** loop is initiated to determine the values of x1 and x2.

   c. In the first run, the value of xl[0] (a.k.a. x1, or x-one,) is set to be equal to the lesser of:
      i. 15 times the height of the transmit antenna above ground level, or:
      ii. one-tenth of the transmitter to horizon obstruction path length, the ground distance from the tx tower site to the horizon obstacle.
   d. In the second run, the value of x1[1] (a.k.a. x2) is first set to be equal to the lesser of:
      i. 15 times the height of the receive antenna above ground level, or:
      ii. 1/10 of the distance from the receive site to the horizon.
   e. After the loop is complete, the value of xl[1] is then reset to be equal to the path distance less the existing value of x2 determined in step a. This makes it equal to the distance from the transmitter site to the point near the receiver site.

Line 1307:
```
for(j=0; j<2; j++)
        xl[j]=mymin(15.0*prop.hg[j],0.1*prop.dl[j]);

xl[1]=prop.dist-xl[1]
```

8. The value of prop.dh, the terrain irregularity parameter (a.k.a. delta h) is then determined by calling subroutine ***dlthx2***(pfl,xl[0],xl[1].

Line 1310: prop.dh=dlthx2(pfl,xl[0],xl[1]);

dlthx2 calls ***mymin, mymax, assert, zlsql*** and ***qtile.***

dlthx2 returns dlthxv2, the Δh (a.k.a. delta h or *dh*) terrain irregularity parameter, which is stored in prop.dh

9. An ***if*** statement is initiated to split the path to:

a. The old method, used with low resolution databases, or in the area mode with no database. This subroutine has been re-ordered to more closely match the order of the FORTRAN version in ITS-67. Tests were run with the if dlb<=prop.dist statement enabled, which was buried and inoperative in the ITM FORTRAN (ITS-67) and c++ versions 1.2.2 to 7.0. This computation proved to be unstable and problem-causing beyond a distance of 67 kilometers, and was reburied, as it is not used for high-definition terrain databases, and has been inactive since at least 1967; to leave it where it cannot be activated makes it compatible with the ITM versions.
b. Following an else statement, a new set of instructions to be used when the database has interval widths of less than 150 meters, a value set to activate for databases of 3-arc-second or smaller interval widths.
c. If pfl[0], the number of database intervals is equal to none, indicating that a terrain database has not been loaded into pfl, and plf[1], the interval width is greater than 150 meters, indicating an old terrain database with less than 3-arc-seconds between data points, then the old ITM methodology is followed for the area mode or for use with a database with 30 arc second or larger intervals between the terrain data:

Line (new): If ((pfl[0])=0.0 || (pfl[1]>150.0))

10. Next, a second, embedded **if** statement is initiated. If dlb, the sum of prop.dl[0} and prop.dl[1], the horizon distances from the transmitter to the horizon and the receiver to the horizon, is less than 1.5 times the total path distance prop.dist, then the paths of the transmitter to horizon and receiver to horizon do not overlap enough to insure that the path is line of sight, and not grazing the earth at the horizon.

If the transmit to receive path distance is 10,000 meters (10 km), then the combined total of the distance to the horizon from the transmit site, and of the distance to the horizon from the receive site, must equal or be greater than 150% of 10 km, or 15,000 meters (15 km), overlapping each other by an average of 1/3 of each, for the RF path to be determined to be a definite line-of-sight path.

As a second example, if there is a single obstacle, then the combination of the distance to the horizon (the obstacle) from the transmit site and of the distance to the horizon (the obstacle) from the receive site, would equal the total path distance, and would therefore not meet or exceed the 150% of total path length overlap requirement necessary to determined to be a line-of-sight path. The

determination would be made that the path is trans-horizon, i.e. interrupted by a mutual horizon, by separate horizons, or by one or more obstacles. Therefore, if dlb is not greater than 1.5*prop.dist, indicating a trans-horizon or trans-obstacles path that is not line-of-sight, then:

Line 1312: **if** (dlb<1.5*prop.dist)

11. If the path is a line-of-sight path, the **if** statement then continues:

a. subroutine ***zlsql*** is called with inputs (*pfl, x1[0],xl[1*]), where:
   i. *pfl* is the elevation array;
   ii. *xl[0]* is the transmit site elevation, and
   iii. the term 0.9**prop.dl[0]* specifies a location that is at a point that is 9/10 of the distance from the transmit site toward the transmitter site horizon or highest "visible" obstacle.

b. in order to calculate an average terrain line between points x1[0] and xl[1], and determine the average elevation height on that line at the location of x1[0] and x1[1], ***zlsq1*** then returns:

   za = z0, the elevation value of the average terrain line at the transmitter site.

   q = z1, the elevation value of the average terrain line at the point 9/10 of the distance from the transmit site toward the transmit horizon; either the horizon, or the highest obstruction "visible" from the transmitter site. Note that q here is a placeholder to collect a value stored but not used.

c. The subroutine ***zlsq1*** is called again, this time with inputs (*pfl, prop.dist*–0.9**prop.dl[1],xl[1]*);

Where: *plf* is the elevation array;

the term (*prop.dist*–0.9**prop.dl[1]*) specifies a location that is at a point 9/10th of the distance from the receive site toward the receive site horizon, and;

xl[1] is the receive site elevation

Subroutine ***zlsq1*** calculates an average terrain line between the point represented by (*prop.dist*–0.9**prop.dl[1]*), and the point represented by x1[1], the middle 80% of the path between the receiver horizon (or the last obstruction), and the receiver, and determines the average elevation height on that line at those two points;

zlsq1 then returns:

q = z0, the elevation value of the average terrain line at a point 9/10th of the distance from the receive site toward the receive site horizon. Note; q here is a placeholder; the subroutine output value stored here is not used.

zb = z1, the elevation value of the average terrain line at the receive site.

d. The effective height of the transmit site, *he(0)*, is set to be equal to *prop.hg(0) + pfl(2) – za*, but only if *pfl(2) > za*. If *pfl(2)* is not > *za*, then *he(0)* is set to be equal to *prop.hg(0).*

Therefore *prop.he(0)*, the effective height of the transmit site, is set to be equal to the transmitter antenna height above ground, *prop.hg(0)*, and, (if the actual ground height of the transmit site, *pfl(2)*, is above the average elevation height plane end point at the transmit site, *za*, as established in ***zlsq1)***, then the amount of the difference in height, found by subtracting the average elevation height plane transmit end point height, *za*, from the actual transmitter site ground height from *pfl[2]*, is added to *prop.he(0)*.

d. the effective height of the receive site, *prop.he(1)*, is then similarly set to be equal to *prop.hg(1) + pfl(np+2) – zb*, but only if *pfl(np+2) > zb*. If *pfl(np+2)* is not > *zb*, then *prop.he(1)* is set to be equal to *prop.hg(1).*

Therefore prop.he(1) is set to be equal to prop.hg(1), and if the existing ground height of the receive site, *pfl(np+2)*, is above the average elevation plane end point height at the receive site, *zb*, the difference in height between the average receive site elevation height and the ground height is also added to prop.he(1).

NOTE: In October of 2004, Hammett & Edison, a well respected and highly regarded communications engineering consulting firm in San Francisco, CA, submitted comments to the Federal Communications Commission (FCC) in CS Docket 98-201, regarding the use of Longley-Rice in calculating Grade B TV Signal Coverage. They stated in paragraph 20:

"This ongoing work has convinced us that the implementation of the L-R model is even more flawed than had been originally suspected. For example is has come to light that the OET-69 software calculates the depression angle to a calculation point using the sources height above ground, not its height above sea level. This coding mistake by itself will introduce errors of perhaps 10-20 dB in the calculation results."

The error noted by Hammett and Edison, however, occurs in the FCC's wrap around software, not in the ITM. The ITM has its own problems, which are discussed at the appropriate times.

First, however, we continue to describe the function of the software:

12. In the ITM source code, the effective heights of the transmitter and receiver are originally derived from the input to the call to point_to_point, for tht_m and

thr_m, the transmitter and receiver antenna radiation center heights above ground level, RC-AGL, both in meters. To this is added the height of the transmitter and receiver sites, in meters, above mean sea level, obtained from the profile (pfl) array, which comes from the elevation (elev) array input in the call to point_to_point.

Line : zlsq1(pfl,xl[0],0.9*prop.dl[0],za,q);
zlsq1(pfl,prop.dist-0.9*prop.dl[0],xl[1],q,zb);
prop.he[0]=prop.hg[0]+FORTRAN_DIM(pfl[2],za);
prop.he[1]=prop.hg[1]+FORTRAN_DIM(pfl[np+2],zb);
}

Note: In ITS 67, section 2.4, on page 9 and on page 12, it states that these approximations should only be used when individual path profiles are not available to use to more exactly compute the terrain reflection points, terminal heights, horizon and/or obstruction distances and elevation angles, using subroutine ***hzns***; today, this should apply for area mode only. These approximations have therefore been bypassed for ITWOM for 3-arc-second and finer terrain profiles, and the effective heights are determined by adding the height above ground, $hg_{0,1}$ to the ground height from the terrain database height array $pfl_{2,np+2}$.

13. Above, subroutine ***qlrpfl2*** modifies the effective heights to be no lower than the effective reflection plane between the transmitter and the receiver, as represented by its end points, *za* and *zb*. Qlrpfl2 also modifies the effective heights when the path transitions from the old, original line-of sight range into the old, original beyond-the-horizon range.

14. A **for** loop, with two loops (j=0 and j=1), within the above **for** loop (started in Step 5.) is initiated here, to determine or re-determine the values of prop.dl[0], the distance from the transmitter site to the horizon, and prop dl[1], the distance from the receive site to the horizon, for a line of sight analysis.

NOTE: The Environmental Science Services Administration (ESSA) Technical Report ERL 79-ITS 67, "Prediction of Tropospheric Radio Transmission Loss Over Irregular Terrain, A Computer Method – 1968" by A.G. Longley and P. L. Rice, states on page 12, starting with paragraph 2:

"When individual path profiles are not available, median values of the horizon distances $d_{L1,2}$ are estimated as functions of the median effective antenna heights h_{e1} and h_{e2} determined above, the terrain irregularity factor Δh, and the smooth-earth horizon distances D_{Ls1} and D_{Ls2}. The smooth earth distance from each antenna to its horizon over a <u>smooth</u> <u>earth</u> is defined as:

$$D_{Ls1,\,2} = (.002 * a * h_{e1,\,2})^{.5} \text{ in km.} \qquad \text{ITS67 (5a)}$$

where the effective antenna heights $h_{e1,2}$ are in meters and the effective earth's radius a is in kilometers, as defined by (1). The sum of the smooth-earth horizon distance is

$$D_{Ls} = D_{Ls1} + D_{Ls2}, \text{ in km.} \qquad \text{ITS67 (5b)}$$

Median values of horizon distances over <u>irregular</u> <u>terrain</u> are estimated as

$$D_{L1,2} = D_{Ls1,2} \exp(-.07 (\Delta h/h_e)^{.5} \text{ in km,} \qquad \text{ITS67 (5c)}$$

where

$$h_e = h_{e1,2} \text{ for } h_{e1,2} \geq 5 \text{ meters, or } 5 \text{ meters otherwise.}$$

The total distance, $d_{L,}$ between the antennas and their horizons is

$$d_L = d_{L1} + d_{L2}, \text{ in km".} \qquad \text{ITS67 (5d)}$$

To use these formulas in this subroutine, we convert from km to meters:

$$D_{Ls1,2} = (.002 * a * h_{e1,2})^{.5} \text{ in km.} \qquad \text{ITS67 (5a)}$$

becomes:

$$D_{Ls1,2} = (2 * h_{e1,2} * a)^{.5} \text{ in meters}$$

The effective earth's radius *a*, in kilometers, is defined by ITS67 (1). The term *gma*, representing earth's actual curvature, is a simplified approximation, as it treats the earth as a sphere, not a spheroid. It is established in subroutine ***qlrps*** to be equal to 157e–9 1/meter, and used in step 4 of ***qlrps*** to calculate the effective earth curvature, *gme*, which is then stored in array *prop* at *prop.gme.*

So what is the relationship between *a* and *gma?* One might reasonably assume that earth's actual curvature would be defined as the change per meter of circumference of the earth. If the actual earth's radius is r, then the earth's circumference is:

$$c_e = 2 * (PI) * r$$

and the actual earth's curvature might be defined, per meter, by 1divided by the circumference;

$$gma = 1/ c_e = 1/(2 * PI * r)$$

So for an actual earth radius of *r* = 6,370,000 meters, *gma* would be = *.0000000249 or 249e-10;* but it is not. T*he established value for gma is 157e9, equal to 1/6,370,000 meters.* Therefore, the actual relationship between *a* and *gma* is:

$$gma = 1/(\text{actual earth's radius, in meters})$$

The same relationship therefore applies between the effective earth's radius and the effective earth's curvature:

$$gme = 1/a, \text{ in units of 1/meters, and } a = 1/ gme.$$

So then, $D_{Ls1,2} = (2 * h_{e1,2} * a)^{.5}$ m. becomes $D_{Ls1,2} = (2 * h_{e1,2} / gme)^{.5}$ m.

In converting from km to meters:

$$D_{L1,2} = D_{Ls1,2} \exp(-.07 (\Delta h/h_e)^{.5} \text{ in km,} \qquad \text{ITS67 (5c)}$$

Becomes:

$$D_{L1,2} = D_{Ls1,2} \exp(-.07 (\Delta h/h_e)^{.5} \text{ in meters,}$$

Substituting the formula for $D_{Ls1,2}$ derived above:

$$D_{L1,2} = ((2 * h_{e1,2} / gme) \exp(-.07 (\Delta h/h_e)^{.5})^{.5} \text{ in meters,}$$

where

$$h_e = h_{e1,2} \text{ for } h_{e1,2} \geq 5 \text{ meters, or } 5 \text{ meters otherwise.}$$

Which we can restate in the notation primarily used in this text as:

```
Prop.dl[j] =  ((2*prop.he[j]/prop.gme)^( –.07*(prop.dh/mymax(prop.he[j],5)^.5)
```

The formula for the total distance between the antennas and their horizons is the same as long as all units are in either km or meters:

$$d_L = d_{L1} + d_{L2}, \text{ in meters;}$$

This subroutine, with respect to both line-of-sight and trans-horizon paths, is that the Irregular Terrain Model code, ITMDLL.cpp, still uses a formula derived from ITS67 (5c) to estimate horizon distances for the point_to_point mode, despite the fact that Anita Longley and Phil Rice flatly stated, at the beginning of paragraph 2, page 12 of the ESSA Technical Report ERL 79-ITS 67 quoted above, that it was to be used "only when individual path profiles are not available".

The point_to_point mode utilizes an individual path profile, input as array pfl. The use of the NTIA-released ITMDLL.cpp c++ software requires the use of additional code, a "wrap-around" package (an example being the open source program SPLAT, or the Givens & Bell Inc. LORIS series, that is compiled with the irregular terrain model windows-compatible software (or in the case of SPLAT!, the linux-friendly itm.cpp) that prepares the input, including deriving the pfl array from the raw elevation database data, and processes the output from the core itm.cpp subroutines.

The pfl array, especially when used in multiple runs to analyze signal loss and/or reception over a large area, usually contains far more elevation data, extending along the great circle path through the receive site, than is required to derive the distance to the actual horizon. The first two values stored in the array pfl, as sent to the

point_to_point subroutine call, are pfl[0], the value of the number of intervals, and pfl[1], the value of the width, in meters, of an individual interval. The value of pfl[0] is set to indicate the number of intervals between the transmit site, and a receive site to be considered, and indicates the minimum number of elevation data values stored in the array pfl. The value of pfl[0] does not necessarily indicate the maximum number of elevation data values stored in the array pfl. The pfl array, especially when used in multiple runs to analyze signal loss and/or reception over a large area, usually contains far more elevation data, extending along the great circle path through the receive site, than is required to derive the distance to the receive site in question, as the wrap-around software will store elevation values in the pfl array extending out several tens of kilometers, in anticipation of repeating the point_to_point call to derive loss values at hundreds of receive locations along the rf path being studied. Therefore, the elevation data stored in the pfl array usually, if not always, represents elevation data along the great circle path extending far beyond any value of the horizon for a ground-mounted reception site. In the few cases where this data is not available, i.e. where the database from which the pfl array was derived does not extend to the horizon, this methodology could remain available as a default, and an additional kwx flag could be generated, to indicate that the pfl array does not extend to the radio horizon and that the distances to the radio horizon are estimated.

Therefore, with today's comprehensive elevation databases, including the SRTM and NED elevation data, there is little or no call to continue to use this approximation instead of deriving a more accurate result, where available, for the distances to the actual horizons, from the elevation database. Subroutine ***hzns*** later replaces these estimates with actual values.

15. The value of prop.dl[0] is estimated as a median value of a horizontal distance over irregular terrain using the formula:

    Prop.dl[0] = ((2*prop.he[0]/prop.gme)^(.07*(prop.dh/mymax(prop.he[0],5)^.5)
    ITS67 (5c)

    Where:

    prop.he[0] is the effective height of the transmitter site (from step 10)
    prop.he[1] is the effective height of the receive site (from step 10)
    prop.gme is the effective earth's curvature, input with the point_to_point subroutine call
    prop.dh is the terrain irregularity parameter, Δh, or delta h, obtained from the ***dlthx*** subroutine call on line 1310.

    Similarly, the value of prop dl[1] is initially estimated as:

    Prop.dl[1] = ((2*prop.he[1]/prop.gme)^(–.07*(prop.dh/mymax(prop.he[1],5)^.5)
    ITS67 (5c)

Line 1318: for (j=0; j<2; j++)

```
prop.dl[j]=
sqrt(2.0*prop.he[j]/prop.gme)*exp(–0.07*sqrt(prop.dh/mymax(prop.he[j],5.0)));
```

16. The variable *q* is then set to be equal to the combined total of the distance to the horizon from the transmit site, dl[0], and the distance to the horizon from the receive site dl[1], for a line of sight analysis.

Line 1321: `q=prop.dl[0]+prop.dl[1];`

Note: Many of these initial estimates are replaced later by actual values obtained from the terrain database; for example, after the hzns subroutine is called, the horizon distances in prop.dl[] will have been reset based on the actual horizon and/or obstacles found in the terrain radial path.

17. An **if** loop is initiated; if q, currently holding the sum of the transmitter and receiver estimated horizon distances (later known as dla), is less than the total path distance from the transmit site to the receive site, then the variable *temp* is set to be equal to the total path distance, *prop.dist*, divided by *q*, and the value of *q* is then reset to be equal to the square of the value stored in *temp.*

Therefore, the If statement determines if the sum of the horizon path distances is less than the total path distance. If so, then a rounded horizon, or two obstacles, separates the transmitter horizon from the receive horizon, and the path analysis is ready to move from the line-of-sight range into the diffraction range. The two following calculations then result in the utility variable q holding the value of:

$$q = [prop.dist/(prop.dl[0]+prop.dl[1])]^2$$

A rare comment in the itm.cpp c++ code, referring to the earlier FORTRAN version, states: q=pow(prop.dist/q,2.0); indicating that the ITMDLL.cpp may have used that version of this calculation. A pow calculation is a relatively slow c++ computation; in the itm.cpp, we find a faster version:

Line 1323:
```
if (q<=prop.dist)
{
        /* q=pow(prop.dist/q,2.0); */
        temp=prop.dist/q;
        q=temp*temp;
```

18. A **for** loop is initiated, again with two loops, j=0 and j=1, for the transmitting and receive site locations. This for loop:

a. changes the value of prop.he[0] to be equal to the existing value of prop.he[0], the effective height of the transmit antenna, multiplied by the value of q, and;

b. changes the value of prop.he[1] to be equal to the existing value of prop.he[1], the effective height of the receive antenna, multiplied by the value of q.
c. again resets the value of prop.dl[0], which represents the distance from the transmit site to the horizon, to be equal to:

```
dl[0]= sqrt(2.0*prop.he[0]/prop.gme)*exp(–0.07*sqrt(prop.dh/mymax(prop.he[0],5.0)));
                                                                    ITS67 (5c)
```

Let's take a look at the effect of the adjustment of the effective heights. When the sum of the two horizon distances are equal, as they would be just as the horizon starts to interrupt the path, or when one obstacle separates the transmitter and receiver, the value of q would be 1 or near 1, and the adjustment of the effective heights would be minimal. As the single obstacle becomes two obstacles, and the distance between them increases, q would increase; for a scenario with two obstacles, where the distance between the obstacles is equal to the sum of the two horizon distances, then q would equal $[(1+2+1)/(1+1)]^2=4$, and each effective height would be increased by a factor of 4! So the effective height of the transmit and receive antennas is increased in the diffraction range as the distance between the transmitter horizon obstacle and the receive horizon obstacle increases.

Where, as in step 11 above:

prop.he[0] is the effective height of the transmitter site (from step 10)
prop.he[1] is the effective height of the receive site (from step 10)
prop.gme is the effective earth's curvature, input with the point_to_point subroutine call
prop.dh is the terrain irregularity parameter, Δh, or delta h, obtained from the ***dlthx*** subroutine call on line 1310.

d. And also again resets the value of prop.dl[1], which represents the distance from the receive site to the horizon, to be equal to:

```
dl[1]= sqrt(2.0*prop.he[1]/prop.gme)*exp(–0.07*sqrt(prop.dh/mymax(prop.he[1],5.0)));
                                                                    ITS67 (5c)
```

```
Line 1329:      for (j=0; j<2; j++)
                        {
                   prop.he[j]*=q;
                   prop.dl[j]=sqrt(2.0*prop.he[j]/prop.gme)*exp(–
0.07*sqrt(prop.dh/mymax(prop.he[j],5.0)));
                        }
                }
```

19. A **for** loop is initiated, again with two loops, j=0 and j=1, for the transmitting and receive site locations. This for loop determines the initial value of prop.the[0], the "theta" look-up, or grazing, angle between a line from the transmitter antenna and

the horizon or highest "visible" obstruction, and a horizontal line at the transmitter antenna, and prop.the[1], the receive site look-up angle:

a. resets the value of *q* to be equal to 2 * prop.he[0]/prop.gme
b. sets the value of prop.the[1] equal to:
   prop.the[0]=(0.65*prop.dh*(q/prop.dl[0] –1.0) –2.0*prop.he[0])/q;
c. resets the value of *q* to be equal to 2 * prop.he[1]/prop.gme
d. sets the value of prop.the[1] equal to:
   prop.the[1]=(0.65*prop.dh*(q/prop.dl[1] –1.0) –2.0*prop.he[1])/q;

Line 1336:

```
        for (j=0; j<2; j++)
    {
        q=sqrt(2.0*prop.he[j]/prop.gme);
        prop.the[j]=(0.65*prop.dh*(q/prop.dl[j] –1.0) –2.0*prop.he[j])/q;
    }
}
```

20. An else statement follows, so for when the number of terrain database intervals is greater than zero, indicating the use of a terrain database and the point-to-point mode instead of the area mode, and the terrain database has interval distances smaller than 150 meters (a 3-arc-second terrain database or better), then:

21. Two new steps added for the ITWOM, computes the receiver approach (theta receive approach, or θ_{ra}), and the near receiver approach (theta near receiver, or θ_{nr}) angles.
    a. The receive approach path start location, *rad*, defined as the distance from the transmitter site to the receive approach path start location, is set to be equal to the maximum of the path distance, prop.dist, or prop.dist-450. This defines the receiver approach path to be the 450 meters (or path distance, if less) leading up to the receiver.
    b. The subroutine calls ***zlsq1*** with inputs (*pfl,rad,prop.dist*);
       Where *plf* is the elevation array;
       rad is the distance from the transmitter to a location 550 meters away from of the receive site, or zero (the transmitter site) if the receive site is less than 550 meters from the transmitter.
       prop.dist is the path distance from the transmitter site to the receiver site.

       Subroutine ***zlsq1*** calculates an average terrain line between *rad* and the receive site, and determines the average elevation height on that line at the locations *rad* and the receive site.

       zlsq1 then returns:
       rae1 = z0, the elevation value in meters of the receiver approach path average terrain line at *rad,* the receive approach path start location.
       rae2= z1, the elevation value in meters of the average terrain line at the receive site.

c. The slope is then recovered and converted to an angle, θ_{ra}, theta receive approach, the receive approach angle in radians, to be stored in prop array argument *prop.thera*:

prop.thera = arctan((*rae2- rae1*)/ *pfl[1]*);

where *pfl[1]* is the interval width in meters.

d. The angle θ_{nr} , theta near receiver, the slope angle of a line between the last terrain point elevation height before the receiver and the receiver terrain height, in radians, is computed from the respective terrain heights and the interval width, and stored in prop array argument *prop.thenr*:

prop.thenr = arctan((*pfl[np+2]-pfl[np+1]*)/ *pfl[1]*);

where *pfl[np+2]* is the elevation height of the receiver
pfl[np+2] is the elevation height of the terrain database location one interval before the receiver.
pfl[1] is the interval width in meters.

Line:
```
rad=mymax(0.0,prop.dist-450.0);
zlsq1(pfl,rad,prop.dist,rae1,rae2);
prop.thera=atan((rae2-rae1)/pfl[1]);
prop.thenr=atan((pfl[np+2]-pfl[np+1])/pfl[1]);
```

22. The value of prop.mdp, the mode of the propagation model, is set to be equal to – 1, indicating point_to_point mode, and the value of propv.lvar, the level to which coefficients in AVAR must be redefined, is set to be equal to the greater value of either propv.lvar or 3.

Line:
```
prop.mdp=-1;
propv.lvar=mymax(propv.lvar,3);
```

23. An **if** statement is initiated, stating that if mdvarx, a variable representing the mode of variability, is greater than zero, (zero representing the single message mode), then the value of prov.mdvar is set equal to the value of mdvarx, and the value of propv.lvar is set to the be equal to the greater value of propv.lvar, or 4.

Line:
```
if (mdvarx>=0)
{
propv.mdvar=mdvarx;
propv.lvar=mymax(propv.lvar,4);
```

```
}
```

24. An **if** statement is initiated, stating that if klimx, the climate variable, is greater than zero, the value of propv.klim, the climate code, is set to be equal to the value of klimx, and the value of propv.lvar, which may have been set in step 18 above, is reset to be equal to 5.

```
Line:     if (klimx>0)
          {
                propv.klim=klimx;
                propv.lvar=5;
          }
```

Note: for more information re: lvar, mdvar, and klimx, see "A manual for ITM, "Irregular Terrain Model", available at http://flattop.its.bldrdoc.gov/itm/itm_man.txt.

25. Finally, we arrive at the climax; calculating the Longley-Rice path loss. The subroutine then calls ***lrprop*** with inputs 0.0, array prop, and array propa;

```
Line:          lrprop2(0.0,prop,propa);
}
```

lrprop returns the reference attenuation, *aref*. This is the "answer", the amount of loss, in db, in the rf signal level between the transmitter and the receiver.

SUBROUTINE QLRPS: A functional explanation, by Sid Shumate.

Started, May 2007; Last Revised Sept 16, 2007.

Quick Longley-Rice Preparatory Subroutine; ***qlrps.***

Subroutine starts at Line 370.

Note: Used with both point-to-point and area prediction modes.

Derived from the Irregular Terrain Model description by George Hufford, 2002, (ITMD), compared to the ITM.cpp prepared by J. D. McDonald and John Magliacane for compilation on unix and linux systems. "Line" numbers refer to the ITM.cpp as line numbered by Bloodshed Software's DevC++ print function. "Alg" numbers refer to the algorithm formula in "The ITS Irregular Terrain Model, version 1.22, the Algorithm" by G. A. Hufford, 1995. "TN101" numbers refer to equations in "Tech Note 101".

From ITMD Section 41:

Call inputs:

fmhz	Frequency, in MHz range 20 up to 2,000
zsys	general system elevation (calculated by ***lrprop*** subroutine to be the average elevation of the middle 80% of elev_l (a.k.a. pfl) path elevations, starting at 1/10 of the path and ending at 9/10 of the path.)
en0	Surface refractivity reduced to sea level (a.k.a. Atmospheric Bending Constant); normal default value = 301.000 N-Units
ipol	polarity (H=0, V=1) for FM vehicular reception, normal default =1
eps	polarization constant (a.k.a. Earth's Dielectric Constant, or Relative permittivity); normal default value = 15.000
sgm	ground constant (a.k.a. Earth's Conductivity); normal default value = 0.005 Siemens/meter

This subroutine:

1. Declares and establishes the constant *gma* to be equal to 157e-9. The constant *gma* represents an approximation of earth's actual curvature, as it represents the curvature of a sphere, where the earth is in fact, a spheroid. The "earth's actual curvature" is specified as: 157 N-units/km., or (157 * 10^-9)/m. The units of *gma* is 1/meter.

Line 372: double *gma* = 157e-9

2. Converts the frequency *fmhz*, to wave number *wn* .

Line 374: prop.wn=fmhz/47.7 [Alg. 1.1]

The wave number at 100 MHz, as an example, would be = 2.0964/meter.

3. Uses the surface refractivity reduced to sea level, *en0*, and the general system elevation, *zsys*, to calculate the surface refractivity, *ens*, using the formula:

ens = en0^(-zsys/z1), where z1 = 9,460 m. [Alg. 1.2]

Line 375: prop.ens=en0
Line 377: if (zsys!=0.0) /* zsys is preset to 0 by point_to_point for first round. */
Line 380: Prop.ens*=exp(-zsys/9460.0) /* if zsys is not equal to 0.0, multiply Zsys by the exponent (-zsys/9460.0).

If, for example, zsys =946 m., ens =eno^(-.1) = 301^(-.1) = .565 N-units.

4. Uses the constant *gma*, earth's actual curvature, and the surface refractivity *ens* (prop.ens) calculated on line 380, to calculate the effective earth curvature *gme* (prop.gme), using the empirical formula *gme=gma**(1-0.04665^(ens/en1)) [Alg. 1.3] , Where en1 is 179.3 N-units.

Line 380: prop.gme=gma*(1.0-0.04665*exp(prop.ens/179.3))
If, for example, ens = .565 N-units,

prop.gme =(157*10^(-9))*(1-(.04665)^(.565/179.3))= (1.509e-9)/meters.

Units for prop.gme are: 1/meters.

5. Calls ***complex,*** a c++ subroutine, to use the polarization constant, *eps*, the ground constant, *sgm*, the wave number *wn* (prop.wn), and the polarity *ipol*, to calculate the surface impedance *zgnd* (prop_zgnd) as a complex number, with both real (resistance) and imaginary (impedance) values.

Line 381: complex<double> zq, prop_zgnd(prop.zgndreal,prop.zgndimag);
Line 382: zq=complex<double> (eps, 376.62*sgm/prop.wn); [Alg. 1.5]
Line 383: prop_zgnd=sqrt(zq-1.0)

Line 385: if (ipol!=0.0)
Line 386: prop_zgnd=prop_zgnd/zq; [Alg. 1.4]

Line 388: prop.zgndreal=prop_zgnd.real()
Line 399: prop.zgndimag=prop_zgnd.imag()

6. Outputs to prop_type structure (prop_type & prop) :

a. *Prop.wn* wave number
b. *prop.ens* surface refractivity
c. *prop.gme* effective earth curvature
d. *prop.zgnd* surface impedance
e. *prop.zgndreal* real surface impedance (resistance component)
f. *prop.zgndimag* imaginary surface impedance (reactive component)

SUBROUTINE QTILE: A functional explanation, by Sid Shumate.
Last Revised May 18, 2007.

Quartile subroutine: ***qtile***.

Note: Used with point-to-point mode. Called by ***dlthx***, near the end of the routine.
Calls ***mymin, mymax***.

Derived from the Irregular Terrain Model description by George Hufford, 2002, (ITMD), compared to the ITM.cpp prepared by J. D. McDonald and John Magliacane for compilation on unix and linux systems. "Line" numbers refer to the ITM.cpp as line numbered by Bloodshed Software's DevC++ print function. "Alg" numbers refer to the algorithm formula in "The ITS Irregular Terrain Model, version 1.22, the Algorithm" by G. A. Hufford, 1995.

Used to find a quartile. It reorders the array *a* so that *a(j),j = 0..ir* are all greater than or equal to all *a(i),i = ir….nn.* In particular, *a(ir)* will have the same value it would have if *a* were completely sorted in descending order. The returned value is *qtile = a(ir).*

Some Background on Quantile Definitions and Methodologies:

From *Wikipedia:*

Quantiles are points taken at regular intervals from the cumulative distribution function of a random variable. Dividing ordered data into q essentially equal-sized data subsets is the motivation for q-quantiles; the quantiles are the data values marking the boundaries between consecutive subsets. Put another way, the kth q-quantile is the value x such that the probability that a random variables will be less than x is at most k/q and the probability that a random variable will be less than or equal to x is at least k/q. There are $q - 1$ quantiles, with k an integer satisfying $0 < k < q$.

Some quantiles have special names:

- The 100-quantiles are called percentiles.
- The 20-quantiles are called duo-deciles.
- The 10-quantiles are called deciles.
- The 9-quantiles are called noniles, common in educational testing.
- The 5-quantiles are called quintiles.
- The 4-quantiles are called quartiles.

More generally one can consider the quantile function for any distribution. This is defined for real variables between zero and one and is mathematically the inverse of the cumulative distribution function.

Some software programs (including Microsoft Excel) regard the minimum and maximum as the 0th and 100th percentile, respectively; however, such terminology is an extension beyond traditional statistics definitions. For an infinite population, the *k*th quantile is the data value where the cumulative distribution function is equal to *k*/*q*. For a finite *N* sample size, calculate $N \cdot k/q$ --if this is not an integer, then round up to the next integer to get the appropriate sample number (assuming samples ordered by increasing value); if it is an integer then any value from the value of that sample number to the value of the next can be taken as the quantile, and it is conventional (though arbitrary) to take the average of those two values (see Estimating the quantiles).

More formally: the *k*th "q"-quantile of the population parameter *X* can be defined as the value "x" such that:

$$P(X \leq x) \geq p \text{ and } P(X \geq x) \geq 1 - p \text{ where } p = \frac{k}{q}$$

or equivalently

$$P(X < x) \leq p \text{ and } P(X > x) \leq 1 - p \text{ where } p = \frac{k}{q}$$

If instead of using integers *k* and *q*, the *p*-quantile is based on a real number *p* with 0<*p*<1 then this becomes: The *p*-quantile of the distribution of a random variable *X* can be defined as the value(s) *x* such that:

$$P(X \leq x) \geq p \text{ and } P(X \geq x) \geq 1 - p$$

or equivalently

$$P(X < x) \leq p \text{ and } P(X > x) \leq 1 - p.$$

For example, given the 10 data values {3, 6, 7, 8, 8, 10, 13, 15, 16, 20}, the first quartile is determined by 10*(1/4) = 2.5, which rounds up to 3, meaning that 3 is the rank in order of samples (from least to greatest values), at which, approximately 1/4 samples have values less than this third sample, which in this case is 7. The second quartile value (same as the median) is determined by 10*(2/4) = 5, which is an integer, while the number of samples (10) is an even number, so the average of both the fifth and sixth values is taken--that is (8+10)/2 = 9, though any value from 8 through to 10 could be taken to be the median. If the number of data values is odd, then the median value (or 2nd quartile) is the value found at sample=(#values + 1)/2 (so for this example if there had also been a value of 9 between values 8 and 10, making 11 samples total, then (11+1)/2=6, meaning that the sixth sample (in this case the value 9), would be the 2nd quartile, where 1/2 of the samples have values greater than the value at this sample (greater than 9--the value at sample 6 of 11), and 1/2 of the samples have values less than the value at this sample.

The third quartile value for the original example above is determined by 10*(3/4) = 7.5, which rounds up to 8, and the eighth sample is 15. The motivation for this method is that the first quartile should divide the data between the bottom quarter and top three-quarters. Ideally, this would mean 2.5 of the samples are below the first quartile and 7.5 are above, which in turn means that the third data sample is "split in two", making the third sample part of both the first and second quarters of data, so the quartile boundary is right at that sample.

Standardized test results are commonly misinterpreted as a student scoring "in the 80th percentile", for example, as if the 80th percentile is an interval to score "in", which it is not; one can score "at" some percentile or between two percentiles, but not "in" some percentile.

It should be noted that different software packages use slightly varying algorithms, so the answer they produce may be slightly different for any given set of data. Besides the algorithm given above, which is the proper one based on probability, there are at least four other algorithms commonly used (for various reasons, such as of ease of computation, ignorance, etc.).

If a distribution is symmetrical, then the median is the mean (so long as the latter exists). But in general, the median and the mean differ. For instance, with a random variable that has an exponential distribution, any particular sample of this random variable will have roughly a 63% chance of being less than the mean. This is because the exponential distribution has a long tail for positive values, but is zero for negative numbers.

Quantiles are useful measures because they are less susceptible to long tailed distributions and outliers.

Empirically, if the data you are analyzing are not actually distributed according to your assumed distribution, or if you have other potential sources for outliers that are far removed from the mean, then quantiles may be more useful descriptive statistics than means and other moment related statistics.

Closely related is the subject of least absolute deviations, a method of regression that is more robust to outliers than is least squares, in which the sum of the absolute value of the observed errors is used in place of the squared error. The connection is that the mean is the single estimate of a distribution that minimizes expected squared error while the median minimizes expected absolute error. Least absolute deviations shares the ability to be relatively insensitive to large deviations in outlying observations, although even better methods of robust regression are available.

The quantiles of a random variable are generally preserved under increasing transformations, in the sense that for example if m is the median of a random variable X then 2^m is the median of 2^X, unless an arbitrary choice has been made from a range of values to specify a particular quantile. Quantiles can also be used in cases where only ordinal data is available.

[edit] Estimating the quantiles

There are several methods for estimating the quantiles:

Let N be the number of non-missing values of the sample population, and let $x_1, x_2, \ldots, x_N$ represent the ordered values of the sample population such that x_1 is the smallest value, etc. For the kth q-quantile, let $p = k / q$.

Empirical distribution function

$$\begin{cases} x_j, & g = 0 \\ x_{j+1}, & g > 0 \end{cases}$$

j is the integer part of $N \cdot p$ and g is the fractional part

Empirical distribution function with averaging

$$\begin{cases} \frac{1}{2}(x_j + x_{j+1}), & g = 0 \\ x_{j+1}, & g > 0 \end{cases}$$

j is the integer part of $N \cdot p$ and g is the fractional part

Weighted average

$$x_{j+1} + g \cdot (x_{j+2} - x_{j+1})$$

j is the integer part of $(N-1) \cdot p$ and g is the fractional part. This method is used for example in the PERCENTILE function of Microsoft Excel.

Sample number closest to *(N-1)·p+1*

$$\begin{cases} x_j, & g < .5 \\ x_{j+1}, & g \geq .5 \end{cases}$$

j is the integer part of $(N-1) \cdot p + 1$ and g is the fractional part

And from: Weisstein, Eric W. "Quantile." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/Quantile.html

The word quantile has two meanings in probability. Specific elements x in the range of a variate X are called quantiles, and denoted x (Evans *et al.* 2000, p. 5).

The kth n-tile P_k is that value of x, say x_k, which corresponds to a cumulative frequency of Nk/n (Kenney and Keeping 1962). If $n=4$, the quantity is called a quartile, and if $n=100$, it is called a percentile.

A parametrized version of quantile is implemented as `Quantile`[*list*, *q*, {{*a*, *b*}, {*c*, *d*}}], which returns

$$q_{a,b;c,d}(X_1, \ldots, X_N) = Y_{\lfloor x \rfloor} + (Y_{\lceil x \rceil} - Y_{\lfloor x \rfloor})(c + d\,\mathrm{frac}(x)),$$

where Y_i is the ith order statistic, $\lfloor x \rfloor$ is the floor function, $\lceil x \rceil$ is the ceiling function, $\mathrm{frac}(x)$ is the fractional part, and

$$x \equiv a + (N+b)q.$$

There are a number of slightly different definitions of the quantile that are in common use, as summarized in the following table.

#	a	b	c	d	plotting position	Description
Q1	0	0	1	0	i/n	inverted empirical CDF
Q2	--	--	--	--	i/n	inverted empirical CDF with averaging
Q3	$\frac{1}{2}$	0	0	0	$\left(i+\frac{1}{2}\right)/n$	observation numberer closest to qn
Q4	0	0	0	1	i/n	California Department of Public Works method
Q5	$\frac{1}{2}$	0	0	1	$\left(i-\frac{1}{2}\right)/n$	Hazen's model (popular in hydrology)
Q6	0	1	0	1	$i/(n+1)$	Weibull quantile
Q7	1	-1	0	1	$(i-1)/(n-1)$	interpolation points divide sample range into $n-1$ intervals
Q8	$\frac{1}{3}$	$\frac{1}{3}$	0	1	$\left(i-\frac{1}{3}\right)/\left(n+\frac{1}{3}\right)$	unbiased median
Q9	$\frac{3}{8}$	$\frac{1}{4}$	0	1	$\left(i-\frac{3}{8}\right)/\left(n+\frac{1}{4}\right)$	approximate unbiased estimate for a normal distribution

Mathematica's parametrization can handle all of these but Q2. In Q1, the empirical cumulative distribution function is the estimated cumulative proportion of the data set that does not exceed any specified value. Q2 is essentially the same as Q1 except that averages are taken at points of discontinuity. In Q3, the

qth quantile is the observation numbered closest to qn, where n is the sample size. In Q4, the interpolation points divide the sample range into n intervals. In Q6, the vertices divide the sample into $n+1$ regions, each with probability $1/(n+1)$ on average. It was proposed by Weibull in 1939, and plots X_i at the mean position. Q7 divides the range into $n-1$ intervals, of which exactly $100q\%$ lie to the left of q. Q8 plots X_i at the median position. Q9 is used in quantile-quantile plots. If $P(X)$ is the normal distribution and p_k is the plotting position of X_k, then $Q9(p_k)$ is an approximately unbiased estimate of $P^{-1}(p_k)$.

REFERENCES:

Barnett, V. "Probability Plotting Methods and Order Statistics." *Appl. Stat.* **24**, 95-108, 1975.

Cunnane, C. "Unbiased Plotting Positions--A Review." *J. Hydrology* **37**, 205-222, 1978.

Evans, M.; Hastings, N.; and Peacock, B. *Statistical Distributions, 3rd ed.* New York: Wiley, 2000.

Harter, H. L. "Another Look at Plotting Positions." *Comm. Stat., Th. and Methods* **13**, 1613-1633, 1984.

Hyndman, R. J. and Fan, Y. "Sample Quantiles in Statistical Packages." *Amer. Stat.* **50**, 361-365, 1996.

Kenney, J. F. and Keeping, E. S. "Quantiles." §3.5 in *Mathematics of Statistics, Pt. 1, 3rd ed.* Princeton, NJ: Van Nostrand, pp. 37-38, 1962.

LAST MODIFIED: June 5, 2003

Now to the subroutine:
From ITMD Section 52:

Call inputs:

&nn, a constant integer representing the number of data points in the array *a*
a array with *nn* data points:
a[0]
a[1]
a[2]
a[3]

&ir the quantile desired.

declares private, or local, arguments:

q = 0.0
r
m
n number of data points in array; set equal to input nn at start of subroutine
i
j
j1 = 0
iO = 0, k;
bool done=false
bool goto10=true

This subroutine:

1. Presets *m* to be equal to 0, and n=nn . When called by ***dlthx***, nn = number of data points in array ***a.***

Line 1157: m=0;
Line 1158: n=nn

2. Defines k to have a range from 0.0 to the value of n, and presets the value of k to be equal to ir.

Line 1159: k=mymin(mymax(0,ir),n);

3. Starts a ***while*** loop, that continues until *done,*defined during declaration to be equal to a Boolean <u>false</u>, is not equal to <u>true</u>.

Line 1161: while (!done)

4. The first of the series of ***if*** statements in the ***while*** loop states that if (goto10), defined during declaration to be equal to a Boolean true, is true, then *q* is set equal to the input *a* array value at array location a[k], *i0* is set equal to *m*, which on the first pass is 0.0, and *j1* is set equal to *n*, which on the first pass has been set equal to *nn*, the number of data points in the array *a*.

Line 1163:
```
if (goto10)
{
        q=a[k];
        i0=m;
        j1=n;
}
```

5. *i* is then set to be equal to *i0*, which on the first pass was set equal to *m*, which was preset to 0.0. So *i* starts at 0.0.

Line 1170: i=i0;

6. a ***while*** statement is placed inside of the ***while*** loop started on line 1161. This statement acts as long as both the value of *i* is equal or less than n, and the value of the *a* array at location a[i] is equal to or greater than q. If the while statement is true, the value of *i* is incremented, i.e. increased by 1, on each pass.

Line 1172:
```
while (i<=n && a[i]>=q)
        i++;
```

7. The next ***if*** statement in the ***while*** loop started at line 1161, states that if *i* is greater than *n, i* is to be set equal to *n*.

Line 1175:
```
if (i>n)
        i=n;
```

8. Then j is set to be equal to j1. j1 having been preset to 0.0 on the first pass, j is then preset to 0.0 on the first pass.

Line 1178: j=j1;

9. A second ***while*** statement is placed inside of the first. This one acts as long as both the value of j is equal or greater than m, which starts at zero, and the value of

the *a* array value [j] is equal to or less than q. If the while statement is true, the value of *j* is negatively incremented, i.e. decreased by 1, on each pass.

Line 1180:
```
while (j>=m && a[j]<=q)
        j--;
```

10. The next ***if*** statement in the ***while*** loop started at line 1161, states that if *j* is less than *m, j* is set to be equal to *m*. On the first pass, m =0, so if j is less than 0, j is set to be equal to 0.0.

Line 1183:
```
        if (j<m)
        j=m;
```

11. The next ***if*** statement in the ***while*** loop, states that if *i* is less than *j*,

a. *r* is set to be equal to the *a* loop value [i].
b. then the *a* loop value [i] is reset to be equal to the *a* loop value [j].
c. then the *a* loop value [j] is reset to be equal to *r*.
d. *i0* is reset to be equal to the value of *i* + 1.
e. *j1* is reset to be equal to the value of *j* - 1.
f. goto10 is set to be equal to Boolean <u>false</u>.

Line 1186:
```
        if (i<j)
    {
        r=a[i];
        a[i]=a[j];
        a[j]=r;
        i0=i+1;
        j1=j-1;
        goto10=false;
    }
```

12. The ***if*** statement at line 1186 is followed by three ***else if*** statements. The first of these ***else if*** statements operates if *i* is not less than *j* at line 1186, and *i* is less than *k*; in that case,

a. the value of the *a* array value [k] is set to be equal to the value of the *a* array value [i].
b. then the *a* loop value [i] is reset to be equal to the value of *q*.
c. the value of *m* is reset to be equal to the value of *i* + 1.
d. goto10 is set to be equal to Boolean <u>true</u>.

Line 1196:
```
        else if (i<k)
        {
            a[k]=a[i];
            a[i]=q;
            m=i+1;
```

```
            goto10=true;
        }
```

13. The ***if*** statement at line 1186 is followed by three ***else if*** statements. The second of these ***else if*** statements operates if *i* is not less than *j* at line 1186, and *j* is greater than *k*; in that case,

    a. the value of the *a* array value [k] is set to be equal to the value of the *a* array value [j].
    b. then the *a* loop value [j] is reset to be equal to the value of *q*.
    c. the value of *n* is reset to be equal to the value of *j* - 1.
    d. goto10 is set to be equal to Boolean <u>true</u>.

Line 1204:

```
        else if (j>k)
    {
        a[k]=a[j];
        a[j]=q;
        n=j-1;
        goto10=true;
    }
```

14. The ***if*** statement at line 1186 is followed by three ***else if*** statements. The third of these ***else if*** statements operates if *i* is not less than *j* at line 1186, *i* is not less than *k* at line 1196, and *j* is not greater than *k* at line 1204; in that case done is set to be equal to Boolean <u>true</u>, completing the ***while*** loop started at line 1161.

Line 1204:

```
    else
    done=true;
```

15. If the ***while*** loop is complete, the ***qtile*** subroutine reports out the single quantile value *q*.

Line 1216:

```
    return q;
```

SUBROUTINE SAALOS: A functional explanation, by Sid Shumate.

Last Revised: 14August,2010 to force saalos to zero if rcvr height above clutter canopy.
Tuesday August 1, 2010 to add transmitter height input value prop.rch[].
Reference to itwom 2.0a.cpp

Attenuation due to Clutter Losses, calculated using Shumate's Approximations in the Line of Sight prior to an Obstruction subroutine; ***saalos***.

Note: To be used with both point-to-point and area modes. Called by ***alos2.***

Calls ***abq_alos, mymin,*** and ***mymax.***

Descriptions derived from "Deterministic Equations for Computer Approximation of ITU-R P.1546-2" Sid Shumate, presented on June 4, 2008 before the 10th Annual International Symposium on Advanced Radio Technologies, held at NIST Boulder Labs, June 2-4, 2008; "SA" numbers refer to equations in this document. ITU-R numbers refer to the specified International Telecommunications Union Recommendation. "Line" numbers refer to the ITWOM.cpp as line numbered by Bloodshed Software's DevC++ print function.

Background: Place earth field, a.k.a. ray-tracing, two-ray, and multipath calculations.

Plane earth fields calculations, as performed in Longley-Rice, quickly fade out over cluttered and/or irregular terrain paths where the receive antenna is at or below the clutter line, as the absorption of the clutter and the Snell's Law transmissivity loss as the signal crosses the air-to-clutter-canopy interface, reduces the strength of the ground-reflected ray so that its cancellation effect upon arrival at the receiver is no longer significant, except for circumstances near the horizon were the main and reflected cluttered paths lengths are nearly equal.

Reflection coefficient, the PSB and low grazing angles:

"Grazing Behavior of Scatter and Propagation Above Any Rough Surface", was published in January of 1998 in the IEEE Trans. On Antennas and Propagation. The author is Dr. Donald Barrick, a radar propagation expert who served from 1972 to 1982 as Chief of the Sea State Studies Division of NIST's Wave Propagation Laboratory in Boulder, CO. The conclusions include:

> "Our results show that backscattered power depends on grazing angle to the fourth power; the impedance and admittance are constant as grazing is approached. These relations hold true for both polarizations, for arbitrary surface materials (including perfect conductors), for all frequency/roughness scales, and

> for a single deterministic roughness profile as well as averages over surface ensembles. …we considered only backscatter rather than arbitrary bistatic scatter, …but the extension to bistatic scatter is obvious: as either the incidence or scattering angle alone approaches grazing, echo power decreases as grazing angle squared.
>
> Although our approach was primarily employed to establish general grazing-limit behavior, our simple angle-independent constants describing backscatter and propagation are useful in their own right; these expressions allow a single numerical evaluation to serve the entire near-grazing region up to the Brewster angle."

Bistatic radar systems, including passive radar systems, are radar systems where the transmitter and receiver are widely separated. They operate in a like manner to FM and TV broadcasting systems with respect to plane wave considerations; therefore, we can extend Dr. Barrick's conclusions to the case of plane wave reflections.

For additional background, see the chapter on "Shumate's Approximations for the ITU-R P.1546-2".

This is an entirely new subroutine; there is no reference to it in the ITM documentation.

The subroutine function ***saalos*** computes the additional attenuation suffered by a radio signal on a path traversing a fully or partially cluttered path in the line-of-sight range for a total path distance d. This is an extension of the ***alos2*** subroutine found in ITWOM and the ITWOM.cpp. It computes the path, either a direct line from a transmitter at or below an average clutter height level, or for a transmitter above the clutter canopy, a Snell's law-geometry two-ray transmissive path, to a receiver located at or below the level of the clutter canopy top.

The computations in this subroutine utilize Shumate's Approximations; deterministic approximation equations derived from ITU-R 1546-2, Figures 1, 9 and 17, to determine the Radiative Transfer Engine attenuation from clutter loss and scatter where the receive site is below the clutter canopy line, that must be added to two-ray cancellation and free-space dispersion in the line of sight mode.

TN101 states that, except where noted, all logarithmic functions are to the base 10. This new subroutine continues this convention. In the text, any reference to log refers to $\log_{10}$, a common logarithmic function.

Call inputs:
double d flat-earth radio path length from transmitter to receiver, or for when called by adiff, from obstruction top to receiver.
prop_type
& prop array with constants
propa_type

& propa array with constants

defines private, or local, arguments:

double ens	refractive index of atmosphere at surface level (canopy top)
double encc	refractive index of an average clutter canopy
double s	sigma, $\sigma_h(d)$, the standard deviation of Δh
double n	for loop control value
double q	utility variable
double dp	d prime; interim iteration value of d
double tde	earth curvature correction angle for the actual earth radius, $\theta_{\Delta e}$
double hc	earth curvature height
double ucrpc	un-cluttered radio path w/earth curvature correction
double ctip	cosine of the flat earth incident angle θ_i' (theta i prime)
double tip	flat earth incident angle θ_i' (theta i prime)
double tic	total incident angle; θ_{ic}
double stic	sin of the total incident angle, $\sin(\theta_{ic})$
double ctic	cosine of the total incident angle, $\cos(\theta_{ic})$
double sta	sin of the transmission angle, $\sin(\theta_{tc})$
double ttc	transmission angle, θ_{tc}
double cttc	cosine of the transmission angle, $\cos(\theta_{tc})$
double crpc	cluttered radio path with earth correction
double ssnps	sin of the Snell's Law grazing angle, $\sin(\Psi)$
double d1a	clutter canopy surface distance; d_{1a}
double rsp	reflectivity for the selected polarity, R_{SP}
double tsp	transmissivity for the selected polarity, T_{SP}
double arte	attenuation from Radiative Transfer Engine mode function, A_{RTE}
double zi	frequency compensation zero intercept point for RTE I_3 mode
double pd	path distance in meters, from prop.dist for line-of-sight, or from receive horizon path distance, prop.dl[1], beyond the first obstruction.
Double pdk	path distance in kilometers; equal to pd/1000.
double hone	the difference between the actual transmitter antenna radiation center height AMSL and the actual receiver site ground height AMSL.
double tvsr	transmitter versus receiver height, the difference between the actual transmitter antenna radiation center height AMSL and the actual receiver antenna radiation center height AMSL.
double saalosv	output value of Shumate's Approximation attenuation, line of sight

Note: prop.tgh and prop.tsgh were created to be used as additional inputs in this subroutine.

Prop.tgh represents the transmitter antenna height above ground height (HAGL) of either the main transmitter in a call from alos2, or the theoretical transmit antenna height in a call from adiff2.

Prop.tsgh represents the transmitter site ground height (AMSL) of either the main transmitter in a call from alos2, or of an obstruction peak in a call from adiff2.

In this subroutine:

1. An **if** statement is initiated; **if** d is equal to zero, then:
    a. Coefficients are preset, and saalosv is set to be equal to zero.

Line (new):
```
if (d= = 0.0)
{
        tsp=1.0;
        rsp=0.0;
        d1a=50.0;
        saalosv=0.0;
}
```

2. An ***if else*** statement follows; **if** the receive antenna height is equal to or greater than the clutter height, then:
    a. saalosv is set to be equal to zero.

Line (new):
```
if else (prop.hg[1] >prop.cch )
{
    saalosv=0.0;
}
```

3. An **else** statement follows, so if d is not equal to zero, then:

    a. pd is set to be equal to d; the input path distance, equal to prop.dist in a call from alos, or to prop.dl[1] in a call from adiff. In units of meters.
    b. pdk is pd in kilometers, set to be equal to pd/1000.
    c. tsp is initialized to 1.0.
    d. rsp is initialized to 0.0.
    e. dla is preset to be equal to pd, the path distance.
    f. hone, here the difference between the transmit antenna height AMSL and the receive site ground level AMSL, is set to be equal to the transmit antenna height above ground, prop.tgh, and the transmit site ground level AMSL, prop.tsgh, less the sum of: (the actual receive antenna height AMSL, prop.rch[1] – the receive antenna height above ground level prop.hg[1]). All in meters.

    g. An **if** statement is initiated; **if** the transmitter height above ground level prop.tgh is greater than the clutter canopy height, prop.cch, then the Snell's Law path geometric parameters must be calculated.

if(prop.tgh>prop.cch)
{

To calculate them, we first need to determine the refractivity of the air and clutter canopy.

The refractive index of a vacuum is, by definition, the minimum value of $\eta_v = 1.0$. The refractive index of air is about 1.0003; of water, about 1.33. The atmospheric radio refractive index η can be computed from N, the radio refractivity:

$$\eta = 1 + N * 10^{-6} \qquad \text{[ITU-R P.453-7, (1)]}$$

In ESSA Technical Report ERL 79-ITS 67,(ITS-67), Longley and Rice stated that the surface refractivity can vary between 240 to 400 N-Units, and that a commonly used value of Ns is 301 N-units. University of Oklahoma research tests in 2005 using the NOAA National Weather Radar phased array test-bed to determine surface refractivity, showed that atmospheric refractivity stabilizes near 301 N-units on cold, clear days with little wind; dropping to below 240 units during very hot, dry weather, and exceeding 400 N-units on hot days when the humidity approaches 100%.

In the analysis of ITU-R P.1546-2 data used to derive Shumate's Approximations, it was found that the best deterministic match to over-land data in figures 1, 9 and 17, occurs when the average clutter canopy refractivity index η_{cc}, is: $\eta_{cc} = 1.0010$. For compatibility with the data input format for Longley-Rice, we have added an input to the ***point_to_point*** subroutine for the clutter canopy refractivity in N-units, with a preset value of: 1,000 N-units (equal to $\eta_{cc} = 1.0010$).

Therefore, we can calculate the refractivity of the atmosphere at or near ground level, η_s (ens) and the average refractivity of the clutter canopy, η_{cc} (encc).

(1) ens = 1 + prop.ens*10^{-6}
(2) encc = 1 + prop.encc*10^{-6}

We now proceed to calculate, by iteration to the accuracy required, the Snell's Law bent-path geometry for a radio path from a transmitter above the canopy and a receiver at or below the clutter canopy;

h. dp is set to be equal to d, which is equal to the path distance between the transmitter site considered and the receiver, in meters:

dp=d

i. n is set to 5; then a for loop statement is initiated; ***for*** n>0;

Step 1: calculate the earth curvature correction angle for the actual earth radius, $\theta_{\Delta e}$:
$\theta_{\Delta e} = d/r$, as:

tde=dp/6378137

where: dp represents the radio path length (the flat-earth path length is equal to the curved-earth path length) from transmitter site under consideration (may be a secondary site atop obstruction) to receiver site.

r is the actual earth radius: 6,378,137 meters.

Step 2: calculate the earth curvature adjustment height; hc: $hc = (CH + r)(1 - \cos(\theta_{\Delta e}))$:

where: CH (prop.cch) is the average clutter canopy height above ground level

hc=(prop.cch+6378137)*(1– cos(tde))

Step 4: calculate the un-cluttered radio path w/earth curvature correction; ucrpc: $ucrpc = [(h_1 - h_2 - h_{g2} - CH + hc)^2 + (dp)^2]^{1/2}$, where h_1 and h_2 are the respective transmitter and receiver antenna heights above mean sea level (RCAMSL), and h_{g2} is the receive antenna height above ground level. Here, hone has been set to be equal to the height of the transmitter site antenna above the receive site ground level, i.e. $hone = (h_1 - h_2 - hg_2)$, where h_1 is obtained by adding the transmitter (or secondary transmitter) transmitter radiation center above ground, tgh, to the transmitter site ground level height AMSL, tsgl.

Note that in the first for loop cycle, the value of d prime, *dp*, (which represents the ground distance covered by the uncluttered radio path with earth curvature correction, ucrpc), is set to equal the full length of the distance between the transmitter and the receiver. At the end of the for loop, *dp* is then reduced by the value of the ground distance for the cluttered path that is calculated, (recalculated as the path distance less the ground distance, d_{1a}, or *d1a*, for the cluttered path), and the process repeats until the value of *dp* stabilizes.

ucrpc = sqrt((hone–prop.cch+hc)* (hone–prop.cch+hc)+(dp*dp))

Step 5: calculate the cosine of the angle between the ray leaving the transmitter and vertical; the transmitter incident prime angle; $\cos(\theta_i')$: $\cos(\theta_i') = (h_1 - CH + hc)/ucrpc$ as:

ctip=(hone–cch+hc)/ucrpc

Step 6: calculate the transmitter incident prime angle θ_i': $\theta_i' = \arccos[(h_1 - CH + hc)/ucrpc]$ as:

tip=acos(ctip)

Step 7: The total incident angle with earth curvature correction can be calculated from:

$$(\pi - \theta_{ic}) = (\pi/2 - \theta_i') + (\pi/2 - \theta_{\Delta e})$$

which simplifies to: $\theta_{ic} = \theta_i' + \theta_{\Delta e}$, which is used to calculate the total incident angle θ_{ic} as :

```
tic=tip+tde
```

if in the first line-of-sight range, θ_{ic} is reduced by the value of θ_{ra} , the average ground height receive approach angle calculated in qlrpfl:

```
if(d==prop.dist)
{
        tic-=prop.thera;
}
```

And then limited to be no less than zero:

```
tic=mymax(0.0,tic);
```

Step 8: calculate the sin of the total incident angle; $\sin(\theta_{ic}) = \sin(\theta_i' + \theta_{\Delta e})$, as:

```
stic=sin(tic)
```

Step 9: calculate the sin of the transmission angle, θ_{tc} , using Snell's Law: $\sin \theta_{tc} = (\eta_s / \eta_{cc})(\sin(\theta_{ic})$, as:

```
sta=(ens/encc)*stic
```

Step 10: calculate θ_{tc} : $\theta_{tc} = \arcsin [(\eta_s / \eta_{cc})(\sin(\theta_{ic})]$, as:

```
ttc=asin(sta)
```

Step 11: calculate cos θ_{tc}: $\cos(\theta_{tc}) = [1 - \sin^2(\theta_{tc})]^{1/2}$, as:

```
cttc=sqrt(1-(sin(ttc))*(sin(ttc))
```

Step 12: calculate the length of the cluttered radio path with earth correction, in meters; crpc:
crpc = $(CH - h_2)/ \cos(\theta_{tc})$, as:

```
crpc=(prop.cch-prop.hg[1])/cttc
```

Here we insert an if statement to limit overcalculation of the cluttered radio path when the transmitter is near the top of the clutter canopy, and the path distance is short. This

limits the cluttered radio path to be slightly less than the length of the path between the transmitter and antenna site.

```
If (crpc>=dp)
{
        crpc=dp-1/dp;
}
```

Step 13: calculate the sin of the Snell's Law grazing angle Ψ; $\sin \Psi = (\pi/2 - \theta_{ic})$, as:

```
ssnps=(PI/2)-tic
```

Step 14: calculate the clutter canopy surface distance; d_{1a}: $d_{1a} = crpc(\sin(\theta_{tc}))/(1 - 1/r)$, in meters, as:

```
d1a = (crpc*sin(ttc))/(1-1/6378137)
```

Step 15: Using a for loop, repeat steps 1 to 14 using a new d' = d (actual value) - d_{1a} until the required accuracy is obtained; for spreadsheet or computer calculation, a minimum of three iterations are adequate. The preset of n= can be increased for higher accuracy.

Line (new) `dp=dp-d1a;`

Step 16: calculate the cosine of the total incident angle; $\cos(\theta_{ic}) = \cos(\theta_i' + \theta_{\Delta e})$, as:

```
ctic=cos(tic);
```

a. The next step in preparing coefficients for the calculation is to determine the Reflection, R, and Transmissive, T, coefficients for the antenna polarity being used, at the interface point between the air and the clutter canopy top. This interface acts as a signal splitter; R represents the ratio, as 0 (none) to 1.00 (100%), of the signal that reflects back off of the top of the clutter canopy. T represents the ratio of the signal that passes into, or transmits into, the clutter layer. Newton's Law of Conservation of Energy applies, so T = 1- R. The equations for the reflection coefficient, R, are different for horizontal and vertical polarization. We compute the coefficients using:

$$R_H = [(\eta_s \cos(\theta_i) - \eta_{cc} \cos(\theta_t))/((\eta_s \cos(\theta_i) + \eta_{cc} \cos(\theta_t))]^2$$
$$R_V = [(\eta_s \cos(\theta_t) - \eta_{cc} \cos(\theta_i))/((\eta_s \cos(\theta_t) + \eta_{cc} \cos(\theta_i))]^2$$
$$R_{CP} = (R_H + R_V)/2$$
$$T_H = 1 - R_H$$
$$T_V = 1 - R_V$$
$$T_{CP} = 1 - R_{CP}$$

Selected using ptx, the polarity of the transmitted signal; 0 (or default) for horizontal, 1 for vertical, and (new) 2 for circular. The else statements make horizontal polarity the default mode if 1 or 2 are not selected to denote vertical or circular polarity. The code takes the following form:

```
if (prop.ptx= =1)
{
q=((ens*cttc–encc* ctic)/(ens*cttc+encc* ctic))
rsp=q*q
tsp=1-rsp
}

else
{
        if (prop.ptx= =2)
        {
        q=((ens*ctic–encc*cttc)/(ens*ctic+encc*cttc))
        rsp=((ens*cttc–encc* ctic)/(ens*cttc+encc* ctic))
        rsp=(q*q+rsp*rsp)/2
        tsp=1-rsp
        }
        else
        {
        q=((ens*ctic–encc*cttc)/(ens*ctic+encc*cttc))
        rsp=q*q
        tsp=1-rsp
        }
}
```

b. Here we set *tvsr* to be equal to the difference between the transmitter antenna height and the receive antenna height by subtracting the receive antenna height from the value of *hone*.

```
tvsr=hone-prop.hg[1];
```

c. For a transmitter above the canopy, shooting to a receiver at or below the canopy top, the attenuation to be added to free space dispersion and two –ray multipath cancellation in the line-of-sight mode is determined by one of four sub-modes of the Radiative Transfer Engine: Iri, Id (which includes two sub-modes, I_1 and I_2), and I3.

If the canopy-top distance, equal to the under-canopy top ground distance d_{1a} is less than or equal to 50 meters, then the I_{ri} mode (Beer's Law direct absorption to RTE function transition) controls the results. If d_{1a} is greater than 50 meters, and less than or equal to 225 meters, then the combined I_d mode (I_1 and I_2) controls the results. If d_{1a} is greater than 225

meters, and if the combined incident angle θ_{ic} is equal to or less than 1.5775 radians, then the I3 mode controls the results to the first obstruction, or, if the path is over smooth or slightly irregular terrain, out to the transition point to the past-horizon diffraction mode. We use a series of if statements to implement these break points and enable the performance of the appropriate computation from Shumate's Approximations (see chapter on Shumate's Approximations for a full description of the equations):

```
if (d1a<50.0)
{
    arte=0.0195*crpc –20*log10(tsp)
}
else
{
    if (d1a<225.0)
    {
        if (tvsr>1000.0)
        {
            q=d1a*(.03*exp(-.14*pdk)
        }
        else
        {
            q=d1a*(.07*exp(-.17*pdk)
        }
    arte=q+(0.7*pdk-mymax(0.01,log10(prop.wn*47.7) -
2))*(prop.hg[1]/hone);
    }
    else
    {
      q=0.00055*pdk+log10(pdk)*(0.041–0.0017*sqrt(hone)+0.019);
        arte=d1a*q-(18*log10(rsp))/(exp(hone/37.5);
        zi=1.5*sqrt(hone-prop.cch);
    }
    if ((pdk>zi)
    {
        q=(pdk-zi)*10.2*((sqrt(mymax(0.01,log10(prop.wn*47.7)-
2.0)))/(100-zi));
    }
    else
    {
        q=((zi – pdk)/zi)*(-20.0*mymax(0.01,log10(prop.wn*47.7)-
2.0))/sqrt(hone);
    }
    arte=arte+q;
```

```
            }
        }
}
```

d. If the transmit antenna height above receiver antenna height (hone) is at or below the clutter canopy, then an else statement launches the computation of the combined function equations that are valid only for $h_1 \leq C_{H,}$ for the RTE I_{ri} and I_d modes.

The (C_{AB} + C_{AB2}) terms serve as the intercept point and slope times distance terms of the I_{ri}, straight-line equation. With the addition of a exponential term to fade out the C_{AB2} term contribution as it is undercut by the lesser attenuation of the I_d equation, the I_{ri} terms can then be simply added to the I_d logarithmic function, along with the same frequency and height compensation used for I_d functions above the canopy line.

$A_{RTE\text{-}ABC} = C_{AB} + C_{AB2} + 1.34795*20\log(d_1 +1)$ dB,
where:

$C_{AB} = (C_H - h_1)(2.06943 - 1.56184\exp(C_H - h_1)^{-1})$ dB/meter

and:

$C_{AB2} = (17.98 - .84224(CH-h_1))e^{-0.00061(d1)}$

where d_1, C_H and h_1 are in meters.

And the frequency and height compensation is:

$FC_{Id} = -(\log(f)-2)(h_2/h_1)$

Where the heights are relative to the ground height below the clutter, and h_2/h_1 is the ratio of the receive height AGL to the transmitter height AGL. The frequency and height compensation takes the form:

```
else
{
q=(prop.cch – prop.tgh)*(2.06943 -1.56184*exp(1/prop.cch –prop.tgh))
q=q+(17.98 – 0.84224*(prop.cch-prop.tgh))*exp(-0.00061*pd)
arte=q+1.34795*20*log10(pd +1)
arte=arte– ((log10(prop.wn*47.7)-2)*(prop.hg[1]/prop.tgh)
}
```

lastly, the output of the full line-of-sight RTE results is readied:

```
saalosv=arte
}
return saalosv
}
```

Combined, the c++ code is:

```
double saalos(double d, prop_type &prop, propa_type &propa)
```

```
{
	double ens, encc, s, n, q, dp, tde, hc, dx, ucrpc, ctip, tip, tic, stic, ctic, sta,
	double ttc, cttc, crpc, ssnps, d1a, rsp, tsp, arte, zi, pd, pdk, hone;
	double saalosv;

	q=0.0;
	if (d== 0.0)
	{
		tsp=1.0;
		rsp=1.0;
		saalosv=0.0;
	}
	else
	{
		pd=d;
		pdk=pd/1000.0;
		tsp=1.0;
		rsp=0.0;
		dla=pd;

		if(prop.tgh>prop.cch)
		{
			ens=1+prop.ens*0.000001;
			encc=1+prop.encc*0.000001;
			dp=pd;

			for (int j=0; j<5; ++j)
			{
				tde=dp/6378137;
				hc=(prop.cch+6378137)*(1– cos(tde));
				dx=(prop.cch+6378137)*sin(tde);
				hone=prop.tgh+prop.tsgh-prop.rch[1];
				ucrpc=sqrt((hone–prop.cch+hc)*(hone–
				prop.cch+hc)+(dx*dx));
				ctip=(hone–cch+hc)/ucrpc;
				tip=acos(ctip);
				tic=tip+tde;

				if(d==prop.dist)
				{
					tic-=prop.thera;
				}

				tic=mymax(0.0,tic);
				stic=sin(tic);
				sta=(ens/encc)*stic;
```

```
        ttc=asin(sta);
        cttc=sqrt(1-(sin(ttc))*(sin(ttc)));
        crpc=(prop.cch-prop.hg[1])/cttc;
        ssnps=(3.1415926535897/2)-tic;
        d1a=crpc*sin(ttc))/(1-1/6378137);
        dp=dp-dia;
}

ctic=acos(tic);

if (prop.ptx>=1) /* polarity ptx is vertical or circular */
{
        q=((ens*cttc-encc*ctic)/(ens*cttc+encc*ctic));
        rsp=q*q;
        tsp=1-rsp;

        if (prop.ptx==2)  /* polarity is circular */
        {
                q=((ens*ctic–encc* cttc)/(ens*ctic+encc* cttc));
                rsp=((ens*cttc-encc*ctic)/(ens*cttc+encc*ctic));
                rsp=(q*q+rsp*rsp)/2;
                tsp=1-rsp;
        }
}
else  /* ptx is 0, horizontal, or undefined */
{
        q=((ens*ctic–encc*cttc)/(ens*ctic+encc*cttc));
        rsp=q*q;

        tsp=1-rsp;
}

hone=prop.hg[1];

if (d1a<50.0)
{
        arte=0.0195*crpc –20*log10(tsp);
}
else
{
        if (d1a<225)
{

if (hone>1000)
{
        q=d1a*(.03*exp(-.14*pdk);
```

```
                        }
                        else
                        {
                                q=d1a*(.07*exp(-.17*pdk);
                        }
                        arte=q–((log10(prop.wn*47.7)-2)*(prop.hg[1]/hone);
                }
                else
                {
                        q=0.00055*pdk+log10(pdk)*(0.041–0.0017*sqrt(hone)+0.019);
                        arte=d1a*q-(18*log10(rsp))/(exp(hone/37.5));
                        zi=1.5*sqrt(hone-prop.cch);

                        if (pdk>zi)
                        {
                        q=(pdk-zi)*10.2*((sqrt(mymax(0.01, log10(prop.wn*47.7)-
2.0)))/(100-zi));
                        }
                        else
                        {
                        q=(zi-pdk)/zi)*(-20*mymax(0.01,log10(prop.wn*47.7)-
2.0))/sqrt(hone);
                        }
                        arte=arte+q;
                    }
                }
            }
            else
            {
            q=(prop.cch – hone)*(2.06943 –1.56184*exp(1/prop.cch – hone));
            q=q+(17.98 – 0.84224*(prop.cch-hone))*exp(-0.00061*pd);
            arte=q+1.34795*20*log10(pd +1);
            arte=arte–((log10(prop.wn*47.7)-2)*(prop.hg[1]/hone));
        }
    saalosv=arte;
   }
  propa.test0=saalosv;
  return saalosv;
}
```

SUBROUTINE ZLSQ1: A functional explanation, by Sid Shumate.
Last Revised January 27, 2009.

Z1SQ1 Subroutine;
The Linear Least Squares Fit between X1, X2 to the function described by Z--.

Note: Used with point-to-point prediction and area prediction modes.

Called by ***dlthx***, while ***dlthx*** is being called by ***qlrpfl*** , after which ***qlrpfl*** may call ***zlsq1*** directly.

Derived from the Irregular Terrain Model description by George Hufford, 2002, (ITMD), and subroutine descriptions in Appendix A to "A Guide to the Use of the ITS Irregular Terrain Model in the Area Prediction Mode", 1982, Hufford, Longley & Kissick, compared to the ITM.cpp prepared by J. D. McDonald and John Magliacane for compilation on unix and linux systems. "Line" numbers refer to the ITM.cpp as line numbered by Bloodshed Software's DevC++ print function. "Alg" numbers refer to the algorithm formula in "The ITS Irregular Terrain Model, version 1.22, the Algorithm" by G. A. Hufford, 1995. For the background discussion of the Linear Least Squares Fit solution, refers to equations in Chapter 15.2 of *"Numerical Recipes in C, Second Edition"* ©Cambridge University Press.

Background Notes on the Linear Least Squares Fit.

A linear least squares fit subroutine implements a mathematical procedure for finding the best-fitting curve to a given set of points by minimizing the sum of the squares of the offsets ("the residuals") of the points from the curve. The name of this subroutine suggests the use of the linear least squares fitting methodology, the simplest and most commonly applied form of linear regression. This methodology provides a solution to the problem of finding the best fitting line through a set of points. In a linear least squares fit, vertical least squares fitting proceeds by finding the sum of the squares of the vertical (y-axis) deviations of the squares of the deviations of the function values (in this case, elevations) from a straight line, along the x-axis from j=0 to j=n. The square deviations from each elevation point are therefore summed and the resulting residual is then minimized to find the best-fit line. When used in a simple mode, to find the best fitting straight line through a set of points, the process provides a solution for a, an intercept value, and b, the slope value, in the straight line equation y = a +bx .

In Tech Note 101, this equation becomes:

$$h(x) = \cancel{h} + m(x - \cancel{x}) \qquad (5.15a)$$

Where the h(x) term replaces the y, $\cancel{h}$ replaces the intercept variable a, and the m replaces the b slope variable. The term $(x - \cancel{x})$ refers to the location of the reference

zero crossing, where x = 0, being relocated to a position at the center, or midpoint, of the x path.

For a full description of a set of equations used in the methodology, see Chapter 15.2, *Fitting Data to a Straight Line*, in the book "*Numerical Recipes in C, Second Edition"* ©Cambridge University Press (Numerical Recipes). A weakness of the least squares procedure is that it results in outlying points being given disproportionately large weighting. However, if one is dealing with equal-width intervals, and, if we do not know the individual measurement errors, we assume that the individual measurement error factors are equal, and if we set the x,y zero crossing at the midpoint of the path, i.e. use an "x" equidistant path function that causes x to range in value from xi = ((-xa/2)+1) to xi = ((xa/2)-1) as the for loop cycles, causing "Sx"=0.0) then, solving for "a" at the midpoint of the values of "x" along the x axis (the section of the path considered), the formulas specified in 15.2.1 through 15.2.6 of Numerical Recipes simplifies so that "a" is equal to Sy, the sum of the "y" axis data point values, (elevation values along the path, divided by S, the number of "x" values (the number of intervals, represented by the argument "*xa*"). This avoids the need to use the square of the y values in the solution, minimizing the disproportionately large weighting of extremely high or low elevation values due to the squaring of the values.

The formula to solve for "a", given the pre-conditions, simplifies to a = Sy/S, i.e.:

a = (sum of elevation values along the path)/(number of intervals)

This subroutine analyzes a central section of the total path between the tx site and the receive site, starting a short distance, set in part 2, below, from the tx site, and ending a short distance, set in part 3 below, before the receive site. This central section of the path is referred to below as the "section of the path considered".

The "a" term is solved at the point where x = 0, at the midpoint of the path considered, and is later adjusted to be the "a" value where x = 0 at the endpoint of the path considered. The terms z[0], the y value at the transmitter site, is then calculated by solving z[0]= y = a +xb where x is at the transmitter site, and then solved for z[n] = y = a +xb where x is at the receive site; the program outputs the values of z[0] and z[n].

In the same way that the "a" solution formula simplifies, the "b" solution formula given in 15.2.6 of Numerical Recipes can be simplified. To provide more detail for "b" than we did for "a":

$$b = \frac{S*Sxy - Sx*Sy}{Delta} = \frac{S*Sxy - Sx*Sy}{S*(Sxx) - (Sx)^2} \qquad \text{(15.2.6, Numerical Recipes)}$$

Where: (See 15.2.1 through 15.2.5, Numerical Recipes)

Sx simplifies to be = 0.0, as the sum of the negative terms in the x equidistant function cancel out the positive terms as x progresses from - xa/2, through 0, to xa/2.

Sxx simplifies to be = (sum of the squares of xi). This is a non-zero number, as the squares of the negative values of the x equidistant function, i.e. the individual incremental values of xi^2 from xi = ((–xa/2)+1) to zero, are positive values.

Sxy simplifies to be = (sum of x times **y** along the section of the path considered). We will refer to this as: **(**sum of **xi***y). Here x is the x equidistant function values, and y is the elevation values. Sxy is equal to the value of *b* at the completion of the ***for*** loop.

Since S/S = 1, the formula for b then simplifies to: b = Sxy/Sxx, i.e.:
b = **(**sum of **x***y)/(sum of the squares of x)

As the value of x increments from –x to +x in the ***for*** loop. This solves b, the slope for the straight line equation y = a +bx. However, this is not quite what the subroutine does.

The calculation of a, the intercept point, in ***z1sq1*** directly implements the a = Sy/S formula above, using the ***for*** loop to sum the y values. But it divides it by xa, the number of intervals.

DdNeeds work…see spring 09 article.

The step-by-step subroutine analysis follows.

From ITMD Sections 45 and 53:

Call inputs:

z[ ]	a.k.a. z,	z(J+2), J=0,…,en, Function Values.
&x1	a.k.a. x1 (x-one)	
&x2	a.k.a. x2	
& z0	a.k.a. z0	= xi, interval length
& zn	a.k.a. z1, (z-one)	= en, number of intervals

Array *z* must have a special format;

z[0] = en, the number of equally large intervals,
z[1] = xi, a.k.a. sigma, the interval length,
z(j+2), j=0,….n, function values.

local declarations:

xn number of intervals

xa at first, the number of intervals in distance x1 (distance between transmitter site and "start point on the path"); later, the path length, in intervals, between the start point and the end point on the path considered by the for loop.

xb at first, the number of intervals in distance x2 (distance between transmitter site and "end consideration location"). Later, changed to be the distance from the transmitter site to the midpoint of the path considered by the for loop.

x working variable

a working variable used to calculate "a", the intercept variable of formula y = a +xb; after for loop ends, contains sum of elevations along the considered section of the path.

b working variable used to calculate "b", the slope variable of formula y = a +xb.

n number of intervals in the section of the path considered in the for loop.

ja location of for loop consideration along path, counted in intervals from the transmitter site.

jb number of intervals in distance between transmitter site and "end consideration location")

This subroutine, ***z1sq1*** ;

1. Defines xn=z[0], setting xn = the total number of intervals between the transmitter site and the receive site, the two terminals.

Line 1117 xn=z[0];

2. Calculates value for xa by calling FORTRAN_DIM(xl/z[1],0.0)

Note: The FORTRAN_DIM function receives inputs (&x, &y)
And reports out x-y if x is greater than y; otherwise the reported result is 0.0.
So if xl/z[1] >0.0, xa=xl/z[1]-0.0, if xl/z[1] =<0.0, xa = 0.0.

x1 (x-one) is the distance between the transmitter site and the start point for consideration, in meters. At one point set to be the lesser of 1/10 of the path distance or 1/15th of the transmitter HAGL. *z[1]* is the length of one interval, so *x1/z[1]* is the number of intervals in *x1* between the transmitter site and the start point for consideration. So *xa* starts out as a "double" argument holding an integer value representing the number of terrain data intervals that can fit between the transmitter site and the start point of the path considered in the for loop.

Line 1118: xa=int(FORTRAN_DIM(xl/z[1],0.0));

3. Calculates value for *xb* using FORTRAN_DIM function.
If xn is > x2/z[1], *xb*=xn-(xn-x2/z[1]), i.e. x2/z[1]. Otherwise, *xb* =xn-0.0.

x2 is the distance between the transmitter site and the end point for consideration, at one point set to be shorter than the total transmit to receive path length by the lesser of 1/10 of the path distance or 1/15th of the receive antenna HAGL. *z[1]* is the length of one interval, so *x2/z[1]* is the number of intervals in *x2* between the transmitter site and the end point for consideration. *xn* is the total number of intervals in the path between the transmitter site and the receive site. So *xb* represents the number of intervals between the transmitter site and the end point for consideration.

Line 1119: xb=xn-int(FORTRAN_DIM(xn,x2/z[1]));

4. An **if** statement states that if xb is less than or equal to xa, which indicates that the total path is a very short one that causes the start point for consideration to be at or past the end point for consideration, then:

   a. If xa is greater than 1, the value of xa becomes equal to xa-1, if not, xa = 0.0. This reduces the value of xa by 1, unless xa is already 1 or less. If xa is 1 or less, xa is set to be equal to zero.
   b. If xn is greater than (xb+1), i.e. if the total path length (defined in number of intervals) is longer than the distance from the transmitter site to the end point for consideration plus 1 interval, the value of xb is reset to be equal to xn-(xn-(xb+1)), or xb+1, increasing the value of xb by 1.
   c. if xn is not greater than (xb+1), i.e. if the total path length is not longer than the distance from the transmitter site to the end point for consideration plus 1 interval, xb is reset to equal xn-0.0, i.e. = xn, the

number of intervals in the total path length, effectively setting the end point for consideration to be at the receive site.

Line 1123: xa=FORTRAN_DIM(xa,1.0);
Line 1124: xb=xn-FORTRAN_DIM(xn,xb+1.0);

5. The value of *ja* is preset to be equal to xa.
6. The value of *jb* is preset to be equal to xb.
7. The value of *n* is set to be equal to *jb-ja*, the number of intervals between the start point and the end point of path considered by the for loop.
8. The value of *xa* is reset to be equal to *xb-xa*, resetting it to now represent the number of intervals between the start point and the end point of path considered by the for loop.
9. The value of *x* is set to equal –0.5*xa, the negative of half the number of intervals in the section of the path for consideration.
10. The value of *xb* is increased by the value of *x.* (In fact, shortens it, as x is a negative number).
11. The value of *a* is set to be 0.5*(z[ja+2]+z[jb+2]); this presets *a* to be equal to (the elevation of the start point+the elevation of the end point)/2, or the average of the start point and the end point.
12. The value of *b* is set to be .5*(z[ja+2]- z[jb+2])*x ; this presets *b* to be equal to ½ of the difference between the elevation of the start point and the elevation of the end point, multiplied by x. The value of x at this point is a negative value, the absolute value of which is equal to one half of the distance, measured in intervals, between the start point and the end point of the path considered by the for loop.

13. A **for** loop is started at line 1136, starting at i=2, continuing until argument *i* is no longer < n. For each pass, i is incremented (increased by one). The loop then continues with:
    a. ++*ja* This increments *ja* so that the incremented value can be immediately used. *ja* starts at the number of intervals between the transmitter site and the start point of the for loop's consideration of the path elevations, and increases by one path interval for each for loop pass, so the for loop starts with ja being increased to the value of one interval past the start point.
    b. *x* is increased by 1. At the beginning of the for loop, x starts at –(xa)/2, or minus ½ of the number of intervals in the section of the path considered, and has 1 (increment) added to it for each for loop pass, prior to the calculation of a and b, which causes it to end up equal to ((*xa*/2)-1), or one-half of the number of increments between the start point and the end point of the section of path considered, less one, when the ***for*** loop stops one increment shy of the end point. This line generates the values of the x function, x = - xa/2 +n+1, incrementing x from –x to x as n progresses from 2 to the value of (xa –1).
    c. *a* is increased by *z[ja+2]*. *z[ja+2]* is the elevation of the point being considered in this loop; as the for loop continues, this causes *a* to sum the

elevation values at each interval between the start and end points (not including the start and end points).

d. *b* is increased by *z[ja+2]*x*; The *z[ja+2]* term is the elevation of the point being considered in this loop; as the ***for*** loop continues, and x proceeds in value from –1/2 of the path intervals to ½ of the path intervals, this causes b to sum the result of the interval elevations along the path multiplied by x; the end result is a positive or negative number indicating a weighted bias of the sum of the elevations around the midpoint of the path. A negative sum (bias) for b indicates the transmitter end of the path has the higher weighted average of elevations, and a positive sum (bias) indicating the receiver end of the path has the higher weighted elevation average.

14. Once the ***for*** loop ends, the value of "*a*" is reset to be equal to *a* divided by *xa*. The value of *a* then represents the sum of the elevations at each interval along the path length considered, divided by the number of intervals; i.e. the average of the elevations along the path considered, and represents the value of "a" in y = a +xb, solved for the condition where x is equal to 0.0 at the midpoint of the path considered. "a" will be the same for all points along the path, unless the intercept point (where x = 0) is reset. Which it will be, to the end point of the path considered, before solving for z0 and zn

Line 1144: a/=xa;

15. Now we need to solve for b, the slope of the straight-line formula y = a +xb. Here we are dealing with an even number of equal-width intervals; The individual measurement errors are expected to be equal, and we also use the "x" equidistant function (x ranges in value from = (-xa/2)+1 to (xa/2)-1 as the for loop cycles, causing "Sx"=0.0). As discussed above in (14.) and the introductory section "Background on the Linear Least Squares Fit", in solving for "b", the slope value, the formulas specified in 15.2.1 through 15.2.6 of Numerical Recipes simplifies so that "b" is equal to the sum of the "x times y" axis data point values, represented by the argument "b" value at the completion of the for loop), multiplied by the number of intervals (xa), and divided by the sum of the squares of the x values from the equidistant function along the path considered. So the formula to solve for "b", given the pre-conditions, should be:

b = (sum of elevation values multiplied by the x equidistant function value) *(N, the number of intervals)/(sum of the squares of the x equidistant function values)

The "sum of elevation values multiplied by the x equidistant function value", is equal to the value of *b* at the completion of the ***for*** loop.

The number of intervals, N, is equal to the value of *xa*.

But the subroutine fails to calculate the sum of the squares of the x equidistant function values along the path.

Now here we have a mystery, and perhaps errors in the original coding of the linear least squares fit algorithm:

16. At line 1145, the term "b" is "solved" to be equal to b*12.0/((xa*xa+2.0)*xa). The argument *xa* is the number of intervals along the path considered. The b value on the right hand side of the equation is the sum of each individual elevation along the path considered, multiplied by the x equidistant function value at the elevation point's increment point.

I believe the 12/((xa*xa+2)*xa) term is incorrect.

The "12" number appears to replace the term "N" in the formula for "b" stated in (15.) above. If so, the b formula could only be correct if N = 12; i.e. if the path considered had only a fixed number of intervals = 12.

The term "(xa*xa+2)*xa" appears to be an attempt to replace the Sxx term in the b formula, as derived in the introductory section "Background on the Linear Least Squares Fit" above. "(xa*xa+2)*xa" appears to attempt to replace the Sxx term, the (sum of the squares of the x equidistant function values, xi) in the b formula from "Numerical Recipes", by taking the square of the number of intervals, xa, adding 2, and multiplying by the number of intervals, to create a sum of the squares of the number of intervals. The "2" appears to have been included to compensate for the fact that the ***for*** loop starts at *i* = 2, not *i* = 0, and increments *ja* and *x* before the *a* and *b* calculations, causing the ***for*** loop to ignore the start point elevation and the end point elevation of the section of the path considered. This calculation fails, in that the (square of the number of intervals, plus 2), (multiplied by the number of intervals), is not the same as, is not equal to, and provides a larger calculated value than, the sum of the squares of the sequence of x values from the x equidistant function, this sequence of values defined as xi = - xa/2 incremented by 1 per elevation increment, up to (xa/2)-1 .

As a result, the value of b is generally understated. Therefore, there appear to be two problems with the subroutine; If N, the number of intervals in the section of the path considered, is not equal to 12, b is incorrect; if N =12, the substitute for the Sxx term gives a larger value, causing b to be significantly understated. See attached worksheet file for an analysis of the amount understated by length of path.

The b value now attempts to represents the b term, or slope, of the straight-line formula y = a +b*x where x = 0 at the end point of the path considered in the ***for*** loop.

17. The value of *z0*, the y-axis value of the equation line formula y = a +b*x, is calculated to be equal to *a-(b*xb)*. The value of *a* represents the average of the elevations along the path considered. The - (b*xb) term is the slope b times the

distance xb (in increments) between the transmitter site to the end point of the path considered. *z[0]* then equals the value of y = a +b*x at the transmitter site. But since the "b" value is understated, as noted above, this value is incorrect. The b slope is understated, causing the line between z0 to zn to be "flattened" (to approach the a value) from the true value.

18. The value of *zn* the y-axis value of the equation line formula (y= a +b*x) where x is at maximum (receive site) value, is calculated to be *a+(b*(xn-xb))*.
The value of *a* represents the average of the elevations along the path considered. The value of *xn* represents the number of intervals in the total path length. The value of *xb* represents the distance, measured in intervals, from the transmitter to the end point of the path considered by the ***for*** loop. So the term b*(*xn-xb)* term is the slope b times the distance (in increments) between the end point of the path considered, where x = 0, to the receive site. *z[0]* then equals the value of y = a +b*x at the receive site. But since the "b" value is understated, as noted above, this value is incorrect.

19. Outputs:

The start and end y-axis (z) values of the required line:
z0 at 0, the transmitter end of the path considered, and
zn at *x[t] =xi*en, or= z[1]*z[2],* the receive end of the path considered.

x1 and x2 are two-way arguments, both used as input and restated in the output.

How can the calculation errors in b be corrected? By:

a. *Add a value-preset local variable, xi=0.0, to the* ***double*** arguments *declaration line, at Line 1114.*

b. *Insert the line: xi+=(x*x); after line 1139, (x+=1.0;) in the* ***for*** loop. At the completion of the ***for*** loop cycles, xi will represent the sum of the squares of the xi values, equal to the Sxx term in the b solution formula discussed in the "Background Notes on the Linear Least Squares Fit" section above.

c. *Replace line 1145,* b = b*12.0/((xa*xa+2.0)*xa); with b = (b*xa)/xi;

SUBROUTINE Z1SQ2: A functional explanation, by Sid Shumate.
A 2nd revision of z1sq1 created on Sept. 30, 2008
Last Modified Oct 18, 2008.

Z1SQ2 Subroutine ***zlsq2 z-one SQ - two***

The Linear Least Squares Fit between X1, X2 to the function described by Z--.

Note: Used with point-to-point prediction and area prediction modes.

Called by ***d1thx***, while ***d1thx*** is being called by ***qlrpfl2***, after which ***qlrpfl2*** may call ***z1sq2*** directly.

Derived from the Irregular Terrain Model description by George Hufford, 2002, (ITMD), and subroutine descriptions in Appendix A to "A Guide to the Use of the ITS Irregular Terrain Model in the Area Prediction Mode", 1982, Hufford, Longley & Kissick, compared to the ITM.cpp prepared by J. D. McDonald and John Magliacane for compilation on unix and linux systems. "Line" numbers refer to the ITM.cpp as line numbered by Bloodshed Software's DevC++ print function. "Alg" numbers refer to the algorithm formula in "The ITS Irregular Terrain Model, version 1.22, the Algorithm" by G. A. Hufford, 1995. The background discussion of the Linear Least Squares Fit solution refers to equations in Chapter 15.2 of *"Numerical Recipes in C, Second Edition"* ©Cambridge University Press.

Background Notes on the Linear Least Squares Fit.

A linear least squares fit subroutine implements a mathematical procedure for finding the best-fitting curve to a given set of points by minimizing the sum of the squares of the offsets ("the residuals") of the points from the curve. The name of this subroutine suggests the use of the linear least squares fitting methodology, the simplest and most commonly applied form of linear regression. This methodology provides a solution to the problem of finding the best fitting line through a set of points. In a linear least squares fit, vertical least squares fitting proceeds by finding the sum of the squares of the vertical (y-axis) deviations of the function values (in this case, elevations multiplied by the distance in intervals from a midpoint of the path considered) from a straight line, along the x-axis from j=0 to j=n. The deviations from each elevation point are therefore summed and the resulting residual is then minimized to find the best-fit line. When used in a simple mode, to find the best fitting straight line through a set of points, the process provides a solution for a, an intercept value, and b, the slope value, in the straight-line equation y = a + bx.

In Tech Note 101, this equation becomes:

$$h(x) = \bar{h} + m(x - \bar{x}) \qquad (5.15a)$$

Where the h(x) term replaces the y, ~~h~~ replaces the intercept variable a, and the m replaces the b slope variable. The term (x – ~~x~~) refers to the location of the reference zero crossing, where x = 0, being relocated to a position at the center, or midpoint, of the x path.

For a full description of a set of equations used in the methodology, see Chapter 15.2, *Fitting Data to a Straight Line*, in the book "*Numerical Recipes in C, Second Edition*" ©Cambridge University Press (Numerical Recipes). A weakness of the least squares procedure is that it results in outlying points being given disproportionately large weighting. However, if dealing with an even number of equal-width intervals where the individual measurement errors are not known, the uncertainty associated with each measurement can be set to be equal to 1, and the x,y zero crossing can be set at the midpoint of the path.

This uses an "x" equidistant path function that causes x to range in value from xi = ((-xa/2)+1) to xi = ((xa/2)-1) as the for loop cycles, causing "Sx"=0.0. By then solving for "a" at the midpoint of the values of "x" along the x axis (the section of the path considered), the formulas specified in 15.2.1 through 15.2.6 of Numerical Recipes simplify from:

$$\Delta = SS_{xx} - (S_x)^2$$

$$a = (SS_{xx}S_y - S_xS_{xy})/\Delta \qquad (15.2.6)$$

$$b = (SS_{xy} - S_xS_y)/\Delta$$

to:

$$\Delta = SS_{xx}$$

$$a = (SS_{xx}S_y)/\Delta = (SS_{xx}S_y)/SS_{xx} = S_y/S$$

$$b = (SS_{xy} - S_xS_y)/\Delta = (SS_{xy})/SS_{xx} = S_{xy}/S_{xx}$$

so that "a" is equal to Sy, the sum of the "y" axis data point values, (elevation values along the path), divided by S, the number of "x" values (the number of intervals, represented by the argument "*xa*"). This avoids the need to use the square of the y values in the solution, minimizing the disproportionately large weighting of extremely high or low elevation values due to the squaring of the values.

So the formula to solve for a, now "~~h~~", given the pre-conditions, simplifies to:

$$\cancel{h} = a = S_y/S, \text{ i.e.:}$$

~~h~~ = a = (sum of elevation values along the path)/(number of intervals)

The intercept value, when located at the center of the path, is equal to the average value of the elevation along the path.

This subroutine analyzes a central section of the total path between the tx site and the receive site, starting a short distance, set in part 2, below, from the tx site, and ending a short distance, set in part 3 below, before the receive site. This central section of the path is referred to below as the "section of the path considered".

The "a" term is solved for a zero intercept at the center of the path considered, such that x = 0 at the midpoint of the path considered. The terms z[0], the y value at the transmitter site, is then calculated by solving z[0]= y = a +xb where x is at the transmitter site, and then solved for z[n] = y = a +xb where x is at the receive site; the program outputs the values of z[0] and z[n].

Again, S_x simplifies to be = 0.0, as the sum of the negative terms in the x equidistant function cancel out the positive terms as x progresses from - xa/2, through 0, to xa/2. Then the "b" solution formula given in 15.2.6 of Numerical Recipes simplifies to:

$$b = (SS_{xy} - S_xS_y)/\Delta = (SS_{xy})/SS_{xx} = S_{xy}/S_{xx}$$

Sy simplifies to be = (sum of the elevation values along the section of the path considered). This term is irrelevant when S_x is equal to zero.

Sxy simplifies to be = (sum of x times **y** along the section of the path considered). We will refer to this as: **(**sum of **xi***y). Here x is the x equidistant function values, and y is the elevation values. Sxy is equal to the value of *b* at the completion of the ***for*** loop.

Sxx simplifies to be = (sum of the squares of xi). This is a non-zero positive number, as the squares of the negative values of the x equidistant function, i.e. the individual incremental values of xi^2 from xi = ((–xa/2)+1) to zero, are positive values.

Inserting the rest of the simplified S formulas:

$$b = S_{xy}/S_{xx} = (\text{sum of } \mathbf{xi}*y) / (\text{sum of the squares of } xi)$$

Where xi is the value of the x equidistant function, or xi = - xa/2 +(n-2+1), as n progresses in value from 2 to xa-1, and xi progresses in value from (- xa/2)+1 to (xa/2)-1, in the ***for*** loop, where xa is an even integer.

The argument b also includes the values of (**xi***y)/(xi*xi) for the endpoints, computed and preset before the summation for loop.

This solves b, the slope factor for the straight line equation y = a +bx .

However, this is not what the original subroutine ***z1sq1*** does; this has been corrected for the ITWOM version, ***z1sq2***.

The step-by-step subroutine analysis follows:

From ITMD Sections 45 and 53:

Call inputs:

z[ ]	a.k.a. array z,	z(J+2), J=0,…,en,	Function Values.

Array *z* must have a special format;

z[0] = en, the number of equally large intervals,
z[1] = xi, a.k.a. sigma, the interval length,
z(j+2), j=0,….n, function values.

&x1	a.k.a. x1 (x-one)	distance in meters, from transmit site to start of path considered.
&x2	a.k.a. x2	distance in meters, from rcvr site to end of considered path.

Outputs:

& z0	a.k.a. z0	= height value of transmit end of line, in meters.
& zn	a.k.a. z1, (z-one)	= height value of receive end of line, in meters.

local declarations:

xn	number of intervals
xa	first the number of intervals in distance x1 (distance between transmitter site and "start consideration location"); later, the path length of the portion of the path considered by the for loop measured in number of intervals.
xb	number of intervals in distance x2 (distance between transmitter site and "end consideration location")
x	working variable; first represents the value of x at the starting point of the intervals considered, that are located between the end points.
a	working variable used to calculate "a", the intercept variable of formula y = a +xb; after for loop ends, contains sum of elevations along the considered section of the path.
b	working variable used to calculate "b", the slope variable of formula y = a +xb.
n	number of intervals in the section of the path considered in the for loop.
ja	location of for loop consideration along path, counted in intervals from the transmitter site.
jb	number of intervals in distance between transmitter site and "end consideration location")

This subroutine, ***zlsq2*** ;

1. Defines xn=z[0], setting xn = the total number of intervals between the transmitter site and the receive site, the two terminals.

Line 1117 xn=z[0];

2. Calculates value for xa, by calling FORTRAN_DIM(xl/z[1],0.0)

Note: The FORTRAN_DIM function receives inputs (&x, &y)
And reports out x-y if x is greater than y; otherwise the reported result is 0.0.
So if xl/z[1] >0.0, xa=xl/z[1]-0.0, if xl/z[1] =<0.0, xa = 0.0.

x1 (x-one) is the distance between the transmitter site and the start point for consideration, at one point set to be the lesser of 1/10 of the path distance or 1/15th of the transmitter HAGL. The array value *z[1]* is the length of one interval in meters, so *x1/z[1]* is the number of intervals in *x1* between the transmitter site and the start point for consideration. So *xa* represents the number of intervals between the transmitter site and the start point for consideration.

Line 1118: xa=int(FORTRAN_DIM(xl/z[1],0.0));

3. Calculates value for *xb* using FORTRAN_DIM function.
If xn is > x2/z[1], *xb*=xn-xn-x2/z[1]. Otherwise, *xb* =xn-0.0.

The argument *x2* is the distance, in meters, between the transmitter site and the end point for consideration, in meters. For one calculation, this is preset in ***qlrpfl*** to be shorter than the total transmit to receive path length by the lesser of 1/10 of the path distance or 1/15th of the receive antenna HAGL. *z[1]* is the length of one interval in meters, so *x2/z[1]* is the number of intervals in *x2* between the transmitter site and the end point for consideration. *xn* is the total number of intervals in the path between the transmitter site and the receive site. The FORTRAN_DIM function provides the number of intervals between the receive site and the end point for consideration. If the distance, in intervals, from the transmitter site to the end point for consideration is equal to or greater than the total path distance in intervals, the FORTRAN_DIM function result is 0.0. The result of the FORTRAN_DIM function is then subtracted from *xn*, so *xb* represents the number of intervals between the transmitter site and the end point for consideration, where the end point may equal, but not exceed, the receive site distance.

Line 1119: xb=xn-int(FORTRAN_DIM(xn,x2/z[1]));

4. An **if** statement states that if xb is less than or equal to xa, which indicates that the total path is a very short one that causes the start point for consideration to be at or past the end point for consideration, then:

a. If xa is greater than one interval, the value of xa becomes equal to xa-1, if not, xa =0.0. This reduces the value of xa by one interval, unless xa is equal to or less than one; if xa is equal to or less than one, xa becomes 0.0.
b. If xn is greater than (xb+1), i.e. if the total path length (defined in number of intervals) is longer than the distance from the transmitter site to the end point for consideration plus 1 interval, the value of xb is reset to be equal to xn-(xn-(xb+1)), or xb+1, increasing the value of xb by 1.
c. if xn is not greater than (xb+1), i.e. if the total path length is not longer than the distance from the transmitter site to the end point for consideration plus 1 interval, xb is reset to equal xn, the number of intervals in the total path length, effectively setting the end point for consideration to be at the receive site.

```
Line 1123:    xa=FORTRAN_DIM(xa,1.0);
Line 1124:    xb=xn-FORTRAN_DIM(xn,xb+1.0);
```

5. The value of *ja* is preset to be equal to the integer value of xa.
6. The value of *jb* is preset to be equal to the integer value of xb.
7. The value of *xa* is reset to be equal to *xb-xa*, the number of intervals in the section of the path for consideration. *xa* at this point must also be an integer number and an odd number, to provide an even number of intervals below and above a midpoint zero crossing. The end points will be separately considered, so the number can be reduced by 2, and the full adjustment is: *xa* =-2+1+2*int((xb-xa)/2), or (2*int((xb-xa)/2))-1.

8. The argument x must be an even integer number, to provide an even number of intervals below and above a midpoint zero crossing, resulting in an odd total number of terrain points considered, one of which equals zero (*x* times the elevation at *x*=0). Therefore, the value of *x* must be an even integer number, so that the value of *x* determined below will be an even integer number of intervals. The value of *x* is set to equal –0.5* (*xa*+1), the negative integer value of half the number of intervals in the section of the path for consideration, plus 1.

9. The value of *xb* is summed with the value of *x*, a negative number. The result is that *xb* is shortened, and will now represent the value of the number of intervals from the transmitter site to the midpoint of the path to be considered.

10. The value of *ja*, the distance from the transmitter to the start point of the path considered, is reset (increased if necessary) to match *jb*, the distance from the transmitter to the end point of the path considered, less the integer value of (*xa+1)*, the number of intervals to be considered between ja and jb, the end points, plus one.

11. The value of *n* is set to be equal to *jb-ja*, the number of intervals between the end points of the section of the path for consideration, since *i* starts at 2 in the for loop and stops when *i* increases in value to match n.

12. The value of *a* is set to be (z[ja+2]+z[jb+2]); this presets *a* to be equal to the elevation of the start point+the elevation of the end point, taking into consideration the start point and the end point.

13. The value of *b* is set to be (x)(z[ja+2])+(-x)(z[jb+2]); this presets *b* to be equal to the end point elevations times the distance (in intervals) to the midpoint.

14. xi, the sum of the square of the x distances from the midpoint, is preset to be equal to square of the x value at each endpoint, equal to two times the square of (x), calculated as $(x)^2 + (x)^2$.

15. A **for** loop is started at line 1136, starting at i=2, continuing until i is no longer < n. For each pass, i is incremented (increased by one), and then:
    a. *++ja* This increments *ja* so that the incremented value can be immediately used. *ja* starts at the number of intervals between the transmitter site and the start point of the for loop's consideration of the path elevations, and increases by one path interval for each for loop pass.
    b. *x* is increased by 1. At the beginning of the for loop, x starts at –(xa-1)/2, or minus ½ of the number of intervals in the section of the path considered less one, and has 1 (increment) added to it for each for loop pass, prior to the calculation of a and b, which causes it to end up equal to *(xa-1)/2)*, or one-half of the number of increments between the start point and the end point of the section of path considered less one, when the ***for*** loop stops one increment shy of the end point. This line generates the values of the x function, x = x progressing up one interval at a time to –x, as *i* progresses from 2 to the value of n –1.
    c. The argument *xi*, adds the sum of the squares of the x value for each loop to the existing squares of the x value of the endpoints.
    d. *a* is increased by *z[ja+2]*. *z[ja+2]* is the elevation of the point being considered in this loop; as the for loop continues, *a* is increased by the elevation of each interval, becoming the sum of the elevations along the path considered.
    e. *b* is increased by *z[ja+2]*x*; The *z[ja+2]* term is the elevation of the point being considered in this loop; as the ***for*** loop continues, and x proceeds in value from –1/2 of the path intervals to ½ of the path intervals, this causes b to collect the sum of the interval elevations along the path multiplied by x; the end result is a positive or negative number indicating a weighted bias of the sum of the elevations around the midpoint of the path, with a negative sum (bias) for b indicating the transmitter end of the path has the higher weighted average of elevations, and a positive sum (bias) indicating the receiver end of the path has the higher weighted elevation average. The varying value of x weights the averages toward the beginning and end

of the path, with the midpoint, where x passes through zero, having no effect, as the elevation is multiplied by zero.

16. Once the ***for*** loop ends, the value of "*a*" is reset to be equal to *a* divided by (*xa+2)*, the number of elevation points considered in the for loop, plus the end points. The value of *a* then represents the sum of the elevations at each interval along the path length considered, divided by the number of intervals; i.e. the average of the elevations along the path considered, and represents the value of "a" in y = a +xb, solved for the condition where x is equal to 0.0 at the midpoint of the path considered. "a" will be the same for all points along the path.

Line 1144: a/=xa;

17. Now we need to solve for b, the slope of the straight-line formula y = a +xb. Here we are dealing with an odd number of equal-width intervals; we do not know the individual measurement errors, so we set the individual measurement error factor to be equal to 1; and we also use the "x" equidistant function (x ranges in value from = (-xa/2)+1 to (xa/2)-1 as the for loop cycles, causing "Sx"=0.0). As discussed above in (14.) and the introductory section "Background on the Linear Least Squares Fit", in solving for "b", the slope value, the formulas specified in 15.2.1 through 15.2.6 of Numerical Recipes simplifies so that "b" is equal to the sum of the "x times y" axis data point values, represented by the argument "b" value at the completion of the for loop), divided by the sum of the squares of the x values from the equidistant function along the path considered. So the formula to solve for "b", given the pre-conditions, should be: b = b (at completion of loop)/xi.

    The value of *b* at the completion of the ***for*** loop represents the "sum of elevation values multiplied by the x equidistant function value". The value of b is then reset to equal *b*, the sum of elevation values multiplied by the x equidistant function value, divided by *xi*, the sum of the squares of the x equidistant function values). The final *b* value represents the b term, or slope, in meters per interval, of the straight-line formula y = a +b*x where x = 0 at the end point of the path considered in the ***for*** loop.

    The value of *z0*, the y-axis value of the equation line formula y = a +b*x, solved at the transmitter site, is calculated to be equal to *a-(b*xb)*. The value of *a* represents the average of the elevations along the path considered. The - (b*xb) term is the slope b times the distance xb (in increments) between the transmitter site to the midpoint, where x=0, of the path considered. *z[0]* then equals the value of y = a +b*x at the transmitter site.

18. The value of *zn* the y-axis value of the equation line formula (y= a +b*x) where x is at maximum (receive site) value, is calculated to be *a+(b*(xn-xb))*. The value of *a* represents the average of the elevations along the path considered. The value of *xn* represents the number of intervals in the total path length. The value of *xb*

represents the distance, measured in intervals, from the transmitter to the mid point of the path considered by the ***for*** loop. So the term b*(*xn-xb)* term is the slope b times the distance (in increments) between the mid point of the path considered, where x = 0, to the receive site. *z[0]* then equals the value of y = a +b*x at the receive site.

19. Output lines:

The start and end y-axis (z) values of the required line:
z0 at 0, the transmitter site
zn at path.dist, the receive site

x1 and x2 are two-way, or pass-through arguments, used as inputs.

```
Line:       z0 = a-b*xb;
            Zn = a+b*(xn-xb);
}
```

Section II:

A subroutine-by-subroutine analysis:

The following chapters discuss the Longley Rice Irregular Terrain Model (ITM) implementation in the c++ source code file ITMDLL.cpp. After discussing the utilities built into the ITM, the Point-to-Point mode use of the ITM is described in a linear fashion, subroutine by subroutine, as the subroutines are first called. The Area mode, now far less utilized, is then briefly described. The last part of this section describes the changes found in the updated Version 7 of the ITMDLL.cpp, made available on the NTIA website on June 29, 2007.

The following chapters make reference to the equations and descriptions in the following prior documentation, reports, and constructs:

The Irregular Terrain Model description by George Hufford, 2002, (ITMD). The numbers of the sections of the ITMD that apply to each subroutine, are specified near the beginning of each chapter.

"Alg" numbers refer to the algorithm equations found in "The ITS Irregular Terrain Model, version 1.22, the Algorithm" by G. A. Hufford, 1995.

"ITS67" numbers refer to the algorithm equations in "ESSA Technical Report ERL 79-ITS 67, Prediction of Tropospheric Radio Transmission Over Irregular Terrain, A Computer Method – 1968" by A.G.Longley and P.L.Rice.

"TN101" numbers refer to the algorithm equations in Tech Note 101, Volume I and Volume II; "Transmission Loss Predictions for Tropospheric Communications Circuits" revised January 1, 1967, by P. L. Rice, A. G. Longley, K. A. Norton, and A. P. Barsis.

"Line" numbers refer to the ITMDLL.cpp as line numbered by Bloodshed Software's DevC++ Integrated Development Environment print function.

Reference is also often made to:

NTIA Report TR-82-100, "A Guide to the Use of the ITS Irregular Terrain Model in the Area Prediction Mode" (NTIA TR-82-100), April 1982, by G.A. Hufford, A.G. Longley, and W. A. Kissick.

"A manual for ITM, "Irregular Terrain Model", released by the NTIA as itm_man.txt, (ITM Manual).

Chapter 4: Utilities in the ITM:
Mymin, Mymax, FORTRAN_DIM, Abq_alos, and Deg2rad:

This chapter includes explanations for the following specialized short utility subroutines found in the Irregular Terrain Model:

int ***mymin***
int ***mymax***
double ***mymin***
double ***mymax***
FORTRAN_DIM
abq alos
deg2rad

int ***mymin***

Subroutine int ***mymin*** (my minimum) reports out the lowest value of two integer inputs *i* and *j*.

Call inputs:
const int &I, and const int &j

In this subroutine:

1. An **if** statement is initiated. If *i* is less than *j*, subroutine ***mymin*** returns *i*.

Line 72:
```
if (i<j)
        return i;
```

2. An else statement follows, so if *j* is less than or equal to *i*, subroutine ***mymin*** returns *j*.

Line 74:
```
else
        return j;
```

int ***mymax***

Subroutine int ***mymax*** (my maximum) reports out the highest value of two integer inputs *i* and *j*.

Call inputs:

const int &I, and const int &j

In this subroutine:

1. An **if** statement is initiated. If *i* is greater than *j*, subroutine ***mymax*** returns *i*.

Line 79:
```
if (i>j)
        return i;
```

2. An else statement follows, so if *i* is less than or equal to *j*, subroutine ***mymax*** returns *j*.

Line 81:
```
else
        return j;
```

double ***mymin***

Subroutine double ***mymin*** (my minimum) reports out the lowest value of two double precision inputs *a* and *b*.

Call inputs:

const double &a, and const double &b

In this subroutine:

1. An **if** statement is initiated. If *a* is less than *b*, subroutine ***mymin*** returns *a*.

Line 86:
```
if (a<b)
        return a;
```

2. An else statement follows, so if *a* is greater than or equal to *b*, subroutine ***mymin*** returns *b*.

Line 88:
```
else
        return b;
```

double ***mymax***

Subroutine double ***mymax*** (my maximum) reports out the highest value of two double precision inputs *a* and *b*.

Call inputs:

const double &a, and const double &b

In this subroutine:

1. An **if** statement is initiated. If *a* is greater than *b*, subroutine ***mymax*** returns *a*.

Line 86:
```
if (a>b)
        return a;
```

2. An else statement follows, so if *a* is less than or equal to *b*, subroutine ***mymax*** returns *b*.

Line 88:
```
else
        return b;
```

FORTRAN_DIM( )

Subroutine double ***FORTRAN_DIM*** performs the FORTRAN DIMension function in a c++ environment; it will report out (x – y), if x is greater than y. If x is less than or equal to y, it will report out 0.0.

Call inputs:
const double &x, and const double &y

In this subroutine:

1. An **if** statement is initiated. If *x* is greater than *y*, subroutine ***FORTRAN_DIM*** returns $x - y$.

Line 102:
```
if (x>y)
        return x-y;
```

2. An else statement follows, so if *x* is less than or equal to *y*, subroutine ***FORTRAN_DIM*** returns 0.0.

Line 104:
```
else
        return 0.0;
```

abq alos

Subroutine double ***abq_alos*** is called by subroutine ***alos.***

Call inputs:

complex<double> r

In this subroutine:

Subroutine ***abq_alos*** returns the value of: r.real()*r.real()+r.imag()*r.imag(), the sum of the square of the real component value, summed with the square of the imaginary component value, of the complex argument *r*.

Line 307: return r.real()*r.real()+r.imag()*r.imag();

deg2rad

Subroutine double ***deg2rad*** converts the value of an angle from units of degrees to units of radians (rad).

Call inputs:
double d value of an angle in units of degrees

In this subroutine:

There are 2 $*\pi$ radians in a circle of 360 degrees. Subroutine ***deg2rad*** returns the value of: d (in degrees)*2π/360, in radians/degrees. Output units are in radians (rad).

Line 307: return d*3.1415926535897/180.0;

In the Point-to-Point Mode:

Chapter 5: Point-to-Point

Longley-Rice Point-to-Point Profile

Note: This is the primary subroutine called by the commercial, freeware, or custom wrap-around software for point-to-point calculations. In version 7.0 of the ITMDLL.cpp, released in June of 2007, there are two alternative subroutines, ***point_to_pointDH*** and ***point_to_pointMDH***, that provide improvements. This subroutine initiates the point-to-point mode calculation of signal loss between two points, or terminals, over a single irregular terrain path profile. The two terminals are the transmit terminal, normally the transmitting antenna; and the receive terminal, normally the receiving antenna or location. The subroutine reports out a single value of loss, *aref*, the "reference attenuation" in decibels (dB), of the radio signal between the two terminals. Calls ***qlrps, qlrpfl,*** and then ***lrprop,*** and then calls ***avar*** to calculate additional loss due to statistical variation before reporting out *errnum*, the error code, and *dbloss*, (a.k.a. *aref,* the reference attenuation) the path loss.

Special references for this Chapter include: documentation for SPLAT!, an RF Signal Propagation, Loss and Terrain analysis tool.

From ITMD Sections: 4 and 5.

Call inputs:

The inputs to this subroutine were originally prepared and placed on IBM punch cares for input into a 1970's mainframe computer that used the 1966, and, later, 1977 ANSI standard version of the FORTRAN computer language. Today, for this c++ code, the user must provide either a commercially written version (ComStudy, PROBE, RFCAD, and TAP being examples), a freeware version (NTIA's ITMsetup.exe, Radio Mobile, SPLAT, or my own SPLAT with PLOP), or write their own wrap-around input and output processing software, to collect and process the following input data for the point_to_point subroutine. If you wish to "roll your own", as a learning tool or as a basis to build on under the GNU GPL license, the full source code for the wrap around software is only freely available for SPLAT and SPLAT with PLOP; both of these are Linux command line programs.

This data comes from a combination of direct data input from the program user, from standard preset value tables, and from a terrain elevation database [such as GLOBE, the USGS National Elevation Database (NED) or the data from the Shuttle Radar Terrain Mission (SRTM)]. At this time, only SPLAT with PLOP provides the option of using a

combination of two of the databases, one as a ground height database (NED) and one as a radio signal reflection height database (SRTM).

elev[ ] a.k.a. pfl, or array pfl, prepared for use in either version 1.2.2. or version 7.0 of the ITM compilied from source code written in c++. As this array is primarily referred to in other parts of the ITM, and in the NTIA documentation, as the pfl array, so will we. This array of elevation values is prepared by the calling program or subroutine. This array contains the values of terrain elevation heights (in meters), equally spaced along a path starting at the transmit terminal, and ending at the receive terminal, and following great circle path, with:

pfl[0] = *enp*, the number of increments between elevation data points, (also one less that the number of data points)

pfl[1] = *xi*, distance per increment, (i.e. distance between elevation height data points) in meters

pfl[2] = *z(0),* the transmitter tower base AMSL, or elevation height in meters.

pfl[[np+2] = *z(np),* the receive location AMSL, the last elevation height in meters.

Tht_m a.k.a. hg (0); transmitter antenna center of radiation height in meters, above ground level (RCAGL).

Rht_m a.k.a. hg(1); receive antenna center of reception height in meters, above ground level (RCAGL).

Note: tht_m and rht_m also referred to in the documentation collectively as HG, heights above ground.

Eps_dielect Earth's dielectric constant, a.k..a eps; relative permittivity.
Customary default setting: 15.000
Typical values:

Salt water	80
Fresh water	80
Good ground	25
Farmland, forest	15
Average ground	15
Mountain, sand	13
Marshy land	12
City	5
Poor Ground	4

Sgm_conductivity Earth's conductivity; a.k.a. sgm.
Customary default setting: 0.005 Siemens per meter
Typical values:

Salt water	5.000
Good ground	0.020

Fresh water	0.010
Marshy land	0.007
Farmland, forest	0.005
Average ground	0.005
Mountain, sand	0.002
City	0.001
Poor Ground	0.001

Eno_ns_surfref Atmospheric bending constant, (eno); eno varies with elevation above ground; here it is set to equal ens, the refractivity of the atmosphere as it approaches the surface of the earth. Customary Default setting: ens = 301.000 N-units (parts per million).

Frq_mhz Frequency of the transmitter in MHz, Range: 20 to 20000 MHz.

klim, or Radio_climate the radio climate code; set by the user or obtained from a preset list. Customary Default value is 5. The climate codes are:

1. Equatorial; (Africa, along the equator) 2. Continental Subtropical; (Sudan region) 3. Subtropical (a.k.a. Maritime Subtropical (West Coast of Africa); 4. Desert (Death Valley, NV; Sahara); 5 Continental Temperate (usual general U.S. default); 6. Maritime Temperate Over Land (California to State of Washington; West Coast of Europe including U.K.), 7. Maritime Temperate, Over Sea.

pol polarity of the transmitted signal and receive antenna:

1. Horizontal; used primarily for television broadcast and FM home reception.
2. Vertical: used for FM automotive, cellular automotive, and other 2-way vehicular and handheld tranceivers with vertical whip antennas.

conf confidence; a statistical percentage of confidence in the situation; set as .01 to .99 . In ***avar,*** the definition of confidence varies with the value of mdvar, the mode of variability; for mdvar = 0, for example, time, location, and situation variability are combined together. For point_to_point mode, mdvar = -1. See step 18 below, or the chapter on subroutine ***avar***, or itm.man. Usual default setting; 0.50 (50%) (Note: often used instead of location calculation, i.e. to approximate 50% of locations; however, ***avar*** has separate inputs for confidence and location, and point_to_point calls ***avar*** to calculate the reference attenuation with the location variable set at 0.0. See note at input loc below regarding optional subroutine ***point_to_pointMDH***.)

rel reliability; a statistical percentage of combined time and location availability; set as .01 to .99. Usual default setting; for NTSC (analog)

TV, FM broadcast and most FM analog transmissions, set to 0.50 (50% for FCC 50,50); for Digital FM IBOC sidebands, set to .90 or .98, or for television (DTV), set to 0.97 (97% for FCC 50, 97). Note: Caution required if using any setting other than .50 (50%) in point_to_point; may require reset of mdvar constant to 3 in the source code, and recompilation, to work properly. See documentation below.

NOTE: loc, immediately below, is a new option for point-to-point use; see optional alternative subroutine ***point_to_pointMDH*** released in ITMDLL.cpp version 7.0, June 2007.

loc location, a statistical percentage of location availability; set as .01 to .99. Internally fixed to zero in ***point_to_point***; as ***point_to_point*** calls subroutine ***avar*** with the location variable set at 0.0. In the new version 7, the optional alternative subroutine ***point_to_pointMDH*** allows this to be set by the user. Usual user default setting; 0.50 (50%) for 50% of locations. Disabled in subroutine ***point_to_point*** as long as mdvar remains set to 12 (point-to-point, mobile modes) in the source code. Requires reset of mdvar constant to 3 (area, broadcast modes) in the source code, and recompilation, to function. See documentation below.

Note: Users of the Terrain Analysis Program (TAP) ©, Golden Software, a commercial Longley-Rice ITM wrap-around software implementation, should check the documentation; TAP uses a modified version of ITMDLL that allows resetting of the mdvar (mode of variability) code without recompiling.

Outputs:

&dbloss dbloss, a.k.a. *aref*, the reference attenuation, or RF path loss, in dB

strmode output string for use in printed report, indicating mode of operation of the calculation by printing out “Line of Sight Mode”, “Single Horizon”, or “Double Horizon”, and either “Diffraction Dominant”, or Troposcatter Dominant”.

NOTE: New Option for point-to-point use; see optional alternative subroutine ***point_to_pointMDH*** released in ITMDLL.cpp version 7.0, June 2007, which replaces strmode with a single numerical code to eliminate printing “Line of Sight Mode”, etc. hundreds or thousands of times on the reports.

&errnum errnum, a.k.a. kwx; the error indicator. Must be preset by user input to zero at beginning of run. Indicates:
0 = no warning
1 = Warning: Some parameters are nearly out of range. Results should be used with caution.

2 = Note: Default parameters have been substituted for impossible ones. This value indicates an effect on computations. Since errnum (kwx) is cumulative, this effect on computations by substitution may or may not be true when higher numbers (3, 4, etc.) are reported out.
3 = Warning: A combination of parameters is out of range. Results are probably invalid.
4 and higher = Warning: Some parameters are out of range. Results are probably invalid.

Note: Users of the Terrain Analysis Program (TAP) ©, Golden Software, a commercial Longley-Rice ITM wrap-around software implementation, should check the documentation; TAP uses an expanded and modified set of kwx (errnum), definitions.

defines private, or local, arguments:

prop_type prop: array prop with elements:

a. *Prop.wn* wave number, = freq. in MHz/47.7 MHz*m; units in 1/meters
b. *prop.ens* surface refractivity (refractivity of the atmosphere)
c. *prop.gme* effective earth curvature
d. *prop.zgnd* surface impedance
e. *prop.zgndreal* real surface impedance (resistance component)
f. *prop.zgndimag* imaginary surface impedance (reactive component)

prop_type propa; array with elements:

zsys =0	zsys; preset to zero; later calculated to be the average elevation height along a selected portion of the total RF path (between *ja* and *jb).*
zc	conf, or confidence level to be calculated, in percent
zr	rel, or reliability level to be calculated, in percent
eno	atmospheric bending constant
enso	atmospheric bending constant to be preset to zero.
q	utility value-holding variable
ja	a location along the RF path that is 1/10 of the total RF path length rounded to the nearest increment, plus three increments, away from the transmit terminal. Specified in units of increments.
jb	a location along the RF path that is 1/10 of the total RF path length rounded to the nearest increment, plus three increments, away from the receive terminal. Specified in units of increments.
i	incremented value in a **for** loop
np	number of points, the total number of increments between the elevation height measurement points from the profile array starting with the transmit terminal site and ending with the receive terminal.
dkm	total RF path distance, in kilometers
xkm	distance per path increment, in kilometers per increment

fs free space path loss, in dB

This subroutine:

2. Sets *prop.hg[0]* , the height above ground level of the transmitting antenna center of radiation, to be equal to *tht_m*, and sets *prop.hg[1]*, the height above ground level of the receiving antenna center of radiation to be equal to *rht_m*;

Line 1418: prop.hg[0]=tht_m;
prop.hg[1]=rht_m;

3. Sets *propv.klim* to be equal to the *radio_climate* value set by the user; the customary default being 5, Continental Temperate;

Line 1420: propv.klim=radio_climate;

4. Resets prop.kwx, the error indicator (a.k.a. errnum) to be equal to zero.

Line 1421: prop.kwx=0;

5. Presets *propv.lvar* to be equal to five.

Line 1422: propv.lvar=5;

6. Sets *prop.mdp* , the mode of propagation, to be –1, which indicates operation in the point-to-point mode;

Line 1423: prop.mdp=-1;

7. Sets *zc*, the confidence level to be calculated by ***avar***, to be equal to *qerfi(conf)*, and sets *zr*, the reliability level to be calculated by ***avar***, to be equal to *querfi(rel).*

Line 1424: zc = qerfi(conf);
zr = querfi(rel);

8. Sets *np*, the number of increments (one less that the total number of elevation points in the path), equal to the value of *elev[0]* (a.k.a. pfl [0]). The SPLAT version, itm.cpp, defines the argument *np* to be **long**(*elev[0]*) to handle much more detailed (more increments per km) RF terrain paths than those originally anticipated by the ITMDLL.cpp.

Line 1426: np=(long)elev[0];

9. Sets *dkm* to be equal to the number of increments multiplied by the distance in meters between elevation points, divided by 1000 meters per kilometer. The result is the total RF path distance between the terminals, in kilometers.

Line 1427: dkm=(elev[1]*elev[0])/1000.0;

10. Sets xkm to be equal to the distance in meters between elevation points, divided by 1000 meters per kilometer. The result is the length of one increment, in kilometers.

Line 1428: xkm=elev[1]/1000.0;

11. Sets *eno*, the atmospheric bending constant (relative permittivity) equal to the value of eno_ns_surfref;

Line 1429: eno=eno_ns_surfref;

12. Presets *enso* equal to zero.

Line 1430: enso=0.0;

13. Presets *q* to be equal to *enso,* which in step 11, was preset to be equal to zero.

Line 1431: q=enso;

14. An **if** statement is initiated; if *q* is less than or equal to zero, (a given, in that *q* was set to be equal to zero in step 12), then:

a. *ja* is set to be equal to three increments, plus one-tenth of the total RF path distance in increments. The SPLAT version, itm.cpp, defines the argument *ja* to be equal to **long**(3+.1*elev[0]).
b. *jb* is set to be equal to the total number of increments, plus six, less *ja*.. This sets *jb* to be the same distance away from the receive terminal, along the rf path, that *ja* is from the transmit terminal.

Line 1433:
```
if (q<=0.0)
        {
        ja=(long)(3.0+0.1*elev[0]);  /* KD2BD added (long) */
        jb=np-ja+6;
```

15. A **for** loop is initiated, embedded within the **if** statement. The loop operates from *i = ja-1*, until *i=jb*. As the loop cycles, the value of zsys, which was preset to zero when declared, reads and sums up the value of all elevations along the RF path from *ja* to *jb*.

Line 1438: for (i=ja-1; i<jb; ++i)

zsys+=elev[i];

16. After the **for** loop has completed its cycle, then zsys is divided by the value of the total number of elevation points starting at *ja* and ending at *jb*. This results in *zsys* being equal to the average elevation height between *ja* and *jb.*

Line 1441: zsys/=(jb-ja+1);

:

17. *q* is reset to be equal to *eno,* the atmospheric bending constant.. *eno* varies with elevation above ground; here it is set to equal ens, the refractivity of the atmosphere as it approaches the surface of the earth, a user selected input with a customary preset value of 301.000 N-units. The **if** statement then ends its run.

Line 1442: q=eno;
}

18. *propv.mdvar*, the mode of variability, is set to be equal to 12. This consists of a combination of modes 10 and 2. The value "10" is added for the point-to-point mode, which causes location variability to be eliminated. [However, this should not be true in version 7, which allows location variability to be set]. The value "2" indicates "Mobile" mode, where reliability is calculated as a combination of time and location variability. Confidence in mode 2 (or 12) is given by the situation variability.

NOTE: It is interesting to note that for TV or FM broadcast use, in point_to_point mode, the mdvar is set to Mobile mode, not Broadcast. The code for mdvar, the variability mode, which sets the mode of operation, is a tens and single digit code. The single digits represent:

0 - Single message mode. Time, location and situation variability are combined together to give a confidence level.
1 – Accidental mode. Reliability is given by time availability. Confidence is a combination of location and situation variability.
2 – Mobile mode. Reliability is a combination of time and location variability. Confidence is given by the situation variability.
3 – Broadcast mode. Reliability is given by the statement of –at least- qT of the time in qL of the locations. Confidence is given by the situation variability.

The tens code is: No tens; default to area mode; combined code is 0 to 3.

10 – For the point-to-point mode. Location variability is eliminated.
20 – For interference problems. Direct situation variability is eliminated. Note that there may be a small residual situational variability.

So the setting of mdvar equal to 12, hard-coded into the ITMDLL.cpp, means that the current version of ITMDLL.cpp is never intended to be used for broadcast unless the value of mdvar is changed in the source code and the ITMDLL is re-compiled.

Note: Therefore, for broadcast reception prediction use, the ITMDLL should be modified to allow external resetting of the mdvar; this is especially critical in light of the new optional subroutine *point_to_pointMDP*, which allows for setting the percentage value for location. Golden Software, in its TAP© commercial ITM software, has already made this modification prior to the issuance of version 7 of ITMDLL.cpp.

Line 1445: propv.mdvar=12;

19. Subroutine ***qlrps*** is then called with inputs:

a. frq_mhz, the frequency in MHz
b. zsys, the average terrain height along path *ja* to *jb*
c. q, most recently set to be equal to *eno,* the atmospheric bending constant
d. pol, polarity of the transmitted and received RF signal
e. eps_dielect, earth's dielectric constant
f. sgm_conductivity, earth's conductivity
g. prop; the array *prop.*

From these inputs, subroutine ***qlrps*** calculates and processes the following data and inserts it into the prop_type structure (prop_type & prop) :

h. *Prop.wn* wave number (=freq. in MHz/47.7 MHz * meters) units in 1/meters
i. *prop.ens* surface refractivity
j. *prop.gme* effective earth curvature
k. *prop.zgnd* surface impedance
l. *prop.zgndreal* real surface impedance (resistance component)
m. *prop.zgndimag* imaginary surface impedance (reactive component)

Line 1446: qlrps(frq_mhz,zsys,q,pol,eps_dielect,sgm_conductivity,prop);

20. Subroutine ***qlrpfl*** is then called with inputs:

a. elev, array *elev (*a.k.a. array *pfl)*
b. propv.klim, the climate variable
c. propv.mdvar, the mode of variability (operating mode)
d. prop, array *prop*
e. propa, array *propa*
f. propv. array *propv*

Subroutine ***qlrpfl*** calls subroutines ***hzns, dlthx*** (which calls ***zlsq1*** and ***qtile***), after which ***qlrpfl*** may call ***zlsq1*** directly, then ends by calling ***lrprop***. Subroutine ***lrprop*** then places the value of *aref*, the value of the reference attenuation, or RF path loss, along the RF path, into array location *prop.aref*;

Line 1447: qlrpfl(elev,propv.klim,propv.mdvar,prop,propa,propv);

21. The free space path loss is then calculated by determining the amount of RF emitted from a point source, or isotropic antenna, at the center-point of a sphere, that would be received by a frequency-tuned antenna embedded in the surface of the sphere at a radius r equal to the RF path distance, all in free (open and well above the surface) space. The formula, for units of:

a. Frequency in MHz, and
b. Distance in kilometers (prop.dist/1000) is:

Free Space Loss, in dB = 32.45 + 20*$\log_{10}$(Distance) + 20.0*$\log_{10}$(Frequency)

Line 1448: fs=32.45+20.0*log10(frq_mhz)+20.0*log10(prop.dist/1000.0);

In steps 22 to 28, the subroutine utilizes string mode (strmode) to prepare printouts of the operating mode for use in a printed report.

22. Once again we press the utility variable *q* into service, setting it to be equal to prop.dist-propa.dla, subtracting the distance to the sum of the transmitter horizon and the receive horizon (or highest visible obstruction), from the total path distance. This causes q to be equal to zero at the peak of the obstruction, when both the transmitter and the receiver can see the peak of the obstruction, as under this circumstance, prop.dla is equal to prop.dist. If there are multiple obstructions, q will be positive, as prop.dla will be less than prop.dist.

Line 1449: q=prop.dist-propa.dla;

NOTE: The variable q here is used to determine where the printed report stops printing "Line-of-Sight Mode", and prints "Single Horizon" instead. It does not control the actual switch from line of sight to diffraction mode; that happens in subroutine ***lrprop***, and usually does not match the printout. (A bug for which a fix is discussed later).

23. An **if** statement is initiated; if the integer value of *q* , which was calculated in step 22, is negative (less than zero), then the path is line-of-sight, and the string prepares to output "Line-Of-Sight Mode" to an output terminal.

```
Line 1451:    if (int(q)<0.0)
                      strcpy(strmode,"Line-Of-Sight Mode");
```

24. An **else** statement follows, so if *q* is greater than or equal to zero:

```
Line 1453:    else
              {
```

25. An **if** statement is embedded within the **else** statement, so if the integer value of *q* is equal to zero, then: the string prepares to output “Single Horizon” to an output terminal.

```
Line 1455:          if (int(q)==0.0)
                    strcpy(strmode,"Single Horizon");
```

26. An **else if** statement pair is embedded within the **else** statement in step 24, so if the integer value of q is greater than zero, then the string prepares to output “Double Horizon” to an output terminal.

```
Line 1458:    else if (int(q)>0.0)
                    strcpy(strmode,"Double Horizon");
```

27. The following **if** statement is also embedded within the **else** statement in step 24, so if:
    a. *q* is not less than zero, and;
    b. *prop.dist* is less than or equal to *propa.dlsa* (i.e., the total path length, prop.dist, is shorter than or equal to: the sum of the two smooth earth horizon distances) or;
    c. prop.dist is less than or equal to propa.dx, the distance beyond which troposcatter mode has dominance, then:
    d. the string prepares to concatenate (append) the phrase “, Diffraction Dominant” to the phrase preloaded in steps 25 or 26 above.

```
Line 1461:    if (prop.dist<=propa.dlsa || prop.dist <= propa.dx)
                    strcat(strmode,", Diffraction Dominant");
```

28. An **else if** statement pair follows; they are also embedded within the **else** statement in step 24, and counteroffer the **if** statement in step 27. So if prop.dist is greater than propa.dx, the distance beyond which troposcatter mode has dominance, then the string prepares to concatenate (append) the phrase “, Troposcatter Dominant” to the phrase preloaded in steps 25 or 26 above.

```
Line 1464: else if (prop.dist>propa.dx)
                    strcat(strmode, ", Troposcatter Dominant");
    }
```

29. The subroutine ***avar*** is called with inputs:
    a. *zr*, the time reliability, (50% of the time) as a percentage decimal value between 0.01 and 0 .99, to be calculated.
    b. 0.0, the location percentage (at 00% of locations) to be calculated. (Location percentage is disabled for mdvar= 2 or 12).
    c. *zc*, the confidence, as a percentage decimal value between 0.01 and 0.99, to be calculated.
    d. array *prop*,

e. array *propv*

Subroutine ***avar*** returns *avarv*, a value representing the reference attenuation *aref* adjusted for the additional loss to be included as a result of calculating for statistical variation as a percentage of time, location, and confidence as authorized by the setting of the mdvar code. The value of dbloss (a.k.a. aref, the reference attenuation in dB) is set to be equal to the value of *avarv* in dB returned by subroutine ***avar*** and the value of the free space attenuation.

```
Line 1468:    dbloss=avar(zr,0.0,zc,prop,propv)+fs;
```

30. The value of the output-accessible argument *errnum* is set to be equal to the value of *prop.kwx*, the error indicator.

```
Line 1469     errnum=prop.kwx;
        }
```

Subroutine ***point_to_point*** ends. The user-supplied wrap-around input-output program continues. For a single terrain profile, the wrap-around input-output program may execute subroutine ***point_to_point*** hundreds of times to calculate the RF signal loss for each terrain point (except the transmitter site) calculated along the terrain path. For a modern point-to-point area mapping of signal loss, the wrap-around input-output program may run ***point_to_point*** thousands, hundreds of thousands, or even millions of times to calculate the RF signal loss for each terrain point. So it is important to keep the code "lean and tight" for purposes of speed of execution.

Chapter 5: Qlrps

Quick Longley-Rice Preparatory Subroutine; ***qlrps.***

Note: Used with both point-to-point and area prediction modes.

From ITMD Section 41:

Call inputs:

fmhz	Frequency, in MHz range 20 up to 2,000
zsys	general system elevation (calculated by ***lrprop*** subroutine to be the average elevation of the middle 80% of elev_l (a.k.a. pfl) path elevations, starting at 1/10 of the path and ending at 9/10 of the path.)
en0	Surface refractivity reduced to sea level (a.k.a. Atmospheric Bending Constant); normal default value = 301.000 N-Units
ipol	polarity (H=0, V=1) for FM vehicular reception, normal default =1
eps	polarization constant (a.k.a. Earth's Dielectric Constant, or Relative permittivity); normal default value = 15.000
sgm	ground constant (a.k.a. Earth's Conductivity); normal default value = 0.005 Siemens/meter

This subroutine:

31. Declares and establishes the constant *gma* to be equal to 157e-9. The constant *gma* represents an approximation of earth's actual curvature, as it represents the curvature of a sphere, where the earth is in fact, a spheroid. The "earth's actual curvature" is specified as: 157 N-units/km., or (157 * 10^-9)/m. The units of *gma* is 1/meter.

Line 372: double *gma* = 157e-9

32. Converts the frequency *fmhz*, to wave number *wn* .

Line 374: prop.wn=fmhz/47.7 [Alg. 1.1]

The wave number at 100 MHz, as an example, would be = 2.0964/meter.

33. Uses the surface refractivity reduced to sea level, *en0*, and the general system elevation, *zsys,* to calculate the surface refractivity, *ens,* using the formula:

ens = en0^(-zsys/z1), where z1 = 9,460 m. [Alg. 1.2]

```
Line 375:     prop.ens=en0
Line 377:     if (zsys!=0.0)    /* zsys is preset to 0 by point_to_point for first round. */
Line 380:     Prop.ens*=exp(-zsys/9460.0)          /* if zsys is not equal to 0.0, multiply
                                                       Zsys by the exponent (-zsys/9460.0).
```

If, for example, zsys =946 m., ens =eno^(-.1) = 301^(-.1) = .565 N-units.

34. Uses the constant *gma*, earth's actual curvature, and the surface refractivity *ens* (prop.ens) calculated on line 380, to calculate the effective earth curvature *gme* (prop.gme), using the empirical formula *gme=gma**(1-0.04665^(ens/en1)) [Alg. 1.3] , Where en1 is 179.3 N-units.

```
Line 380:     prop.gme=gma*(1.0-0.04665*exp(prop.ens/179.3))
              If, for example, ens = .565 N-units,
```

prop.gme =(157*10^(-9))*(1-(.04665)^(.565/179.3))= (1.509e-9)/meters.

Units for prop.gme are: 1/meters.

35. Calls ***complex,*** a c++ subroutine, to use the polarization constant, *eps,* the ground constant, *sgm*, the wave number *wn* (prop.wn), and the polarity *ipol,* to calculate the surface impedance *zgnd* (prop_zgnd).

```
Line 381:     complex<double> zq, prop_zgnd(prop.zgndreal,prop.zgndimag);
Line 382:     zq=complex<double> (eps, 376.62*sgm/prop.wn);          [Alg. 1.5]
Line 383:     prop_zgnd=sqrt(zq-1.0)

Line 385:     if (ipol!=0.0)
Line 386:     prop_zgnd=prop_zgnd/zq;              [Alg. 1.4]

Line 388:     prop.zgndreal=prop_zgnd.real()
Line 399:     prop.zgndimag=prop_zgnd.imag()
```

36. Outputs to prop_type structure (prop_type & prop) :

a. *Prop.wn* wave number
b. *prop.ens* surface refractivity
c. *prop.gme* effective earth curvature
d. *prop.zgnd* surface impedance
e. *prop.zgndreal* real surface impedance (resistance component)
f. *prop.zgndimag* imaginary surface impedance (reactive component)

Chapter 6; Qlrpfl

Quick Longley-Rice Profile

Note: Used with point-to-point mode only. Called by ***point_to_point*** after calling ***qlrps***. Calls ***hzns, dlthx*** (which calls ***zlsq1*** and ***qtile***), after which ***qlrpfl*** may call ***zlsq1*** directly, then ends by calling ***lrprop***.

Please note that the ***qlrpfl*** subroutine, and the ***dlthx, hzns*** and ***zlsq1*** subroutines that are called during ***qlrpfl,*** were intended to be experimental early versions, but are still in use today with few modifications or corrections. George Hufford, in The ITM Manual states:

> "It should be noted that the original ITM is silent on many of the details for defining some of the path parameters. This is particularly true of the effective heights HE, and, to some lesser degree, of the terrain irregularity parameter DH. The effective height, for example is defined as the height above the "effective reflecting plane," and in the past the investigator has been urged to use his own best judgement as to where that plane should be placed. The subroutine QLRPFL, in trying to automate the definition of all parameters, has been forced to define explicitly all missing details. It has done this in a way that seems reasonable and in full accord with the intent of the model. One should not, however, conclude that these efforts are completed. Hopefully, better results are obtainable."

From ITMD Section 43:

Call inputs:

pfl	terrain elevation profile array, starting at tx, ending at rcvr, following great circle path, with: pfl[0] =*enp*, the number of increments pfl[1] = *xi*, distance per increment pfl[2] = *z(0),* the transmitter tower base AMSL, or elevation height pfl[[np+2] = *z(np),* the receive location AMSL, the last elevation.
klimx	a.k.a. propv.klim, the climate code
mdvarx	a.k.a. propv.mdvar, the mode of variability; preset to 12.0 in the ***point_to_point*** subroutine, and readjusted during ***qlrpfl***;

defines private, or local, arguments:

np	number of points
j	terminal, either 0 (1, or transmit site) or 1 (2, or receive site)
x1[0]	position on terrain path a short distance from transmitter site
x1[1]	position on terrain path a short distance from receive site
q	Δh(s), delta h adjusted for the path length
za	elevation value in meters of the average terrain height at the transmit site

zb elevation value in meters of the average terrain height at the receive site
temp temporary value holding variable used at line 1326 and line 1327.

This subroutine:

37. Uses *pfl [0]*, number of points, and *pfl [1]*, increment distance, to calculate path length *prop.dist*.

Line 1302: prop.dist=pfl[0] * pfl[1];

38. Defines *np*, number of points, to be equal to the value stored in *pfl [0]*.

Line 1303: np=(int)pfl (0);

39. Calls subroutine ***hzns***, forwarding as input, arrays *pfl* and *prop.*

Line 1304: hzns(pfl,prop); See separate description for subroutine ***hzns.***

40. ***hzns*** returns:

a. *prop.the[0]* horizon elevation angle as seen from tx antenna center; specified as vertical units of increase or decrease per horizontal distance.
b. *prop.the[1]* horizon elevation angle as seen from rcvr antenna center; specified as vertical units of increase or decrease per horizontal distance.
c. *prop.dl[0]* distance from transmitter tower base to horizon
d. *prop.dl[1]* distance from receive antenna ground point to horizon

41. A **for** loop is initiated to determine the values of x1 and x2. This is a short loop, 2 cycles, from j=0 to j<2, i.e. for j=0 and j=1. j refers to the terminals, j=0 represents the transmitter site terminal, j=1 represents the receive site terminal.

Line 1306: for (j=0; j<2; j++)

42. The loop performs the following operations:

a. On the first pass, sets the value of xl[0] (a.k.a. x1, or x-one,) equal to the lesser of (15 times the height of the transmit antenna, or 1/10 of the distance from tx tower base to horizon.

Line 1307: xl[j]=mymin(15.0*prop.hg[j],0.1*prop.dl[j]

b. On the second pass, sets the value of x1[1] (a.k.a. x2) equal to the lesser of 15 times the height of the receive antenna, or 1/10 of the distance from the receive site to the horizon.

43. The value of xl[1] is then set equal to the path distance less the existing value of xl[1]. This makes it equal to the distance from the transmitter site to the point near the receiver site.

Line 1309: xl[1]=prop.dist-xl[1]

44. The value of prop.dh, the terrain irregularity parameter (a.k.a. delta h) is then determined by calling subroutine ***dlthx***(pfl,xl[0],xl[1].

Line 1310: prop.dh=dlthx(pfl,xl[0],xl[1]);

Dlthx calls ***mymin, mymax, assert, zlsql*** and ***qtile.***

dlthx returns dlthxv, the Δh (a.k.a. delta h or *dh*) terrain irregularity parameter, which is stored in prop.dh

45. Next, an **if** statement is initiated; the first task of this **if** statement is to determine if the path is a line-of-sight path, or a trans-horizon path. If the sum of prop.dl[0} and prop.dl[1], the horizon distances from the transmitter to the horizon and the receiver to the horizon, is less than 1.5 times the total path distance prop.dist, then the path is determined to be a line-of-sight path.

For example, if the transmit to receive path distance is 10,000 meters (10 km), then the combined total of the distance to the horizon from the transmit site, and of the distance to the horizon from the receive site, must equal or be greater than 150% of 10 km, or 15,000 meters (15 km), overlapping each other by an average of 1/3 of each, for the RF path to be determined to be a line-of-sight path.

As a second example, if there is a single obstacle, then the combination of the distance to the horizon (the obstacle) from the transmit site and of the distance to the horizon (the obstacle) from the receive site, would approximately equal the combination of the distance to the horizon (obstacle) from the transmit site and of the distance to the horizon (obstacle) from the receive site, and would therefore not meet or exceed the 150% of total path length overlap requirement necessary to determined to be a line-of-sight path. The default determination made would be that the RF path is a trans-horizon path, and the computer program would jump to the **else** statement on Line 1343.

Line 1312: **if** (prop.dl[0] +prop.dl[1]>1.5*prop.dist)

46. If the path is a line-of-sight path, the **if** statement then continues:

a. subroutine ***zlsql*** is called with inputs (*pfl, x1[0],xl[1*]), in order to calculate an average terrain line between points x1[0] and xl[1], and determine the average elevation height on that line at the location of x1[0] and x1[1] ;

b. ***zlsq1*** then returns:

za = z0, the elevation value of the average terrain line at the transmitter site.

zb = z1, the elevation value of the average terrain line at the receive site.

c. The effective height of the transmit site, *he(0)*, is set to be equal to *prop.hg(0) + pfl(2) – za*, but only if *pfl(2) > za*. If *pfl(2)* is not > *za,* then *he(0)* is set to be equal to *prop.hg(0).*

Therefore *prop.he(0)*, the effective height of the transmit site, is set to be equal to *prop.hg(0)*; and if the existing ground height of the transmit site, *pfl(2)*, is above the average elevation height at the transmit site, *za,* (established by ***zlsq1***), then the difference in height between the average transmit site elevation height and the ground height is also added to prop.he(0).

d. the effective height of the receive site, *prop.he(1)*, is set to be equal to *prop.hg(1) + pfl(np+2) – zb*, but only if *pfl(np+2) > zb*. If *pfl(np+2)* is not > *zb,* then *prop.he(1)* is set to be equal to *prop.hg(1).*

Therefore prop.he(1) is set to be equal to prop.hg(1), and if the existing ground height of the receive site, *pfl(np+2)*, is above the average elevation height at the receive site, *zb,* the difference in height between the average receive site elevation height and the ground height is also added to prop.he(1).

```
Line 1314:   zlsq1(zpfl,x1[0],x1[1],za,zb);
Line 1315:   prop.he[0]=prop.hg[0]+FORTRAN_DIM(pfl[2],za);
Line 1316:   prop.he[1]=prop.hg[1]+FORTRAN_DIM(pfl[np+2],zb);
```

47. A **for** loop, with two loops (j=0 and j=1), within the above **for** loop (started in Step 5.) is initiated here, to determine or re-determine the values of prop.dl[0], the distance from the transmitter site to the horizon, and prop dl[1], the distance from the receive site to the horizon, for a line of sight analysis.

NOTE: The Environmental Science Services Administration (ESSA) Technical Report ERL 79-ITS 67, "Prediction of Tropospheric Radio Transmission Loss Over Irregular Terrain, A Computer Method – 1968" by A.G. Longley and P. L. Rice, states on page 12, starting with paragraph 2:

"When individual path profiles are not available, median values of the horizon distances $d_{L1,2}$ are estimated as functions of the median effective antenna heights h_{e1} and h_{e2} determined above, the terrain irregularity factor Δh, and the smooth-earth horizon distances D_{Ls1} and D_{Ls2}. The smooth earth distance from each antenna to its horizon over a <u>smooth</u> <u>earth</u> is defined as:

$$D_{Ls1,2} = (.002 * a * h_{e1,2})^{.5} \text{ in km.} \qquad \text{ITS67 (5a)}$$

where the effective antenna heights $h_{e1,2}$ are in meters and the effective earth's radius a is in kilometers, as defined by (1). The sum of the smooth-earth horizon distance is

$$D_{Ls} = D_{Ls1} + D_{Ls2}, \text{ in km.} \qquad \text{ITS67 (5b)}$$

Median values of horizon distances over <u>irregular</u> <u>terrain</u> are estimated as

$$D_{L1,2} = D_{Ls1,2} \exp(-.07 (\Delta h/h_e)^{.5} \text{ in km,} \qquad \text{ITS67 (5c)}$$

where

$$h_e = h_{e1,2} \text{ for } h_{e1,2} \geq 5 \text{ meters, or } 5 \text{ meters otherwise.}$$

The total distance, d_L, between the antennas and their horizons is

$$d_L = d_{L1} + d_{L2}, \text{ in km".} \qquad \text{ITS67 (5d)}$$

To use these formulas in this subroutine, we convert from km to meters:

$$D_{Ls1,2} = (.002 * a * h_{e1,2})^{.5} \text{ in km.} \qquad \text{ITS67 (5a)}$$

becomes:

$$D_{Ls1,2} = (2 * h_{e1,2} * a)^{.5} \text{ in meters}$$

The effective earth's radius *a*, in kilometers, is defined by ITS67 (1). The term *gma,* representing earth's actual curvature, is a simplified approximation, as it treats the earth as a sphere, not a spheroid. It is established in subroutine ***qlrps*** to be equal to 157e–9 1/meter, and used in step 4 of ***qlrps*** to calculate the effective earth curvature, *gme*, which is then stored in array *prop* at *prop.gme.*

So what is the relationship between *a* and *gma?* One might reasonably assume that earth's actual curvature would be defined as the change per meter of circumference of the earth. If the actual earth's radius is r, then the earth's circumference is:

$$c_e = 2 * (PI) * r$$

and the actual earth's curvature might be defined, per meter, by 1divided by the circumference;

$$gma = 1/ c_e = 1/(2 * PI * r)$$

So for an actual earth radius of *r* = 6,370,000 meters, *gma* would be = *.0000000249 or 249e-10;* but it is not. T*he established value for gma is 157e9, equal to 1/6,370,000 meters.* Therefore, the actual relationship between *a* and *gma* is:

$$gma = 1/(\text{actual earth's radius, in meters})$$

The same relationship therefore applies between the effective earth's radius and the effective earth's curvature:

$$gme = 1/a, \text{ in units of 1/meters, and } a = 1/gme.$$

So then, $D_{Ls1,2} = (2 * h_{e1,2} * a)^{.5}$ m. becomes $D_{Ls1,2} = (2 * h_{e1,2} / gme)^{.5}$ m.

In converting from km to meters:

$$D_{L1,2} = D_{Ls1,2} \exp(-.07 (\Delta h/h_e)^{.5} \text{ in km,} \qquad \text{ITS67 (5c)}$$

Becomes:

$$D_{L1,2} = D_{Ls1,2} \exp(-.07 (\Delta h/h_e)^{.5} \text{ in meters,}$$

Substituting the formula for $D_{Ls1,2}$ derived above:

$$D_{L1,2} = ((2 * h_{e1,2} / gme) \exp(-.07 (\Delta h/h_e)^{.5})^{.5} \text{ in meters,}$$

where

$$h_e = h_{e1,2} \text{ for } h_{e1,2} \geq 5 \text{ meters, or } 5 \text{ meters otherwise.}$$

Which we can restate in the notation primarily used in this text as:

```
Prop.dl[j] =  ((2*prop.he[j]/prop.gme)^( –.07*(prop.dh/mymax(prop.he[j],5)^.5)
```

The formula for the total distance between the antennas and their horizons is the same as long as all units are in either km or meters:

$$d_L = d_{L1} + d_{L2}, \text{ in meters;}$$

A problem with this subroutine, with respect to both line-of-sight and trans-horizon paths, is that the Irregular Terrain Model code, ITMDLL.cpp, still uses a formula derived from ITS67 (5c) for the point_to_point mode, despite the fact that Anita Longley and Phil Rice flatly stated, at the beginning of paragraph 2, page 12 of the ESSA Technical Report ERL 79-ITS 67 quoted above, that it was to be used "only when individual path profiles are not available".

The point_to_point mode utilizes an individual path profile, input as array pfl. The use of the NTIA-released ITMDLL.cpp c++ software requires the use of additional code, a "wrap-around" package (an example being the open source program SPLAT, or its experimental, advanced concept development cousin SPLAT with PLOP) that is compiled with the irregular terrain model windows-compatible software (or in the case of SPLAT, the linux-friendly itm.cpp) that prepares the input, including deriving the pfl array from the raw elevation database data, and processes the output from the core itm.cpp subroutines,

The pfl array, especially when used in multiple runs to analyze signal loss and/or reception over a large area, usually contains far more elevation data, extending along the great circle path through the receive site, than is required to derive the distance to the actual horizon. The first two values stored in the array pfl, as sent to the point_to_point subroutine call, are pfl[0], the value of the number of intervals, and pfl[1], the value of the width, in meters, of an individual interval. The value of pfl[0] is set to indicate the number of intervals between the transmit site, and a receive site to be considered, and indicates the minimum number of elevation data values stored in the array pfl. The value of pfl[0] does not necessarily indicate the maximum number of elevation data values stored in the array pfl. The pfl array, especially when used in multiple runs to analyze signal loss and/or reception over a large area, usually contains far more elevation data, extending along the great circle path through the receive site, than is required to derive the distance to the receive site in question, as the wrap-around software will store elevation values in the pfl array extending out several tens of kilometers, in anticipation of repeating the point_to_point call to derive loss values at hundreds of receive locations along the rf path being studied. Therefore, the elevation data stored in the pfl array usually, if not always, represents elevation data along the great circle path extending far beyond any value of the horizon for a ground-mounted reception site. In the few cases where this data is not available, i.e. where the database from which the pfl array was derived does not extend to the horizon, this methodology could remain available as a default, and an additional kwx flag could be generated, to indicate that the pfl array does not extend to the radio horizon and that the distances to the radio horizon are estimated.

Therefore, with today's comprehensive elevation databases, including the SRTM and NED elevation data, there is little or no call to continue to use this approximation instead of deriving a more accurate result, where available, for the distances to the actual horizons, from the elevation database.

Therefore, this subroutine's code is eligible for review and revision in order to make today's ITM computer programs operate more in accordance with Longley and Rice's original concept, procedure, and instructions, by deriving the actual distance to the radio horizons from the transmit and receive sites, from additional terrain profile data in the pfl array.

48. The value of prop.dl[0] is estimated as a median value of a horizontal distance over irregular terrain using the formula:

Prop.dl[0] = ((2*prop.he[0]/prop.gme)^(.07*(prop.dh/mymax(prop.he[0],5)^.5)
ITS67 (5c)

Where:

prop.he[0] is the effective height of the transmitter site (from step 10)

prop.he[1] is the effective height of the receive site (from step 10)
prop.gme is the effective earth's curvature, input with the point_to_point subroutine call
prop.dh is the terrain irregularity parameter, Δh, or delta h, obtained from the ***dlthx*** subroutine call on line 1310.

Similarly, the value of prop dl[1] is estimated as:

Prop.dl[1] = ((2*prop.he[1]/prop.gme)^(–.07*(prop.dh/mymax(prop.he[1],5)^.5)
ITS67 (5c)

Line 1318: for (j=0; j<2; j++)
prop.dl[j]=
sqrt(2.0*prop.he[j]/prop.gme)*exp(–0.07*sqrt(prop.dh/mymax(prop.he[j],5.0)));

49. The variable *q* is then set to be equal to the combined total of the distance to the horizon from the transmit site, dl[0], and the distance to the horizon from the receive site dl[1], for a line of sight analysis.

Line 1321: q=prop.dl[0]+prop.dl[1];

50. An **if** loop is initiated; if q, as defined in step 12 above, is less than the total path distance from the transmit site to the receive site, then the variable *temp* is set to be equal to the total path distance, *prop.dist*, divided by *q,* and the value of *q* is then reset to be equal to the square of the value stored in *temp.*

A rare comment in the itm.cpp c++ code, referring to the earlier FORTRAN version, states: q=pow(prop.dist/q,2.0);

Line 1323:
```
if (q<=prop.dist)
{
        /* q=pow(prop.dist/q,2.0); */
        temp=prop.dist/q;
        q=temp*temp;
```

51. A **for** loop is initiated, again with two loops, j=0 and j=1, for the transmitting and receive site locations. This for loop:

a. changes the value of prop.he[0] to be equal to the existing value of prop.he[0], the effective height of the transmit antenna, multiplied by the value of q, and;

b. changes the value of prop.he[1] to be equal to the existing value of prop.he[1], the effective height of the receive antenna, multiplied by the value of q.
c. again resets the value of prop.dl[0], which represents the distance from the transmit site to the horizon, to be equal to:

dl[0]= sqrt(2.0*prop.he[0]/prop.gme)*exp(–0.07*sqrt(prop.dh/mymax(prop.he[0],5.0)));
ITS67 (5c)

Where, as in step 11 above:
prop.he[0] is the effective height of the transmitter site (from step 10)
prop.he[1] is the effective height of the receive site (from step 10)
prop.gme is the effective earth's curvature, input with the point_to_point subroutine call
prop.dh is the terrain irregularity parameter, Δh, or delta h, obtained from the ***dlthx*** subroutine call on line 1310.

d. And also again resets the value of prop.dl[1], which represents the distance from the receive site to the horizon, to be equal to:

dl[1]= sqrt(2.0*prop.he[1]/prop.gme)*exp(–0.07*sqrt(prop.dh/mymax(prop.he[1],5.0)));
ITS67 (5c)

Line 1329:
```
for (j=0; j<2; j++)
        {
   prop.he[j]*=q;
   prop.dl[j]=sqrt(2.0*prop.he[j]/prop.gme)*exp(–
0.07*sqrt(prop.dh/mymax(prop.he[j],5.0)));
        }
}
```

52. A **for** loop is initiated, again with two loops, j=0 and j=1, for the transmitting and receive site locations. This for loop determines the value of prop.the[0] and prop.the[1], the "theta" angle:

a. resets the value of *q* to be equal to 2 * prop.he[0]/prop.gme
b. sets the value of prop.the[1] equal to:
prop.the[0]=(0.65*prop.dh*(q/prop.dl[0] –1.0) –2.0*prop.he[0])/q;
c. resets the value of *q* to be equal to 2 * prop.he[1]/prop.gme
d. sets the value of prop.the[1] equal to:
prop.the[1]=(0.65*prop.dh*(q/prop.dl[1] –1.0) –2.0*prop.he[1])/q;

Line 1336:
```
        for (j=0; j<2; j++)
{
        q=sqrt(2.0*prop.he[j]/prop.gme);
        prop.the[j]=(0.65*prop.dh*(q/prop.dl[j] –1.0) –2.0*prop.he[j])/q;
}
}
```

53. If the path was determined to be line-of-sight, the program then ignores the **else** statement below, and proceeds to step 17. If the path determination defaulted to trans-horizon, the program proceeds to execute the **else** statement.

a. For a trans-horizon path, the **else** statement calls ***zlsq1*** with inputs (*pfl,xl[0]*,0.9**prop.dl[0]*);
Where *plf* is the elevation array;
xl[0] is the transmit site elevation, and
the term 0.9**prop.dl[0]* specifies a location that is at a point 9/10th of the distance from the transmit site toward the transmit site horizon.

Subroutine ***zlsq1*** calculates an average terrain line between points x1[0] and the point represented by (0.9**prop.dl[0]*), and determines the average elevation height on that line at the locations x1[0] and (0.9**prop.dl[0]*);

zlsq1 then returns:

za = z0, the elevation value of the average terrain line at the transmitter site.
q= z1, the elevation value of the average terrain line at the point 9/10th of the distance from the transmit site toward the transmit site horizon.

b. The subroutine ***zlsq1*** is called with inputs (*pfl, prop.dist*–0.9**prop.dl[1],xl[1]*);
Where: *plf* is the elevation array;
the term (*prop.dist*–0.9**prop.dl[1]*) specifies a location that is at a point 9/10th of the distance from the receive site toward the receive site horizon, and;
xl[1] is the receive site elevation

Subroutine ***zlsq1*** calculates an average terrain line between the point represented by (*prop.dist*–0.9**prop.dl[1]*), and the point represented by x1[1], and determines the average elevation height on that line at those two points;

zlsq1 then returns:

q = z0, the elevation value of the average terrain line at a point 9/10th of the distance from the receive site toward the receive site horizon.
zb = z1, the elevation value of the average terrain line at the receive site.

c. The effective height of the transmit site, *he(0)*, is set to be equal to *prop.hg(0) + pfl(2) – za*, but only if *pfl(2) > za*. If *pfl(2)* is not > *za,* then *he(0)* is set to be equal to *prop.hg(0).*

Therefore *prop.he(0)*, the effective height of the transmit site, is set to be equal to *prop.hg(0)*; and if the existing ground height of the transmit site, *pfl(2)*, is above the average elevation height at the transmit site, *za,* (established by ***zlsq1***), then the difference in height between the average transmit site elevation height and the ground height is also added to prop.he(0).

d. the effective height of the receive site, *prop.he(1)*, is set to be equal to *prop.hg(1) + pfl(np+2) – zb*, but only if *pfl(np+2) > zb*. If *pfl(np+2)* is not > *zb,* then *prop.he(1)* is set to be equal to *prop.hg(1).*

Therefore prop.he(1) is set to be equal to prop.hg(1), and if the existing ground height of the receive site, *pfl(np+2)*, is above the average elevation height at the receive site, *zb,* the difference in height between the average receive site elevation height and the ground height is also added to prop.he(1).

Note: The procedures in steps 15 and 16 do for a trans-horizon path what the procedure in step 10 does for a line-of-sight path. Therefore, the note in step 10 about the October, 2004 comments of Hammett & Edison to the FCC, regarding an error in the code, also applies to the code in steps 15 and 16.

Line 1343: else

```
{
    z1sq1(pfl,xl[0],0.9*prop.dl[0],za,q);
    z1sq1(pfl,prop.dist-0.9*prop.dl[1],xl[1],q,zb);
    prop.he[0]=prop.hg[0]+FORTRAN_DIM(pfl[2],za);
    prop.he[1]=prop.hg[1]+FORTRAN_DIM(pfl[np+2],zb);
}
```

54. The value of prop.mdp, the mode of the propagation model, is set to be equal to – 1, indicating point_to_point mode, and the value of propv.lvar, the level to which coefficients in AVAR must be redefined, is set to be equal to the greater value of either propv.lvar or 3.

Line1351: prop.mdp=-1;

```
propv.lvar=mymax(propv.lvar,3);
```

55. An **if** statement is initiated, stating that if mdvarx, a variable representing the mode of variability, is greater than zero, (zero representing the single message mode), then the value of prov.mdvar is set equal to the value of mdvarx, and the value of propv.lvar is set to the be equal to the greater value of propv.lvar, or 4.

Line 1354: if (mdvarx>=0)

```
{
        propv.mdvar=mdvarx;
```

```
        propv.lvar=mymax(propv.lvar,4);
    }
```

56. An **if** statement is initiated, stating that if klimx, the climate variable, is greater than zero, the value of propv.klim, the climate code, is set to be equal to the value of klimx, and the value of propv.lvar, which may have been set in step 18 above, is reset to be equal to 5.

```
Line 1360:    if (klimx>0)
              {
                  propv.klim=klimx;
                  propv.lvar=5;
              }
```

Note: for more information re: lvar, mdvar, and klimx, see "A manual for ITM, "Irregular Terrain Model", available at http://flattop.its.bldrdoc.gov/itm/itm_man.txt.

57. Finally, we arrive at the climax; calculating the Longley-Rice path loss. The subroutine then calls ***lrprop*** with inputs 0.0, array prop, and array propa;

```
Line 1366:    lrprop(0.0,prop,propa);
          }
```

lrprop returns the reference attenuation, *aref*. This is the "answer", the amount of loss, in db, in the rf signal level between the transmitter and the receiver.

Chapter 8: Hzns

HoriZoNS subroutine ***hzns***

Note: Used with point-to-point mode. Called by ***qlrpfl***, mid-routine.

From ITMD Sections 44 and 47:

Background; Part I: Tech Note Section 6.4, Equation 6.15:

NBS TN101 defines "launch angles" for the signal path line as it leaves the transmitter site, and as it arrives at the receive site. These launch angles, described as θ_{et}, the angular elevation of the transmit horizon ray, and θ_{er}, the angular elevation of the receive horizon ray, may be determined by field survey or from a terrain profile plot, but are usually computed using:

$$\theta_{et} = [(h_{Lt} - h_{ts}) / (d_{Lt})] - (d_{Lt}/2a) \text{ and } \theta_{er} = [(h_{Lr} - h_{rs}) / (d_{Lr})] - (d_{Lr}/2a) \quad \text{[TN101 6.15]}$$

where:

- h_{Lt}, h_{Lr} are the heights of the horizon obstacle (or obstacle peaks), above mean sea level
- h_{ts}, h_{tr} are antenna elevations above sea level, (i.e. effective height of antenna above ground level plus the ground elevation height above mean sea level)
- d_{Lt}, d_{Lr} are the distances from the terminals (the transmit site and the receive site) to the horizon (obstacle peak).
- a the effective earth's radius (utilized in the c++ code as gme, the effective earth's curvature, where gme is equal to 1/a.)

From the NBS TN101, Volume I, Section 6.4, note that it states: "As a general rule, the location of a horizon obstacle is determined from the terrain profile by using [TN101 6.15] to test all possible horizon locations. The correct horizon point is the one for which the horizon elevation angle θ_{et} or θ_{er} is maximum. When the trial values are negative, the maximum is the value nearest zero."

Background; Part II: A mathematical proof for the conversion of a measurement of the length ratio of the non-hypoteneuse sides of a right triangle, to an angle measured in radians.

How do we convert θ, a.k.a. *th*, calculated as a vertical to horizontal ratio in rectangular co-ordinates, to radians? There are 2π radians in a full cycle, or 360°. A radian is defined as the angle subtended at the center of a circle by an arc of circumference that is equal in length to the radius of the circle. Draw this construct on a circle, with one radii of length r on the horizontal plane, and a distance of r on the circumference between the

two radii. Now draw a vertical line from the point where the non-horizontal radii touches the circumference of the circle, to a point perpendicular to the horizontal radii, forming a right triangle. The radius then becomes the hypotenuse of a right triangle with an angle, subtended at the center of the circle, between the two radii, of one radian, or 57.2958 degrees. The length of the vertical line is then equal to the sine function of the angle θ, which is equal to the ratio of the length of the vertical line to the length of the hypotenuse of the triangle, which is equal to r. So we can now obtain the length of the vertical line by multiplying sin θ by the hypotenuse length, r. This results in the equation:

$$V = (\sin \theta) * r$$

The length of the horizontal line is then equal to the cosine function of the angle θ, which is equal to the ratio of the length horizontal line of the triangle to the length of the hypotenuse of the triangle, which is equal to r. So we can obtain the length of the horizontal line by multiplying cos θ by the hypotenuse length, r. This results in the equation:

$$H = (\cos \theta) * r$$

We can now obtain the ratio of the vertical length to the horizontal length by dividing the equation for Y by the equation for H. and canceling out the "r" terms:

$$V/H = [(\sin \theta) * r] / [(\cos \theta) * r] = (\sin \theta)/(\cos \theta)$$

In trigonometry, by definition, the tangent function is: $\tan \theta = (\sin \theta)/(\cos \theta)$, so the equation becomes:

$$V/H = (\tan \theta)$$

Here we will refer to the units as radians, as in c++ code, the trigonometric functions report out in radians, not degrees. This can be used to convert from the angle in radians, to the ratio used for the take-off angle in the code, but we also need to know how to convert from the vertical-distance-to- horizontal-distance ratio (V/H ratio) used for *th*, to radians. For this we use the arc tangent (a.k.a. $\tan^{-1}$) subroutine function:

$$\text{atan } (V/H) = \theta \text{ , in radians (rads)}$$

These conversion ratios will be used below.

Subroutine Call inputs:

pfl terrain elevation profile array, starting at tx, ending at rcvr, following great circle path, with:
pfl[1] =*enp*, the number of increments
pfl[2] = *xi*, distance per increment
pfl[3] = *z(0),* the transmitter tower base AMSL, or elevation height
pfl[[np+3] = *z(np),* the receive location AMSL, the last elevation.

Inputs from (and later outputs to) prop_type &prop structure, consisting of:
(All structure elements are listed; not all used by ***hzns***.)

aref the reference attenuation
dist total propagation path distance (in km?)
hg(0) transmitter site radiation center above ground level (RCAGL).
hg(1) receive site radiation center above ground level (RCAGL).
wn the wave number = freq. in MHz/47.7 MHz*m.; units in 1/meters.
dh delta H, (Δh), the terrain irregularity factor
ens refractivity of the atmosphere at sea level
gme effective earth curvature, (actual + refraction)
zgndreal resistance component of earth impedance
zgndimag reactive component of earth impedance
he(0) effective height of transmit antenna
he(1) effective height of receive antenna
dl(0) transmitter antenna horizon (or highest "visible" obstacle) distance
dl(1) receive antenna horizon (or highest "visible" obstacle) distance
the(0) take off angle, at transmit terminal, from the antenna to the transmit horizon or highest "visible" obstacle.
the(1) take off angle, at receive terminal, from the antenna to the receive horizon or highest "visible" obstacle.
kwx the error indicator value, a.k.a. errnum, or error number
mdp the mode of propagation model used. The mdp mode code is:
- 1 point to point mode
1 initializing area prediction mode
0 area prediction mode has initialized and is continuing

defines private, or local, arguments:

wq "false" if q is less than zero during for loop; (true=entire path is Line of Sight.)
np number of points in pfl elevation array
xi increment distance between points in pfl elevation array
za transmitter site radiation center height above mean sea level (RCAMSL), in meters.
zb receive site antenna center height above mean sea level in meters.
qc equal to ½ of the effective earth curvature (*gme*); units in 1/meters
q utility variable; starts as half of the effective earth curvature times the path distance, is redefined several times to represent the working variable at the moment.
sb distance, in meters, from receive site to elevation point being studied in the for loop
sa distance, in meters, from transmit site to elevation point being studied in for loop

This subroutine:

58. Defines *np*, number of points, to be equal to *pfl [0]*.

Line 1067: np=(int)pfl (0);

59. Defines *xi*, increment distance, in meters/increment, to be equal to *pfl [1]*.

Line 1068: xi=(int)pfl (1);

60. Calculates *za* to be equal to the transmitter site ground elevation height, *pfl[2],* added to *prop.hg[0]*, the transmitter site radiation center height above ground level (RCAGL). *za* becomes the transmitter site radiation center height above mean sea level. (RCAMSL). [Note: for dual database use, prop.hg must be adjusted to compensate for the height difference between the pfl array database and the ground level database, and may be a negative number.]

Line 1069: za=pfl[2] +prop.hg[0]

61. Calculates *zb*, the receive site reception center above mean sea level, using pfl [np+2], the last elevation point, and prop.hg[1], the receive site height above ground level. [Note: for dual database, prop.hg must be adjusted to compensate for the height difference between the pfl array database and the ground level database, and may be a negative number.]

Line 1070: zb= pfl[np+2] +prop.hg[1]

62. Calculates *qc* , to be equal to ½ of *gme*, the effective actual earth curvature. The constant *gme, (a.k.a. prop.gme),* equal to 157e-9 /meter, is the inverse of earth's radius.

Line 1071: qc=0.5 *prop.gme

63. Calculates *q* to be equal to *qc,* half of the effective earth curvature, times *prop.dist*, the propagation path total distance.

Line 1072: q=qc*prop.dist;

64. Presets *prop.the[1]* , the receive antenna angle of departure toward the horizon, to be equal to the height of the receive antenna above the height of the transmit antenna, divided by the path distance. If the receive antenna is below the transmit antenna, this will be a negative number. Provides the change in height between the receive antenna and transmit antenna divided by the path distance, or the ratio of difference in height to the path distance.
Units should be meters for h and qc, meters for prop.dist.

Line 1073: prop.the[1] =(zb-za)/prop.dist

65. Presets *prop.the[0]* to be equal to prop.the[1] (see 7. above) less q (see 6. above).

If the transmit antenna is below the receive antenna, this will be a negative number. Calculates *prop.the[0],* the transmitter take-off angle, to be the change in height between the transmit antenna and receive antenna divided by the path distance, less half of the path distance divided by the effective earth radius, or:

$$\text{The[0]} = \frac{\textit{difference in height between tx and receiver}}{\text{total path distance}} - \frac{0.5 * \textit{total path distance}}{\text{effective earth radius}}$$

[TN101 6.15a] Units should be in radians. But the first term is in meters of height per meter of distance.

Line 1074: prop.the[0] =prop.the[1] -q

66. Recalculates *prop.the[1]* to be equal to the negative sum of prop.the[1] (see 7. above) and q (see 6. above), or:

$$\textit{The[1]} = \frac{\textit{difference in height between receiver and tx,}}{\textit{total path distance}} - \frac{0.5 * \textit{total path distance}}{\textit{effective earth radius}}$$

[TN101 6.15b] Units should be in radians. But the first term is in meters of height per meter of distance.

Line 1075: prop.the[1] =-prop.the[1] –q

Note: Either *the[1]* or *the[0]* will be negative, indicating which site is lower, unless they are equal, indicating that the transmitter is the same height as the receiver. However, reportedly, *the[0] = the[1]* may cause a calculation problem later.

Here we have two take-off angle calculation problems, but these are not due to the specific cause claimed by Hammett and Edison in their 2004 comments to the FCC regarding the use of Longley Rice for TV reception prediction.

The calculation problems trace back all the way to Tech Note 101, Section 6.4; and relates to equation [TN101 6.15]. Of the two, the first problem appears to come from not keeping track of units, especially when mixing co-ordinate systems, and the second one to an omission. The omission is a missing equation; a separate version of [TN101 6.15] that applies when the path is no longer line-of-sight.

Problem I. The first term of the equations in 6.15 is calculated as a vertical distance change over a horizontal distance change ratio, in rectangular co-ordinates over theoretical flat earth. But TN101 specifies, in Section 6 in the paragraph preceding [TN 6.15], that <u>all</u> angles are to be in radians unless otherwise specified.

The second term, (1/2*path distance/effective earth's radius), adds the angle reduction due to the curvature of the earth and refractivity; it divides a partial circumference of a circle by a radius, and is therefore, by definition, in polar geometric co-ordinates, resulting in output units in radians.

For a smooth earth model calculation, this is an approximation that works fine, as the error at a smooth earth horizon distance, from not converting the term: (distance in height between terminals/total path distance) to radians, does not show up until the eighth decimal place. But for Irregular Terrain Model calculations with a nearby obstruction, or take off angles calculated at the base of a tall tower, skyscraper, or mountain on which is located a transmit terminal, the error becomes significant.

The calculation of the line-of-sight take-off angles can be corrected by modifiying one line:

Line 1074: prop.the[0] =prop.the[1] -q

To become:

Line 1074: prop.the[0] =atan(prop.the[1]) -q

As this correction will carry forward into the calculation of both the transmit terminal and receive terminal line-of-sight take-off angles in the next two steps.

The second place we have to apply the corrections related to the Problem I errors, occurs when recalculating the take-off angle to the peak of an obstruction.

67. Calculates both *prop.d1(0)* and *prop.d1(1)* to be equal to the propagation path distance *prop.dist*. If the path is line-of-sight, this will remain the default value.

Line 1076: prop.dl[0] = prop.dist
Line 1077: prop.dl[1] = prop.dist

68. If the number of points is greater than 2; i.e. if there are more than two points, more than just the transmitter and receiver,

Line 1079: if (np>2)

69. The value of *sa* is preset to be equal to 0.0. *sa* will represent the distance from the transmitter site to a elevation point being considered. Setting *sa* to 0.0 starts the following loop calculation from the transmitter site.

Line 1081: *sa* = 0.0.

70. A **for** loop is started that starts with i = 1 and continues until i is no longer less than the number of elevation points *np*. This causes it to consider each elevation point individually along the path from the transmitter to the receiver.

Line 1085: for (int i=1; i<np; i++)

a. *sa* starts at 0.0 intervals, and is increased each pass by an amount equal to *xi*, the interval distance between elevation points, measured in meters/interval. *sb,* which starts out equal to the path length, in meters, is decreased each pass by an amount equal to *xi. sa* then represents the distance between the transmit terminal and the elevation point being studied, with units in meters, and *sb* the distance, in meters, between the elevation point being studied and the receive terminal.

Line 1087: sa+=xi
Line 1088: sb-=xi

b. *q* is reset to be equal to *pfl[i+2]*, the elevation point height being studied on this loop, less two reduction adjustments:
   (1) the effective curvature of the earth (actual + refraction) between the transmitter and the point being studied, (qc*sa), is added to the take-off angle. In calculating the take-off angle above, this term was subtracted; so adding it here removes the effective earth's curvature adjustment. (The effective earth's curvature adjustment combines the actual earth's curvature and refractivity adjustments.) This leaves the term (qc*sa+prop.the[0]) representing the vertical to horizonal ratio of the take-off angle over theoretical flat earth from the transmit terminal to the horizon. Multiplying this term by sa, gives us the vertical distance change of the terminal to horizon take off angle (over theoretical flat earth) from the transmit terminal to the obstruction.
   (2) the transmitter elevation above mean sea level (*za)*.

This should leave q representing the vertical distance change, over a theoretical flat earth at the obstruction, between a theoretical line running from the transmit terminal antenna to the receive terminal antenna or horizon where it crosses the possible obstruction location, to the top of the elevation height at the interval location being considered in the for loop. If q is greater than zero, there is an obstruction at this location; and the vertical distance *q* represents the vertical distance change at the obstruction that must be added to the horizon take-off angle to compute the take off angle from the terminal to the peak of the obstruction.

Line 1089: q=pfl[i+2]–((qc*sa+prop.the[0])*sa) – za;

The second correction required, which is related to the first problem, occurs here, when recalculating the take-off angle to the peak of an obstruction. In line 1089, we need to recover the vertical height/horizontal height ratio that was converted to radians in our correction on line 1074. So, after making the correction discussed in Step 9, it is necessary to also adjust line 1089 to read:

Line 1089: q=pfl[i+2]–(tan(qc*sa+prop.the[0])*sa) – za;

Before proceeding, a discussion of Problem II, the omission, is necessary. Equation [TN101 6.15], even after correction for units conversion, is still only valid for a line-of-sight calculation. As a Irregular Terrain Model equation, it is a poor approximation in that it assumes, on finding one obstruction, that the obstruction is approximately in the center, and that the horizon distances are approximately equal, an assumption only generally valid for a smooth earth calculation or the special case of one, mutually visible obstruction, approximately in the middle of the path. When the path length extends beyond the mutual horizon, or more than one obstruction is found, equations [TN101 6.15] start to miscalculate the effective earth curvature adjustment. While an attempt has obviously been made to convert 6.15 from a line-of-sight equation to a beyond-the-horizon-and/or-obstruction equation by replacing total path length, D_L, with D_{Lt}, the transmit terminal horizon distance, and D_{Lr}, the receive terminal horizon distance, and two of the terminal heights with the obstruction height, the critical major change that occurs at the horizon or first obstruction was missed.

I noticed the increase in miscalculation this causes past the first obstruction, in the late 1980's, in the commercial Longley-Rice software I was using then; but only now do I know why.

It has to do with the ½ in the second term, the effective earth curvature adjustment. As long as the path is line-of-sight, $D_L = D_{Lt} = D_{Lr.}$ And $D_{Lt} + D_{Lr.} = 2 * D_{L.}$ For line of sight, the receive terminal is the end point for the transmitted signal; the reverse is true for the receive terminal, and the path length is the same distance. So approximately ½ of the effective earth curvature adjustment is applied to the transmitter end of the take-off angle calculation, and the other ½ is applied to the receive end.

But when the path length reaches to where the curvature of the earth starts to act as an obstruction, or an obstruction breaks the line of sight path, D_{Lt} and $D_{Lr.}$ suddenly shorten by approximately ½, and the second equation changes to $D_{Lt} + D_{Lr.} = D_{L.}$ When multiple obstructions occur, or the path length extends beyond a mutual horizon, the lengths D_{Lt} and D_{Lr} become independent, probably do not equal each other, and the second equation changes again to be: $D_{Lt} + D_{Lr.} < D_L$.

Therefore, [TN101 6.15] must adapt to these changes.

The simplest practical way of doing this is to say that 6.15, with the units correction shown, applies only for a line-of-sight path, and should revert to this form, re-designated 6.15a:

For when: $D_L = D_{Lt} = D_{Lr.}$ or $D_{Lt} + D_{Lr.} = 2 * D_L$ (Line-of-sight test)

$\theta_{et} = [\text{atan}((h_{rs} - h_{ts}) / (d_L))] - (d_L/2a)$ and $\theta_{er} = [\text{atan}((h_{ts} - h_{rs}) / (d_L))] - (d_L/2a)$

[TN101 6.15a, corrected and modified for line-of-sight path application only]

Here we first state equation 6.15b, as corrected and modified for use on beyond line-of-sight paths, only:

For when: $D_L \geq D_{Lt} + D_{Lr.}$ (Beyond line-of-sight test)

$\theta_{et} = [\text{atan}((h_{Lt} - h_{ts}) / (d_{Lt}))] - (d_{Lt}/a)$ and $\theta_{er} = [\text{atan}((h_{Lr} - h_{rs}) / (d_{Lr}))] - (d_{Lr}/a)$

[TN101 6.15, corrected and modified for beyond line-of-sight paths only]
where:

- h_{Lt}, h_{Lr} are the heights of the horizon obstacle (or obstacle peaks), above mean sea level in meters.
- h_{ts}, h_{tr} are antenna elevations above sea level, (i.e. effective height of antenna above ground level plus the ground elevation height above mean sea level), in meters.
- d_{Lt}, d_{Lr} are the distances from the terminals (the transmit site and the receive site) to the horizon (obstacle peak), in meters.
- a the effective earth's radius (utilized in the c++ code as gme, the effective earth's curvature, where gme is equal to 1/a.) Units: 1/m

Here we have another calculation problem, related to both Problems I and II. The prop.the[0] used at line 1093 was converted from flat earth to being calculated over the surface of a great circle, by multiplying it by an effective earth curvature term (1/2 path distance/a), or (1/2 path distance*earth*gme), that used a path distance = the smooth earth horizon distance. But the new angle stops at the obstruction. The path distance of the new angle, therefore, is equal to the value momentarily held by sa. The effective earth curvature adjustment must now be based on this different distance, *sa*, instead of the original, horizon distance, *prop.dist*.

So on line 1093, the calculation of the new angle suffers from two problems; the term q/sa is in V/H ratio, not radians; and the effective earth curvature term that has been subtracted at line 1093 was for the terminal to smooth earth horizon distance, not the new distance from the terminal to the obstruction.

There is a also a weakness in this method of computation, especially when using a detailed elevation database; the multitude of calculations that occur as the *for* loop climbs the transmit-terminal-facing face of each successive obstacle, adding multiple very tiny angles to prop.the to recompute the take off angle for each interval as successively higher points on the obstruction are considered, affect the accuracy of the final result. Because of this, while we are fixing the major problems here, we will also improve the calculation; it is far more accurate to compute the take off angle in flat-earth rectangular co-ordinates, then convert to radians and add the effective earth curvature adjustment (in radians), for each successive change in the take-off angle as the *for* loop cycles. And therefore, for correct calculation of the effective earth curvature on the second and higher for loops, we must insert, on the next line:

(new insert) Line 1090: qc = prop.gme

c. (1). If a point is reached during the for loop where *q* is greater than 0.0, indicating that either the actual horizon, and/or a first diffraction point as seen from the transmitter, has been reached, *prop.the[0]*, the transmitter take-off angle, is increased by an amount equal to *q/sa*, to direct it toward the top of the horizon point and/or diffraction point, and prop.dl[0] is made equal to sa, the path length between the tx and elevation point being studied, now the transmitter horizon/first diffraction point. The Boolean argument *wq* is set to "false", indicating that the entire path will NOT be a line-of-sight study.

Line 1091: if q>0.0
Line 1093: prop.the[0]+=q/sa;
Line 1094: prop.dl[0]=sa;
Line 1095: wq=false;

From [TN101 6.15b] we need to calculate:

$$\text{the[0]} = \text{atan}\left(\frac{(\text{obstacle hgt-transmit hgt})/\text{tx to obst. distance}}{}\right) - \frac{\text{tx to obst. dist}}{\text{Eff. earth radius}}$$

For the vertical distance difference in flat earth rectangular co-ordinates, subtract the transmit terminal height above mean sea level, from the terrain height in meters AMSL at the obstacle location.

where:

za	**transmitter site radiation center height above mean sea level (RCAMSL).**
qc	**equal to ½ of the effective earth curvature (gme), (or, later, = gme)**
q	**utility variable; here representing: the vertical distance change, over a theoretical flat earth at the obstruction, between the a line running from the transmit terminal antenna to the receive terminal antenna or**

horizon where it crosses the possible obstruction location, to the top of the elevation height at the interval location being considered in the for loop

sa distance from transmit site to elevation point being studied in for loop

pfl[i+2] elevation height at sa.

Since this calculation only occurs for non-line of sight situations, and taking our cues from: Steps 7, 8, and 9, and [TN101 6.15b] we replace:

Line 1093: prop.the[0]+=q/sa;

with

Line 1093: prop.the[0]=atan((pfl[i+2]-za)/sa)–sa*prop.gme [TN101 6.15b corrected]

We then have to apply the same correction later for the receive take off angle.

(2). If wq is not true, (i.e. if wq is "false", indicating a path length long enough to reach the horizon or a first diffraction point), q is set to be equal to the additional vertical height to be added to the take-off angle equation [TN101 6.15], to redirect the receive site take-off angle to the peak of the obstruction found.

Here we must reset the value of qc. For correct calculation of the effective earth curvature on the second and higher for **loops, we must insert, on the next line:**

Line 1090: qc = prop.gme

In line 1100, we again need to recover the vertical height/horizontal height ratio that was converted to radians in our correction in step 9. So, after making the correction at Step 8, it is necessary to also adjust line 1100 to read:

Line 1100: q=pfl[i+2]–(tan(qd*sb+prop.the[0])*sa) - za;

And so that the vertical angle will correctly be calculated on the 2nd and higher *for* **loop passes, we must add the following:**

In the declaration statements, *qd* must be added to line 1065:

Line 1065: double xi, za, zb, qc, q, sb, sa, qd;

The value of *qd* **must be defined by adding on line 1078:**

(added to blank line) Line 1078: qd = qc

Then add on line 1101, following the line: q=pfl[i+2]–(tan(qd*sb+prop.the[0])*sa) - za;

(added to blank line) Line 1101: qd = prop.gme

Line 1098: If (!wq)
{
Line 1100: q=pfl(i+2) –(qc*sb+prop.the[1])*sb-zb;

(2)(a) As the if loop sweep from sa to sb continues, at any time that q is then greater than 0.0, indicating a horizon, or diffraction point as seen from the receive site, prop.the[1], the receiver take-off angle, is increased by an amount equal to q/sb, to recalculate it to the obstruction peak, (or completely recalculated if corrected as stated above) and prop.dl[1] is made equal to sb, indicating that the receiver horizon is at the obstruction peak, and the receive horizon distance is the distance from the receive site to the last obstruction peak visible from the receive antenna.

At the end of the if loop, prop.the[1] should then indicate the take-off angle to the most distant obstruction peak "visible" (to the RF signal) from the transmitter site, and prop.dl[1] should indicate the distance to the most distant obstruction peak visible from the receive site.

Line 1102: If (q>0.0)
Line 1104: prop.the[1] +=q/sb;
Line 1105: prop.dl[1]=sb;

As we did in step 13, we must also replace the calculation on Line 1104:

where:

zb **receive site antenna center height above mean sea level (RCAMSL).**
qd **equal to ½ of the effective earth curvature (gme) (or, later, = gme)**
q **utility variable; here representing: the vertical distance change, over a theoretical flat earth at the obstruction, between a line running from the receive terminal antenna to the receive terminal antenna or horizon where it crosses the possible obstruction location, to the top of the elevation height at the interval location *sb*.**
sb **distance from receive site to elevation point being considered.**
pfl[i+2] **elevation height at sb.**

Line 1104 then becomes:

Line 1104: prop.the[0]=atan(zb -(pfl[i+2])/sb)-sb*prop.gme
[TN101 6.15 corrected]

71. return or endpoint for ***for*** loop.

72. ***hzns*** returns:

Outputs to prop_type structure:

a. *prop.the[0]* horizon take-off angle from transmitter
b. *prop.the[1]* horizon take-off angle from receiver
c. *prop.dl[0]* distance from transmitter to horizon (or 1st obst.)
d. *prop.dl[1]* distance from receiver to horizon (or last obstacle).

Finally, a note regarding a famous protest:

Hammett and Edison, in their October of 2004 comments submitted to the Federal Communications Commission (FCC) in CS Docket 98-201, regarding the use of Longley-Rice in calculating Grade B TV Signal Coverage, stated in paragraph 20:

"This ongoing work has convinced us that the implementation of the L-R model is even more flawed than had been originally suspected. For example is has come to light that the OET-69 software calculates the depression angle to a calculation point using the sources height above ground, not its height above sea level. This coding mistake by itself will introduce errors of perhaps 10-20 dB in the calculation results."

The code studied here does not, in and of itself, contain this error. In step 3 above, *za*, the transmitter height used for the angle calculations, is the transmit antenna height above sea level, determined by adding the transmitter site ground elevation height, *pfl[2],* to *prop.hg[0]*, the transmitter site radiation center height above ground level (RCAGL). The same is true of *zb*, the receive antenna height above sea level. So if the original claimed source of the error spoken of by Hammett and Edison still exists, it exists in the data preparatory subroutines written for OET-69 to prepare the information for the ITM subroutines, not in the ITM 1.2.2 code, including the new version 7, publicly released by the NTIA since 2003.

Chapter 9: Dlthx

Delta h (experimental) subroutine

Note: Used with point-to-point mode. Called by ***qlrpfl***, mid-routine.
Calls ***mymin, mymax, assert, zlsql, qtile***.

Used to find dh, (a.k.a. delta h) the terrain irregularity factor.

Computes the terrain irregularity parameter *dh* from the profile elevation array *pfl* between points located at x1 and x2. x1 is defined as a distance from the transmitter site to the start point of a path of elevation points considered; x2 is defined as a distance from the transmitter site to the end point. Both x1 and x2 must be specified in meters.

Please note that the *qlrpfl* subroutine, and the *dlthx, hzns* and *zlsq1* subroutines that are called during *qlrpfl,* were intended to be experimental early versions of L-R software. They are still in use today, with few modifications or corrections. The ITM Manual states:

> **"It should be noted that the original ITM is silent on many of the details for defining some of the path parameters. This is particularly true of the effective heights HE, and, to some lesser degree, of the terrain irregularity parameter DH. The effective height, for example is defined as the height above the "effective reflecting plane," and in the past the investigator has been urged to use his own best judgement as to where that plane should be placed. The subroutine QLRPFL, in trying to automate the definition of all parameters, has been forced to define explicitly all missing details. It has done this in a way that seems reasonable and in full accord with the intent of the model. One should not, however, conclude that these efforts are completed. Hopefully, better results are obtainable."**

Background on Delta H, (Δh), the Terrain Irregularity Parameter

Also described as "Δh, the interdecile range of elevations between the two points x1 and x2". The **interdecile range** is a specific interquartile range; it is computed as the difference between the 10th and 90th percentiles. It comprises the middle 80% sample of the population: in this case, a set of elevation values derived from the elevation data contained in the *pfl* array. The term x1 is defined as a distance from the transmitter site to the start point; x2 is defined as a distance from the transmitter site to the end point. We find by studying the Δh, or delta h, used in the irregular terrain model, that the concept and calculation does not closely follow the definition of the terrain roughness factor, σ_h, specified in Tech Note 101.

In Tech Note 101, on page 5-9, it states:

"5.2.2. **The terrain roughness factor, σ_h.** The τερραιν roughness factor σ_h in (5.1) is the root-mean-square deviation of modified terrain elevations, y_i, relative to the smooth curve defined by (5.16), within the limits of the first Fresnel zone in the horizontal reflecting plane. The outline of a first Fresnel zone ellipse is determined by the condition that:

$$r_{11} + r_{21} = r_1 + r_2 + \lambda/2$$

Where:

$r_{11} + r_{21}$ is the length of a ray path corresponding to reflection from a point on the edge of the Fresnel zone,

$r_1 + r_2$ is the length of the reflected ray for which angles of incidence and reflection are equal.

Norton and Omberg [1947] give general formulas for determining a first Fresnel zone ellipse in the reflecting plane. Formulas are given in annex III for calculating distances x_a and x_b from the transmitter to the two points where the first Fresnel ellipse cuts the great circle plane."

Later, on page 5-13, it states: "the terrain roughness factor σ_h, (in Tech Note 101 here designated by a lowercase sigma sub h), is the root-mean-square deviation of modified terrain elevations relative to the curve y(x) within the limits of the first Fresnel zone in the horizontal reflecting plane. The first Fresnel ellipse cuts the great circle plane at two points, x_a and x_b kilometers from the transmitter. The distances x_a and x_b may be computed using equations (III.18) or (III.19) to (III.21) of annex III."

The root-mean-square is the square root of the average of the squares of a set of numbers. If we have a set of values in an array: $x_1, x_2, x_3, \ldots, x_n$, the root-mean-square can be computed, using a for loop, as the square root of the sum of the squared array values divided by n, or:

$$x_{rms} = ((x_1^2 + x_2^2 + x_3^2 \ldots + x_n^2)/n)^{1/2}$$

Deviation refers to the amount of difference between the value being considered and the arithmetic mean value. One of the most important uses of the root-mean-square is to determine the *standard deviation* from the arithmetic mean. The standard deviation from the mean is the root-mean-square of the deviations from the mean.

In classical statistics, the formula for calculating the variance of an unknown population variance is:

$$\sigma^2 = \frac{\Sigma(x - \mu)^2}{N}$$

Here the population parameter is abbreviated with the Greek letter sigma in lower case, the mean is a population parameter (mu), and the number of samples is represented with a

capital N. The term $[x - \mu]$ represents the difference between the sample value (x) and the mean value μ; this term is the deviation of the sample.

The standard deviation, σ, is simply the square root of the variance, or:

$$\sigma = ((\Sigma(x - \mu)^2)/N)^{1/2}$$

Here we are using a sample of the total population. For calculating statistics of a sample of the population, statisticians indicate that the mean is of a sample population by replacing μ with the arithmetic mean $\bar{x}$, and they decrease the denominator by 1, to account for the percent probability that a wider deviation exists in the population than in the sample of the population. The formula then becomes:

$$\sigma = ((\Sigma(x - \bar{x})^2)/(n))^{1/2}$$

Where $n = (N - 1)$

If the arithmetic mean $\bar{x}$ is equal to:

$$\bar{x} = (x_1 + x_2 + x_3 \ldots + x_n)/n,$$

then the standard <u>deviation</u> s can be computed, using a for loop, as:

$$s = (((x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + (x_3 - \bar{x})^2 \ldots + (x_n - \bar{x})^2)/n)^{1/2}.$$

The terrain roughness factor is the root-mean-square deviation of modified terrain elevations relative to the curve y(x) within the limits of the first Fresnel zone in the horizontal reflecting plane. We can now see how to obtain the root-mean-square deviation of terrain height data taken from a selected part of the path between the transmitter and the receiver; the information is in the pfl array. But what is meant by "modified terrain elevations relative to the curve y(x)"?

In 5.2.1, a Curve-Fit to Terrain, found on page 5-8 of Tech Note 101, it states:

> "A smooth curve is fitted to terrain visible from both antennas. It is used to define antenna heights h'_1 and h'_2, as well as to determine a single reflection point where the angle of incidence of a ray r_1 is equal to the angle of a reflection of a ray r_2 in figure 5.1. This curve is also required to obtain the deviation of terrain heights used in computing R_e in (5.1).
>
> First, a straight line is fitted by least squares to equidistant heights $h_i(x_i)$ above sea level, and $(x_i)^2/(2a)$ is then subtracted to allow for the sea level curvature 1/a illustrated in figure 6.4. The following equation describes a straight-line h(x) fitted to 21 equidistant values of $h_i(x_i)$ for terrain between $x_i = x_0$ and $x_i = x_{20}$ kilometers from the transmitting antenna. The points x_0 and x_{20} are chosen to exclude terrain adjacent to either antenna which is not visible from the other:"

$$h(x) = \bar{h} + m(x - \bar{x}) \qquad (5.15a)$$

Here, $\bar{h}$ is the arithmetic mean of the elevation heights, i.e. the average terrain height:

$$\bar{h} = ((h_1 + h_2 + h_3 \ldots + h_{20})/21 \qquad (5.15b, 1 \text{ of } 3)$$

$\bar{x}$ is the arithmetic mean of the distances from the transmitter site to the to $x_0 + x_{20}$, i.e. the distance from the transmitter site to the center of the x path:

$$\bar{x} = (x_0 + x_{20})/2 \qquad (5.15b, 2 \text{ of } 3)$$

m is the slope of the line. To simplify the later calculation of the least squares solution, the slope is calculated with reference to an x co-ordinate referenced to the center of the x path. If *n* is the number of elevation points, in this example 21, then (*n*-1) is equal to the number of intervals between the elevations points, equal in this example to 21 – 1 = 20. The center of this path of equidistant intervals is then at i = (*n* – 1)/2, or (21 – 1)/2 = 10. So if we want to start a for loop calculation from the transmitter end of the path, we have to start at the recalibrated x path position, the interval closest to the transmitter site, in this example now equal to (i – 10). So the slope calculation will start at i = 0 and proceed to i = 20, represented in the below calculation by i – 10, therefore calculating from –10 to +10.

$$m = \frac{2 * (h_1(i-10) + h_2(i-10) + h_3(i-10) \ldots + h_{20}(i-10))}{77 * (x_{20} - x_0)} \qquad (5.15b, 3 \text{ of } 3)$$

This is a slightly different notation, but is still the same formula given in 5.15b, 3rd of 3. <u>The derivation of the 2/(77*(77 * (x_{20} - x_0))) set of terms is not explained in Tech Note 101; *and provides a correct answer only if the number of increments equals 21.*</u>

The derivation of a more universal version of this formula, which will work with any number of increments, is shown in the description of the subroutine ***zlsq1,*** which this subroutine, ***dlthx*** calls, in an attempt to perform this least-squares-fit-to-a-line calculation. Here, we will simply state that the version of the formula for the slope, m, which provides correct results for any number of increments, is:

$$m = \frac{(h_1(i-10) + h_2(i-10) + h_3(i-10) \ldots + h_{20}(i-10))}{((i_0-10)^2 + (i_1 10)^2 + (i_2-10)^2 \ldots + (i_{20}-10)^2)}$$

And is intended to be incorporated in a revised version of ***zlsq1***, to be named ***zlsq2.***

After 5.15b, Tech Note 101 states:

"Smooth modified terrain values given by

$$y(x) = h(x) - x^2/(2a) \qquad (5.16)$$

will then define a curve of radius a which is extrapolated to include all values of x from x = 0 to x =d, the positions of the antennas."

Here, a represents the effective radius of the earth under the great circle signal path. The $- x^2/(2a)$ term accounts for the effective curvature of the earth; it adds the effective increase in terrain height due to the effective curvature of the earth, to the "flat earth" signal path average terrain line formula. Section 4 of Tech Note 101 describes a method of deriving the effective earth's radius, a, from n_s, the atmospheric refractive index at the surface of the earth, and a_o, the actual radius of the earth.

Note before reading further: The Δh correction term defined by (6.12) in the next quote, mentioned after (5.17), is not the same Δh we have previously discussed in this section. It is only distantly related to, and not the same as, the delta h (Δh) function calculated by, and result *dlthxv* provided by, subroutine ***dlthx***.

Continuing with Tech Note 101 after (5.16). it states:

"The heights of the antennas above this curve are:

$$h'_1 = h_{ts} - h(0), \qquad h'_2 = h_{rs} - h(d) \qquad (5.17)$$

If h'_1 or h'_2 is greater than one kilometer, a correction term, Δh, defined by (6.12) and shown on figure 6.7, is used to reduce the value given by (5.17).

Where terrain is so irregular that it cannot be reasonably well approximated by a single curve, σ_h is large and $R_e = 0$, not because the terrain is very rough, but because it is irregular. In such a situation, method 3 of section 5.1 may be useful."

Now, on to the description of the subroutine. From ITMD Section 44:

Call inputs:

pfl — terrain elevation profile array, starting at transmitter site and ending at receiver site, following great circle path, with:
pfl[0] =*enp*, the number of increments
pfl[1] = *xi*, distance per increment (in meters)
pfl[2] = *z(0),* the transmitter tower base AMSL, or elevation height
pfl[[np+2] = *z(np),* the receive location AMSL, the last elevation.

&x1	x1, input from qlrpfl as xl[0], the start point of the section of the total path to be considered, defined as a distance (measured in meters) from the transmitter site; also an output.
&x2	x2, output from qlrpfl as xl[1], the end point of the section of the total path to be considered, defined as a distance (in meters) from the transmitter site; also an output

This subroutine declares the following private, or local, arguments:

Note: in this case, I am noting the argument types (int, and double, in this case) because this subroutine incorporates conversions between (int) and (double).

Declared as type integers (int):

np	number of points in pfl elevation array.
ka	equal to 1/10 of (8+length of the section of the total path to be considered), range limited to between 4 and 25.
kb	equal to n- (ka+1).
n	number of intervals between the transmitter site and the end point of a path to be considered in this subroutine only.
k	initially set to be one more than xa.
j	counting variable in a ***for*** loop.

Declared as type double decimal precision (double):

d1thxv	calculated terrain irregularity parameter; output from dlthx.
sn	one less than n. The total path length, measured in increments, of path n.
xa	working variable related to beginning of test path; the value and definition of this variable changes as noted in the text, often completely redefined from use to use in various for and if loops.
xb	working variable related to end of test path; the value and definition of this variable changes as noted in the text, often completely redefined from use to use in various for and if loops.
*s	an array of elevation variables, partially derived from pfl.

This subroutine:

73. Defines *np*, number of points, to be equal to *pfl [0],* the number of increments in array pfl.

Line 1249: np=(int)pfl [0];

74. Defines *xa* to be equal to xl/*pfl [1], equal to x1(one, the transmitter site) divided by the increment distance,* initially making double *xa* equal to the number of increments from the transmitter site to the start point of the section of the total path to be considered.

Line 1250: xa=x1/pfl[1];

75. Defines *xb* to be equal to x2/*pfl [1], equal to x2(receiver site) divided by the increment distance,* initially making double *xb* equal to the number of increments from the transmitter site to the end point of the section of the total path to be considered.

Line 1251: xb=x2/pfl[1];

76. Presets dlthxv=0.0

Line 1252: dlthxv=0.0.

77. if statement used as check; if (xb - xa<2.0), is true, indicating a path length too short to calculate dlthxv, it causes the program to exit, returning dlthxv = 0.0 .

```
Line 1254:    if  (xb-xa<2.0)
Line 1255:            return d1thxv;
```

78. Sets *ka* to be equal to 1/10 of (xb-xa+8), or 1/10 of (8+length of the section of the total path to be considered, measured in increments), then sets the range of *ka* to be between 4 and 25 increments.

Since *ka* is declared as an int, the data used to calculate its value must be int; but the declared local xb and xa are of type double. The function, by stating (int) after the equal sign and before the calculation, forces (creates) an output that is (int).

```
Line 1257:    ka=(int)(0.1*(xb-xa+8.0));
Line 1258:    ka=mymin(mymax(4,ka),25);
```

79. Sets value of *n* equal to (10 times ka) – 5; since ka is limited in range to between 4 and 25, then n is limited in range to between 35 and 245 increments.

NOTE: THIS IS AN ARCHAIC, PROBLEM-CAUSING RANGE LIMIT FOR n, FAR TOO SMALL FOR TODAY'S 90 METER (3 ARC-SECOND), 30 METER (1 ARC-SECOND) AND 10 METER (1/3 ARC-SECOND) PATHS, WITH THOUSANDS OF INTERVALS. It limits n to 245 times 90 m, or 22 km, for 3 arc-second database interval sizes, which should leave it marginally functional, but less than optimum, but it limits the maximum consideration to only 7.35 km for 1 arc-second database all-points intervals, and 2.45 km for 1/3 arc second database all-points intervals. The range limit must be able to handle long horizons, such as those from a tall tower or mountaintop transmitter site that can transmit over relatively flat land; an example being paths from transmitter sites atop Cheyenne Mountain that can transmit toward Denver.

The FCC regulations for their version of [delta] h specifies a minimum 10 km, and up to a 50 km maximum, limit; the n limit of 7.35 km for 1 arc-second database all-points intervals, and 2.45 km for 1/3 arc second database intervals, are both below the minimum FCC limit for consideration.

In continuing the comparison to the FCC's terrain consideration rules, we find that FCC 47CFR73.7313 indicates that the consideration of terrain roughness extends from n = 10 to 50 km; i.e. the maximum length of the path to be considered was 50 km, the calculation was to be made using terrain information on the signal path starting at 10 km and extending to the receive point, up to a maximum of 50 km. No consideration was to be made for terrain roughness if the path was less than 10 km. The length of n, being 10 times ka less 5, can only compare well with the 10 to 50 meter maximum range of the FCC's delta h determination method, unless ka = 1, where n would = 5; since ka is range-limited to be not less than 4, this is not possible. So let us look at another part of the FCC regulations; where a minimum number of 50 increments over the 40 km consideration path section are required. Converting from paper maps to a digital database, this is 1.2 increments per km, equivalent to an 830-meter database, or roughly equivalent to, and therefore compatible with, the 900 meter GLOBE 30 arc-second database utilized by the Windows version of the ITM (itm.exe), made available by NTIA.

The maximum and minimum limits for ka, therefore, need to be set based upon the path lengths, not the number of intervals. We only need to reset the maximum limit, as ka is calculated to be far less than xb. The maximum length of n, to be limited to 50 km, or 50,000 meters, is dependent on ka, so we want to set ka to be a maximum of 5,050 meters. Distance in meters can be converted to intervals by dividing by pfl[1], the increment width, so:

On Line 1245; add kmx, between k, and j on the int declaration line.

Assuming the units of pfl[1] is meters/increment, in dlthx2, line 1258 should be replaced by:

Line 1258r: ***kmx=(int)(5000.5/pfl[1])***
Line 1259r: ***ka=mymin(mymax(4,ka),kmx);***

Line 1258r sets the kmx, or ka maximum, to be the integer value of 5500.5 divided by the increment width, leaving kmx measured in increments. Line 1259r replaces 25 with kmx.

END CORRECTION NOTE. Returning to the discussion of dlthx;

Line 1259: n=10*ka-5

80. Sets value of *kb* equal to *n-ka*+1

Line 1260: kb= n–ka+1

81. Sets value of *sn* equal to *n*-1, the total path length, measured in increments, of path n.

Line 1261: sn=n-1,

82. Calls subroutine ***assert*** with input parameters (s=new double[n+2]) !=0). ***assert*** is a standard c function prototype that, in c++, returns an error message and aborts the program if the expression, in this case s=new double[n+2], is equal to zero, i.e. the expression is false. Note the use of new to create and allocate memory for the s array, and of double to declare the s array values to be doubles, where n was declared as an integer (int). So if *n* +2 = 0, the program aborts, returning an error message.

Line 1262: assert (s=new double[n+2]) !=0)

83. Sets value of array value s[0] equal to sn, which is equal to the length, in increments, of the path considered, also = (n – 1).

Line 1263: s[0]= sn

84. Sets value of array value s[1] equal to 1.0

Line 1264: s[1]=1.0

85. Sets value of *xb* to be equal to *(*xb-*xa)/sn*, where:

(xb, prior to step 13) = the number of increments from the transmitter site to the end point of the path considered.
xa = the number of increments from the transmitter to the start point of the path considered.

sn is the length of the path considered, measured in increments.

So (xb-*xa*) is the length, in increments, of the path considered, and *xb* is redefined to be the length of the path considered divided by the length of the path defined by n, i.e. sn = n – 1, measured in intervals. This makes *xb* equal to the ratio of the length of the "x" path to the length of the "n" path.

Line 1265: xb=(xb-xa)sn

86. Sets value of *k* equal to xa+1.0, equal to the number of increments from the transmitter to the start point, plus one. The use of (int) after the equal sign allows and forces a calculation using (double) xa, to produce an integer result. The value of xa is truncated, not rounded off, to zero decimal places, so 3.9 and 3.3 both would calculate as 3, i.e. if xa is 3.9, *k* = 3 +1 = 4, if xa is 3.3, *k* = 3 +1 = 4.

Line 1266: k=(int) (xa+1.0);

87. Resets value of double xa by subtracting value of k. The use of the (double) before k allows and forces a type double result to a calculation incorporating the integer (int) k. Since k = (xa +1), with the value truncated to zero decimal places by the double to integer conversion, this calculation will produce a double xa that is equal to the value of xa less the value of (xa +1) truncated to no decimal places, resulting in xa = a negative value, between -1 and slightly negative of zero. The new xa value after the calculation on line 1267 is equal to the negative of the amount to the right of the decimal point of xa prior to the calculation on line 1267.

Line 1267: xa-=(double) k;

88. Initiates **for** loop, starting with j=0, continuing until j=n;

Line 1269: for (j=0, j<n; j++)

a. A **while** loop is initiated within the **for** loop, running while *xa* > 0.0 and k<np. When *xa* is greater than zero, it:
   (1) Subtracts 1 from *xa*.
   (2) Increments the value of *k*, increasing the value of *k* by 1.

```
Line 1271:  while (xa>0.0 && k<np)
Line 1273:          xa-=1.0;
Line 1274:          ++k
```

The **for** loop then continues;

b. It sets value of s[j+2) equal to pfl[k+2]+(pfl[k+2]-pfl[k+1])*xa,
c. And then redefines the value of xa, this time equal to xa + xb.

Line 1277: s[j+2]= pfl[k+2]+(pfl[k+2]-pfl[k+1])*xa
Line 1278: xa=xa+xb

An interesting sequence of events happens in this **for** loop. The loop populates the s array elevation values in s array positions s[2] to s[n +1], i.e. s[(n –1) + 2] with values derived from the values in the pfl loop. Derived is the key word here; the values in pfl from pfl array locations pfl [k+2], with k incrementing approximately every 2nd s array increment, and proceeding toward location pfl [np –1], are being interpolated to fill a set of intervals equal to (n –1), the number of intervals in path "n", with interval widths equal to the intervals in path "n".

89. Subroutine ***zlsq1*** is then called, with inputs (s , 0.0, sn) where:

s is an array with values:
s[0] = sn (see 11. above)
s[1] = 1.0 (see 12. above)
s[2….(n-1)] = elevations calculated in for loop (see 16. above)

0.0 indicating that ***zlsq1*** is to start at the s array elevation value stored in s[2].
sn which is equal to the length, in increments, of the path considered, also = (n – 1), indicating that ***zlsq1*** is to continue considering the s array elevation values all the way to the value stored in s [n + 1], i.e. s ((n – 1)+2).

Output values:

xa now redefined as the z0, or value of average terrain height line at transmitter site.

xb working variable; after xlsq1 called, is the z1, or value of average terrain height line at receiver site.

90. ***zlsq1*** then returns:

xa = z0, the elevation value of the average terrain line at the transmitter site.
xb = z1, the elevation value of the average terrain line at the receive site.

91. The value of xb is then again redefined to be equal to (xb-xa)/sn, or (the elevation value of the average terrain height at the receive site above the elevation value of the average terrain height at the transmitter site), divided by (n – 1).

Line 1282: xb=(xb-xa)/sn

92. A **for** loop is initiated, starting at j=0; running until j is no longer less than n.

Line 1284: for (j=0; j<n; j++)

a. The loop first subtracts the value of xa from s[j+2], subtracting the average elevation height calculated by ***zlsq1*** from the first elevation height in the s array, leaving as a value, only the amount of deviation, in meters, from the average height, stored in the each of the s array locations.

In Tech Note 101, at 5.16, the term $[-x^2/(2a)]$, adding consideration of the effective curvature of the earth, is added to the straight line formula y(x) = h(x), and to the elevation heights, before the plotting of the terrain and calculation of the deviations. However, since here we are not plotting, and the deviations are being obtained by subtracting the straight-line formula results, (the value of xa as incremented by the ***for*** loop), from the terrain heights (found in the s array values prior to processing by this ***for*** loop), the effective curvature term would cancel out. Therefore, the values of the elevation deviations obtained and stored in the s array will be the same with or without consideration of the effective earth radius.

Line 1286: s[j+2]-=xa

b. It then adds the value of (double) xb to (double) xa. As the loop advances, this causes the value of xa to proceed from the z0 value to the z1 value, following along the average elevation height line y= a + bx calculated by ***zlsq1.***

Here again this subroutine branches off from following Tech Note 101, specifically, the description of methodology and procedure specified for determining the terrain roughness factor, σ_h, described in Tech Note 101 at 5.2.2. In its place, the subroutine determines a related value, described as a delta h [Δh]. First, note that this Δh is NOT the same as the Δh correction term defined by Tech Note 101 at (6.12), a term that is to be applied only when the effective heights of the transmitter and/or receiver are greater than a kilometer above sea level. It is related to, but not the same as, the delta h (Δh) function calculated by, and result *dlthxv* provided by, subroutine ***dlthx***.

The subroutine ***dlthx*** is described in Appendix A of NTIA Report TR-82-100, "A Guide to the Use of the ITS Irregular Terrain Model in the Area Prediction Mode", as:

> "Computes the terrain irregularity parameter *dh* from the profile *pfl* between points at x1 [less than] x2.

And in the ITMD, ***dlthx*** is described as:

Using the terrain profile *pfl* we find *Δh*, the interdecile range of elevations between the two points *x1* and *x2*".

The **interdecile range** is a specific interquartile range; it is computed as the difference between the 10th and 90th percentiles. It comprises the middle 80% sample of the population. The term x1 is defined as a distance from the transmitter site to the start point; x2 is defined as a distance from the transmitter site to the end point.

Why is this quantile-based system used to derive *Δh*? Quantiles are less susceptible to long tailed distributions and outliers. Since the elevation data may have anomalies, such as voids in SRTM data, causing outliers far removed from the mean, as long as the outliers are less than 10% of the total data, causing them to be lost in the lower 10% and/or upper 10% of data that is abandoned in the ***qtile*** subroutine, this methodology provides more accurate results than means and other moment-related statistics.

Further explanation of quantiles can be found in the chapter on the ***qtile*** subroutine, which ***dlthx*** calls below in order to determine the 10th and 90th percentile quantiles.

Line 1287: xa=xa+xb

93. ***qtile*** is a subroutine that returns a quantile. The subroutine ***qtile*** is called twice, with the same path length (n-1), using array s and starting at s array location s[2]; once for quantile (ka-1), supposedly the 90th percentile quantile, and the second time for quantile (kb-1), the 10th percentile quantile. The s+2 term causes the subroutine to skip the s array locations s[0] and s[1] which store the increment length and quantity values, and start with the elevation values in s[2].

94. The value of dlthxv is set to be equal to the difference between the two quantile values obtained from the deviation-from-average-terrain values in the s array, equal to quantile (ka-1)-quantile (kb-1).

Line 1290: dlthxv= qtile(n-1,s+2,ka-1)-qtile(n-1,s+2,kb-1)

With regard to describing the methodology and procedure to this point, for determining this *Δh*, the interdecile range of elevations between the two points *x1* and *x2*", the NTIA is not specific. George Hufford stated in the Algorithm, Section 1.3:

> "These quantities, together with *Δh*, are all geometric and should be determined from the terrain profile that lies between the two terminals. We shall not go into detail here."

95. The Algorithm does provide information as to the source of the formula for the next step, where it states in section 3.2., Preparatory calculations for both modes:

    "We also note here the definitions of two functions of a distance s:

    $$\Delta h(s) = (1 - 0.8\, e^{-s/D})\Delta h \quad \text{with } D = 50 \text{ km.} \qquad \text{Alg. (3.9)}$$

and

$\sigma_h(s) = 0.78\ \Delta h(s) \exp\left[-\left(\Delta h(s)/H\right)^{1/4}\right]$ with $H = 16$ meters." Alg. (3.10)

the second formula, Alg. (3.10), shows a relationship between Δh and the terrain roughness factor σ_h used in Tech Note 101.

The formula Alg. (3.9) can be manipulated for use here; replacing s in Alg. (3.9) with the distance (x2 –x1), the distance between the end point and the start point of the path of elevations considered, specified in meters;

$\Delta h(s) = (1 - 0.8\ e^{-s/D})\Delta h$ with D = 50 km, (50,000 meters) becomes:

$$\frac{\Delta h(s)}{(1 - 0.8\ e^{-s/D})} = \Delta h \quad \text{where } D = 50{,}000 \text{ meters, } s = (x2 - x1)$$

The term $(1 - 0.8\ e^{-s/D})$ ranges in value from .2 for s = 0, up to .706 for s = 50 km.

The value obtained at Line 1290 for *dlthxv*, a.k.a. *Δh(s)*, is then divided by 1.0-.8*exp(– (x2-x1)/50,000), a path distance adjustment factor, to obtain Δh;

Line 1291: dlthxv/=1.0-.8*exp(-(x2-x1)/50.0e3)

Note: In the FORTRAN version of this program found in Appendix B of NTIA TR-82-100, this line reads:

DLTHX=DLTHX/(1.0–0.8*EXP(-AMIN1(20.,(X2-X1)/50E3)))

The archaic intrinsic function AMIN1 has been removed in the c++ version of the code.

96. Here, subroutine ***delete []*** is called with regard to array s; to manage the removal of all traces of array s from the computer's working memory that were created by the command **new *[]*** on line 1262.

Line 1292: delete[] s:

97. The output returns the value of dlthxv; Δh, or delta h.

Line 1294: return dlthxv;

Chapter 10: Zlsq1

ZLSQ1 Subroutine ***zlsq1***

The Linear Least Squares Fit between X1, X2 to the function described by Z--.

Note: Used with point-to-point prediction and area prediction modes.

Called by ***dlthx***, while ***dlthx*** is being called by ***qlrpfl*** , after which ***qlrpfl*** may call ***zlsq1*** directly.

Special Reference for this Chapter: For the background discussion of the Linear Least Squares Fit solution, refers to equations in Chapter 15.2 of "*Numerical Recipes in C, Second Edition"* ©Cambridge University Press.

Background Notes on the Linear Least Squares Fit.

A linear least squares fit subroutine implements a mathematical procedure for finding the best-fitting curve to a given set of points by minimizing the sum of the squares of the offsets ("the residuals") of the points from the curve. The name of this subroutine suggests the use of the linear least squares fitting methodology, the simplest and most commonly applied form of linear regression. This methodology provides a solution to the problem of finding the best fitting line through a set of points. In a linear least squares fit, vertical least squares fitting proceeds by finding the sum of the squares of the vertical (y-axis) deviations of the squares of the deviations of the function values (in this case, elevations) from a straight line, along the x-axis from j=0 to j=n. The square deviations from each elevation point are therefore summed and the resulting residual is then minimized to find the best-fit line. When used in a simple mode, to find the best fitting straight line through a set of points, the process provides a solution for a, an intercept value, and b, the slope value, in the straight line equation y = a +bx .

In Tech Note 101, this equation becomes:

$$h(x) = \bar{h} + m(x - \bar{x}) \qquad (5.15a)$$

Where the h(x) term replaces the y, $\bar{h}$ replaces the intercept variable a, and the m replaces the b slope variable. The term $(x - \bar{x})$ refers to the location of the reference zero crossing, where x = 0, being relocated to a position at the center, or midpoint, of the x path.

For a full description of a set of equations used in the methodology, see Chapter 15.2, *Fitting Data to a Straight Line*, in the book "*Numerical Recipes in C, Second Edition"* ©Cambridge University Press (Numerical Recipes). A weakness of the least squares

procedure is that it results in outlying points being given disproportionately large weighting. However, if one is dealing with an even number of equal-width intervals, and, since we do not know the individual measurement errors, we set the individual measurement error factor to be equal to 1, and if we set the x,y zero crossing at the midpoint of the path, i.e. use an "x" equidistant path function that causes x to range in value from xi = ((-xa/2)+1) to xi = ((xa/2)-1) as the for loop cycles, causing "Sx"=0.0) then, solving for "a" at the midpoint of the values of "x" along the x axis (the section of the path considered), the formulas specified in 15.2.1 through 15.2.6 of Numerical Recipes simplifies so that "a" is equal to Sy, the sum of the "y" axis data point values, (elevation values along the path, divided by S, the number of "x" values (the number of intervals, represented by the argument "*xa*"). This avoids the need to use the square of the y values in the solution, minimizing the disproportionately large weighting of extremely high or low elevation values due to the squaring of the values.

So the formula to solve for "a", given the pre-conditions, simplifies to a = Sy/S, i.e.:

a = (sum of elevation values along the path)/(number of intervals)

This subroutine analyzes a central section of the total path between the tx site and the receive site, starting a short distance, set in part 2, below, from the tx site, and ending a short distance, set in part 3 below, before the receive site. This central section of the path is referred to below as the "section of the path considered".

The "a" term is solved at the point where x = 0, at the midpoint of the path considered, and is later adjusted to be the "a" value where x = 0 at the endpoint of the path considered. The terms z[0], the y value at the transmitter site, is then calculated by solving z[0]= y = a +xb where x is at the transmitter site, and then solved for z[n] = y = a +xb where x is at the receive site; the program outputs the values of z[0] and z[n].

In the same way that the "a" solution formula simplifies, the "b" solution formula given in 15.2.6 of Numerical Recipes can be simplified. To provide more detail for "b" than we did for "a":

$$b = \frac{S*Sxy - Sx*Sy}{Delta} = \frac{S*Sxy - Sx*Sy}{(Sxx - (Sx)^2)} \qquad \text{(15.2.6, Numerical Recipes)}$$

Where: (See 15.2.1 through 15.2.5, Numerical Recipes)

S simplifies to be = N, the number of intervals in the path considered.

Sx simplifies to be = 0.0, as the sum of the negative terms in the x equidistant function cancel out the positive terms as x progresses from - xa/2, through 0, to xa/2.

Sy simplifies to be = (sum of the elevation values along the section of the path considered). We will refer to this as the (sum of the elevation values).

Sxx simplifies to be = (sum of the squares of xi). This is a non-zero number, as the squares of the negative values of the x equidistant function, i.e. the individual incremental values of xi^2 from xi = ((–xa/2)+1) to zero, are positive values.

Sxy simplifies to be = (sum of x times **y** along the section of the path considered). We will refer to this as: **(**sum of **xi***y). Here x is the x equidistant function values, and y is the elevation values. Sxy is equal to the value of *b* at the completion of the ***for*** loop.

Since Sx simplifies to be =0.0, the formula for b simplifies:

$$b = \frac{S*Sxy - Sx*Sy}{(Sxx - (Sx)^2)} = \frac{S*Sxy - 0.0*Sy}{(Sxx - (0.0)^2)} = \frac{S*Sxy}{Sxx}$$

Inserting the rest of the simplified S formulas:

$$b = \frac{S*Sxy}{Sxx} = \frac{N * \text{(sum of } \mathbf{xi}*y)}{\text{(sum of the squares of xi)}}$$

Where xi is the value of the x equidistant function, or xi = - xa/2 +(n-2+1), as n progresses in value from 2 to xa-1, and xi progresses in value from (- xa/2)+1 to (xa/2)-1, in the ***for*** loop.

This solves b, the slope factor for the straight line equation y = a +bx .

However, this is not quite what the subroutine does.

The step-by-step subroutine analysis follows:

From ITMD Sections 45 and 53:

Call inputs:

z[ ]	a.k.a. z,	z(J+2), J=0,…,en, Function Values.
&x1	a.k.a. x1 (x-one)	
&x2	a.k.a. x2	
& z0	a.k.a. z0	= xi, interval length
& zn	a.k.a. z1, (z-one)	= en, number of intervals

Array z must have a special format;

z[0] = en, the number of equally large intervals,
z[1] = xi, a.k.a. sigma, the interval length,
z(j+2), j=0,….n, function values.

local declarations:

xn number of intervals

xa first the number of intervals in distance x1 (distance between transmitter site and "start consideration location"); later, the path length of the portion of the path considered by the for loop measured in number of intervals.

xb number of intervals in distance x2 (distance between transmitter site and "end consideration location")

x working variable

a working variable used to calculate "a", the intercept variable of formula y = a +xb; after for loop ends, contains sum of elevations along the considered section of the path.

b working variable used to calculate "b", the slope variable of formula y = a +xb.

n number of intervals in the section of the path considered in the for loop.

ja location of for loop consideration along path, counted in intervals from the transmitter site.

jb number of intervals in distance between transmitter site and "end consideration location")

This subroutine, ***zlsq1*** ;

98. Defines xn=z[0], setting xn = the total number of intervals between the transmitter site and the receive site, the two terminals.

Line 1117 xn=z[0];

99. Calculates value for xa by calling FORTRAN_DIM(xl/z[1],0.0)

Note: The FORTRAN_DIM function receives inputs (&x, &y)
And reports out x-y if x is greater than y; otherwise the reported result is 0.0.
So if xl/z[1] >0.0, xa=xl/z[1]-0.0, if xl/z[1] =<0.0, xa = 0.0.

x1 (x-one) is the distance between the transmitter site and the start point for consideration, at one point set to be the lesser of 1/10 of the path distance or 1/15th of the transmitter HAGL. *z[1]* is the length of one interval, so *x1/z[1]* is the number of intervals in *x1* between the transmitter site and the start point for consideration. So *xa* represents the number of intervals between the transmitter site and the start point for consideration.

Line 1118: xa=int(FORTRAN_DIM(xl/z[1],0.0));

100. Calculates value for *xb* using FORTRAN_DIM function.
If xn is > x2/z[1], *xb*=xn-xn-x2/z[1]. Otherwise, *xb* =xn-0.0.

x2 is the distance between the transmitter site and the end point for consideration, at one point set to be shorter than the total transmit to receive path length by the lesser of 1/10 of the path distance or 1/15th of the receive antenna

HAGL. *z[1]* is the length of one interval, so *x2/z[1]* is the number of intervals in *x2* between the transmitter site and the end point for consideration. *xn* is the total number of intervals in the path between the transmitter site and the receive site. So *xb* represents the number of intervals between the transmitter site and the end point for consideration.

Line 1119: xb=xn-int(FORTRAN_DIM(xn,x2/z[1]));

101. An **if** statement states that if xb is less than or equal to xa, which indicates that the total path is a very short one that causes the start point for consideration to be at or past the end point for consideration, then:

a. If xa is greater than 1, the value of xa becomes equal to xa-1, if not, xa =0.0. This reduces the value of xa by 1, unless xa is already 1.0 or less.
b. If xn is greater than (xb+1), i.e. if the total path length (defined in number of intervals) is longer than the distance from the transmitter site to the end point for consideration plus 1 interval, the value of xb is reset to be equal to xn-(xn-(xb+1)), or xb+1, increasing the value of xb by 1.
c. if xn is not greater than (xb+1), i.e. if the total path length is not longer than the distance from the transmitter site to the end point for consideration plus 1 interval, xb is reset to equal xn, the number of intervals in the total path length, effectively setting the end point for consideration to be at the receive site.

Line 1123: xa=FORTRAN_DIM(xa,1.0);
Line 1124: xb=xn-FORTRAN_DIM(xn,xb+1.0);

102. The value of *ja* is preset to be equal to xa.
103. The value of *jb* is preset to be equal to xb.
104. The value of *n* is set to be equal to *jb-ja*, the number of intervals in the section of the path for consideration.
105. The value of *xa* is reset to be equal to *xb-xa,* the number of intervals in the section of the path for consideration.
106. The value of *x* is set to equal –0.5*xa, the negative of half the number of intervals in the section of the path for consideration.
107. The value of *xb* is increased by the value of *x.* (In fact, shortens it, as x is a negative number).
108. The value of *a* is set to be 0.5*(z[ja+2]+z[jb+2]); this presets *a* to be equal to
(the elevation of the start point+the elevation of the end point)/2, or the average of the start point and the end point.
109. The value of *b* is set to be .5*(z[ja+2]; this presets *b* to be equal to ½ of the elevation of the start point.

110. A **for** loop is started at line 1136, starting at i=2, continuing until i is no longer < n. For each pass, i is incremented (increased by one), and then:

a. *++ja* This increments *ja* so that the incremented value can be immediately used. *ja* starts at the number of intervals between the transmitter site and the start point of the for loop's consideration of the path elevations, and increases by one path interval for each for loop pass.
b. *x* is increased by 1. At the beginning of the for loop, x starts at –(xa)/2, or minus ½ of the number of intervals in the section of the path considered, and has 1 (increment) added to it for each for loop pass, prior to the calculation of a and b, which causes it to end up equal to (*(xa)/2)-1)*, or one-half of the number of increments between the start point and the end point of the section of path considered, less one, when the ***for*** loop stops one increment shy of the end point. This line generates the values of the x function,
xi = - xa/2 +n+1, as n progresses from 2 to the value of (xa –1).
c. *a* is increased by *z[ja+2]*. *z[ja+2]* is the elevation of the point being considered in this loop; as the for loop continues, this causes *a* to be increased by the sum of the elevations along the path considered.
d. *b* is increased by *z[ja+2]*x*; The *z[ja+2]* term is the elevation of the point being considered in this loop; as the ***for*** loop continues, and x proceeds in value from –1/2 of the path intervals to ½ of the path intervals, this causes b to collect the sum of the interval elevations along the path multiplied by x; the end result is a positive or negative number indicating a weighted bias of the sum of the elevations around the midpoint of the path, with a negative sum (bias) for b indicating the transmitter end of the path has the higher weighted average of elevations, and a positive sum (bias) indicating the receiver end of the path has the higher weighted elevation average. The varying value of x weights the averages toward the beginning and end of the path, with the midpoint, where x passes through zero, is smallest, having the least effect.

111. Once the ***for*** loop ends, the value of "*a*" is reset to be equal to *a* divided by *xa*. The value of *a* then represents the sum of the elevations at each interval along the path length considered, divided by the number of intervals; i.e. the average of the elevations along the path considered, and represents the value of "a" in y = a +xb, solved for the condition where x is equal to 0.0 at the midpoint of the path considered. "a" will be the same for all points along the path, unless the intercept point (where x = 0) is reset. Which it will be, to the end point of the path considered, before solving for z0 and zn

Line 1144: a/=xa;

112. Now we need to solve for b, the slope of the straight-line formula y = a +xb. Here we are dealing with an even number of equal-width intervals; we do not know the individual measurement errors, so we set the individual measurement error factor to be equal to 1; and we also use the "x" equidistant function (x ranges in value from = (-xa/2)+1 to (xa/2)-1 as the for loop cycles,

causing "Sx"=0.0). As discussed above in (14.) and the introductory section "Background on the Linear Least Squares Fit", in solving for "b", the slope value, the formulas specified in 15.2.1 through 15.2.6 of Numerical Recipes simplifies so that "b" is equal to the sum of the "x times y" axis data point values, represented by the argument "b" value at the completion of the for loop), multiplied by the number of intervals (xa), and divided by the sum of the squares of the x values from the equidistant function along the path considered. So the formula to solve for "b", given the pre-conditions, should be:

b = (sum of elevation values multiplied by the x equidistant function value) *(N, the number of intervals)/(sum of the squares of the x equidistant function values)

The "sum of elevation values multiplied by the x equidistant function value", is equal to the value of *b* at the completion of the ***for*** loop.

The number of intervals, N, is equal to the value of *xa.*

But the subroutine fails to calculate the sum of the squares of the x equidistant function values along the path.

Now here we have a mystery, and perhaps errors in the original coding of the linear least squares fit algorithm:

113. At line 1145, the term "b" is "solved" to be equal to b*12.0/((xa*xa+2.0)*xa). The argument *xa* is the number of intervals along the path considered. The b value on the right hand side of the equation is the sum of each individual elevation along the path considered, multiplied by the x equidistant function value at the elevation point's increment point.

The 12/((xa*xa+2)*xa) term is incorrect.

The "12" number appears to replace the term "N" in the formula for "b" stated in (15.) above. If so, the b formula could only be correct if N = 12; i.e. if the path considered had only a fixed number of intervals = 12.

The term "(xa*xa+2)*xa" appears to be an attempt to replace the Sxx term in the b formula, as derived in the introductory section "Background on the Linear Least Squares Fit" above. "(xa*xa+2)*xa" appears to attempt to replace the Sxx term, the (sum of the squares of the x equidistant function values, xi) in the b formula from "Numerical Recipes", by taking the square of the number of intervals, xa, adding 2, and multiplying by the number of intervals, to create a sum of the squares of the number of intervals. The "2" appears to have been included to compensate for the fact that the ***for*** loop starts at *i* = 2, not *i* = 0, and increments *ja* and *x* before the *a* and *b* calculations, causing the ***for*** loop to ignore the start point elevation and the end point elevation of the section of the path considered. This calculation fails, in that the (square of the number of intervals, plus 2), (multiplied

by the number of intervals), is not the same as, is not equal to, and provides a larger calculated value than, the sum of the squares of the sequence of x values from the x equidistant function, this sequence of values defined as xi = - xa/2 incremented by 1 per elevation increment, up to (xa/2)-1 .

As a result, the value of b is generally understated. Therefore, there appear to be two problems with the subroutine; If N, the number of intervals in the section of the path considered, is not equal to 12, b is incorrect; if N =12, the substitute for the Sxx term gives a larger value, causing b to be significantly understated. See attached worksheet file for an analysis of the amount understated by length of path.

The b value now attempts to represents the b term, or slope, of the straight-line formula y = a +b*x where x = 0 at the end point of the path considered in the *for* loop.

114. The value of *z0*, the y-axis value of the equation line formula y = a +b*x, is calculated to be equal to *a-(b*xb)*. The value of *a* represents the average of the elevations along the path considered. The - (b*xb) term is the slope b times the distance xb (in increments) between the transmitter site to the end point of the path considered. *z[0]* then equals the value of y = a +b*x at the transmitter site. But since the "b" value is understated, as noted above, this value is incorrect. The b slope is understated, causing the line between z0 to zn to be "flattened" (to approach the a value) from the true value.

115. The value of *zn* the y-axis value of the equation line formula (y= a +b*x) where x is at maximum (receive site) value, is calculated to be *a+(b*(xn-xb))*. The value of *a* represents the average of the elevations along the path considered. The value of *xn* represents the number of intervals in the total path length. The value of *xb* represents the distance, measured in intervals, from the transmitter to the end point of the path considered by the ***for*** loop. So the term b*(*xn-xb)* term is the slope b times the distance (in increments) between the end point of the path considered, where x = 0, to the receive site. *z[0]* then equals the value of y = a +b*x at the receive site. But since the "b" value is understated, as noted above, this value is incorrect.

116. Outputs:
The start and end y-axis (z) values of the required line:
z0 at 0, the transmitter end of the path considered, and
zn at *x[t] =xi*en, or= z[1]*z[2],* the receive end of the path considered.

x1 and x2 are two-way arguments, both used as input and restated in the output.

How can the calculation errors in b be corrected? By:

a. ***Add a value-preset local variable, xi=0.0, to the double arguments declaration line, at Line 1114.***

b. ***Insert the line: xi+=(x*x); after line 1139, (x+=1.0;) in the for loop. At the completion of the for loop cycles, xi will represent the sum of the squares of the xi values, equal to the Sxx term in the b solution formula discussed in the "Background Notes on the Linear Least Squares Fit" section above.***

c. ***Replace line 1145, b = b*12.0/((xa*xa+2.0)*xa); with b = (b*xa)/xi;***

Chapter 11: Qtile

Quartile subroutine: ***qtile***.

Note: Used with point-to-point mode. Called by ***dlthx***, near the end of the routine.
Calls ***mymin, mymax***.

Derived from the Irregular Terrain Model description by George Hufford, 2002, (ITMD), compared to the ITM.cpp prepared by J. D. McDonald and John Magliacane for compilation on unix and linux systems. "Line" numbers refer to the ITM.cpp as line numbered by Bloodshed Software's DevC++ print function. "Alg" numbers refer to the algorithm formula in "The ITS Irregular Terrain Model, version 1.22, the Algorithm" by G. A. Hufford, 1995.

Used to find a quartile. It reorders the array *a* so that *a(j),j = 0..ir* are all greater than or equal to all *a(i),i = ir….nn.* In particular, *a(ir)* will have the same value it would have if *a* were completely sorted in descending order. The returned value is *qtile = a(ir).*

From ITMD Section 52:

Call inputs:

&nn, a constant integer representing the number of data points in the array *a*
a array with *nn* data points:
a[0]
a[1]
a[2]
a[3]

&ir the quantile desired.

declares private, or local, arguments:

q = 0.0
r
m
n number of data points in array; set equal to input nn at start of subroutine
i
j
j1 = 0
iO = 0, k;
bool done=false
bool goto10=true

This subroutine:

117. Presets *m* to be equal to 0, and n=nn . When called by ***dlthx***, nn = number of data points in array *a.*

Line 1157: m=0;
Line 1158: n=nn

118. Defines k to have a range from 0.0 to the value of n, and presets the value of k to be equal to ir.

Line 1159: k=mymin(mymax(0,ir),n);

119. Starts a ***while*** loop, that continues until *done,*defined during declaration to be equal to a Boolean <u>false</u>, is not equal to <u>true</u>.

Line 1161: while (!done)

120. The first of the series of ***if*** statements in the ***while*** loop states that if (goto10), defined during declaration to be equal to a Boolean <u>true</u>, is true, then *q* is set equal to the input *a* array value at array location a[k], *i0* is set equal to *m*, which on the first pass is 0.0, and *j1* is set equal to *n*, which on the first pass has been set equal to *nn*, the number of data points in the array *a*.

```
Line 1163:  if (goto10)
            {
                q=a[k];
                i0=m;
                j1=n;
            }
```

121. *i* is then set to be equal to *i0*, which on the first pass was set equal to *m*, which was preset to 0.0. So *i* starts at 0.0.

Line 1170: i=i0;

122. a ***while*** statement is placed inside of the ***while*** loop started on line 1161. This statement acts as long as both the value of *i* is equal or less than n, and the value of the *a* array at location a[i] is equal to or greater than q. If the while statement is true, the value of *i* is incremented, i.e. increased by 1, on each pass.

Line 1172: while (i<=n && a[i]>=q)
i++;

123. The next ***if*** statement in the ***while*** loop started at line 1161, states that if *i* is greater than *n, i* is to be set equal to *n*.

Line 1175: if (i>n)
i=n;

124. Then j is set to be equal to j1. j1 having been preset to 0.0 on the first pass, j is then preset to 0.0 on the first pass.

Line 1178: j=j1;

125. A second ***while*** statement is placed inside of the first. This one acts as long as both the value of j is equal or greater than m, which starts at zero, and the value of the *a* array value [j] is equal to or less than q. If the while statement is true, the value of *j* is negatively incremented, i.e. decreased by 1, on each pass.

Line 1180: while (j>=m && a[j]<=q)
j--;

126. The next ***if*** statement in the ***while*** loop started at line 1161, states that if *j* is less than *m, j* is set to be equal to *m*. On the first pass, m =0, so if j is less than 0, j is set to be equal to 0.0.

Line 1183: if (j<m)
j=m;

127. The next ***if*** statement in the ***while*** loop, states that if *i* is less than *j*,

a. *r* is set to be equal to the *a* loop value [i].
b. then the *a* loop value [i] is reset to be equal to the *a* loop value [j].
c. then the *a* loop value [j] is reset to be equal to *r*.
d. *i0* is reset to be equal to the value of *i* + 1.
e. *j1* is reset to be equal to the value of *j* - 1.
f. goto10 is set to be equal to Boolean <u>false</u>.

Line 1186:
```
if (i<j)
{
    r=a[i];
    a[i]=a[j];
    a[j]=r;
    i0=i+1;
    j1=j-1;
    goto10=false;
}
```

128. The ***if*** statement at line 1186 is followed by three ***else if*** statements. The first of these ***else if*** statements operates if *i* is not less than *j* at line 1186, and *i* is less than *k*; in that case,

a. the value of the *a* array value [k] is set to be equal to the value of the *a* array value [i].
b. then the *a* loop value [i] is reset to be equal to the value of *q*.
c. the value of *m* is reset to be equal to the value of *i* + 1.
d. goto10 is set to be equal to Boolean <u>true</u>.

Line 1196:

```
else if (i<k)
{
        a[k]=a[i];
        a[i]=q;
        m=i+1;
        goto10=true;
}
```

129. The ***if*** statement at line 1186 is followed by three ***else if*** statements. The second of these ***else if*** statements operates if *i* is not less than *j* at line 1186, and *j* is greater than *k*; in that case,

a. the value of the *a* array value [k] is set to be equal to the value of the *a* array value [j].
b. then the *a* loop value [j] is reset to be equal to the value of *q*.
c. the value of *n* is reset to be equal to the value of *j* - 1.
d. goto10 is set to be equal to Boolean <u>true</u>.

Line 1204:

```
else if (j>k)
{
        a[k]=a[j];
        a[j]=q;
        n=j-1;
        goto10=true;
}
```

130. The ***if*** statement at line 1186 is followed by three ***else if*** statements. The third of these ***else if*** statements operates if *i* is not less than *j* at line 1186, *i* is not less than *k* at line 1196, and *j* is not greater than *k* at line 1204; in that case done is set to be equal to Boolean <u>true</u>, completing the ***while*** loop started at line 1161.

Line 1204:

```
else
done=true;
```

131. If the ***while*** loop is complete, the ***qtile*** subroutine reports out the single quantile value *q*.

Line 1216: return q;

Chapter 12: Lrprop

Longley-Rice Propagation subroutine ***lrprop***.

Note: Used with both point-to-point and area modes. For point-to-point mode, called at end of ***qlrpfl.*** Calls subroutines ***adiff, alos, ascat, mymin,*** and ***mymax***.

From ITMD Sections 4, 5 to 9, 15; 16, and 17 with 18 and 19, and 20 with 21, 22 and 23:

The Longley-Rice propagation program. This is the basic program; it returns the reference attenuation *aref*. This is the "PaulM" version of lrprop; "Freds lrprop", found in version 1.2.2, was removed from version 7.0 when it was released on June 26, 2007.

Call inputs:

d path distance

Prop_type

&prop array *prop* with array elements:

propa_type

&propa array *propa* with array elements:

defines private, or local, arguments:

wlos static boolean argument; true if line-of-sight coefficients have been calculated.

wscat static boolean argument; true if troposcatter coefficients have been calculated.

dmin static double argument; minimum acceptable path distance length in meters

xae static double argument; value calculated in Step 11.

prop_zgnd sub array zgnd (average ground impedance) with elements:

prop.zgndreal	resistance element of ground impedance
prop.zgndimag	reactive element of ground impedance

a0
a1
a2
a3
a4
a5
a6

d0
d1
d2
d3
d4
d5
d6

wq	Boolean argument; indicates whether general case 1 or general case 2 applies in calculating line of sight coefficients.
q	working variable; holds various values during several operations
j	either 0 (1 in Fortran), for transmit terminal, or 1 (2 in Fortran) for receive terminal

This subroutine:

Uses *d*, and information in arrays *prop* and *propa* in order to calculate aref, the reference attenuation (radio signal strength loss) along the path between a transmit site and a receive location.

132. An **if** statement is initiated; it operates from lines 675 to 714. If prop.mdp, the mode of the propagation model, is not equal to zero, (zero would have indicated that the area prediction mode has initiated and is continuing), then the mode of the propagation model is either 1, which would initialize the area prediction mode, or –1, which indicates the program is running in the point to point mode. If prop.mdp = 0, the program proceeds to the **for** loop on line 677. If prop.mdp is not zero, then:

Line 675: if (prop.mdp!=0)

133. A **for** statement is initiated with two loops, j=0 and j=1.

a. The first loop sets dls[0], the distance from the transmitter site to the smooth earth horizon, to be equal to the square root of 2 times prop.he[0] divided by prop.gme.

The algorithm formula comes from: "ESSA Technical Report ERL 79-ITS 67, Prediction of Tropospheric Radio Transmission Over Irregular Terrain, A Computer Method – 1968" by A.G.Longley and P.L.Rice, where it states on page 12;
"When individual path profiles are not available, median values of the horizon distances $d_{L1,2}$ are estimated as functions of the median effective antenna heights h_{e1} and h_{e2} determined above, the terrain irregularity factor Δh, and the smooth-earth horizon distances D_{Ls1} and D_{Ls2}. The smooth earth distance from each antenna to its horizon over a <u>smooth</u> <u>earth</u> is defined as:

$D_{Ls1,2} = (.002 * a * h_{e1,2})^{.5}$ in km. [ITS67 (5a)]

where the effective antenna heights $h_{e1,2}$ are in meters and the effective earth's radius a is in kilometers, as defined by (1)."

NOTE: Note the first phrase of the quote above, authored by Longley and Rice. Ms. Longley and Mr. Rice did not intend this formula to be use where individual path profiles are available; therefore, this subroutine is eligible for revision and correction. For more information, please see the notes in the chapter on subroutine *qlrpfl*, which also uses these formulas for point-to-point calculations.

Converting ITS67 (5a) for meters instead of km, we obtain:

$D_{Ls1,2} = (2 * a * h_{e1,2})^{.5}$ in meters.

As derived in the chapter on subroutine ***qlrpfl***, in step 11:

$a = 1/ gme.$

Where: *a* is the earth's effective radius, in meters, and
g*me* is the earth's effective curvature, in units of 1/meters.

The formula then becomes:

$D_{Ls1,2} = (2 * h_{e1,2}/gme)^{.5}$ in meters. [Alg. 3.5]

And in the computer, he[0] is h_{e1}, *gme* is stored in array *prop* at *prop.gme*; and the result, $D_{Ls1,}$ is stored in array *propa* at *propa.dls[0]*.

b. Similarly, the second loop sets dls[1], the distance from the receive site to the smooth earth horizon, equal to the square root of 2 times prop.he[1] divided by prop.gme.

Line 677:
```
for (j=0; j<2; j++)
        propa.dls[j]= sqrt(2.0*prop.he[j]/prop.gme);
```

134. The program then proceeds to:

a. sets propa.dlsa, the sum of the smooth-earth horizon distance, equal to the sum of propa.dls[0] and propa.dls[1], which were calculated in step 2 above, based on:

The sum of the smooth-earth horizon distance is

$$D_{Ls} = D_{Ls1} + D_{Ls2} \qquad \text{[ITS67 (5b) or Alg. 3.6]}$$

b. sets propa.dla, the total distance between the antennas and their horizons, equal to the sum of propa.dl[0] and propa.dl[1], based on:

The total distance, $d_{L,}$ between the antennas and their horizons is

$$d_L = d_{L1} + d_{L2} \qquad \text{[ITS67 (5d) or Alg. 3.7]}$$

c. sets propa.tha to be equal to the greater of either (1) the sum of the theta angles stored in prop.the[0] and prop.the[1], or (2) the result of multiplying (– propa.dla (calculated in step 3(b)), times prop.gme, the effective earth's curvature):

tha= greater of: (the1+the1) or (-dla*gme) [Alg. 3.8]

d. sets the Boolean value of *wlos* and *wscat* to be false, as per instruction in ITMD Section 6:

Line 680:
```
propa.dlsa=propa.dls[0]+propa.dls[1];
propa.dla=prop.dl[0]+prop.dl[1];
propa.tha=mymax(prop.the[0]+prop.the[1],-propa.dla*prop.gme);
wlos=false;
wscat=false;
```

<u>In Steps 4 through 9, the program checks the parameter ranges of the input values, as per instructions in ITMD Section 7.</u>

135. An **if** statement is initiated to check if the frequency is within range; the wave number, prop.wn, which is derived from the frequency in step 2 of subroutine ***qlrps***, is checked to see if it is less than .838 (equivalent to a frequency of 40 MHz) or greater than 210 (equivalent to a frequency of 10 GHz). If prop.wn is outside of the range, prop.kwx, the error marker, is set to equal the greater of: the existing value of prop.kwx; or 1.

Line 686:
```
if (prop.wn<0.838 || prop.wn>210.0)
prop.kwx=mymax(prop.kwx,1);
```

136. A **for** statement is initiated with two loops, j=0 and j=1.

a. An **if** statement is initiated to check if hg[0], the transmitter antenna height above ground level, is within range; if hg[0] is less than one meter or greater than one kilometer, prop.kwx, the error marker, is set to equal the greater of: the existing value of prop.kwx; or 1. A value of kwx = 0 indicates: no warning; kwx = 1 indicates: caution; parameters are close to limits.

b. An **if** statement is initiated to check if hg[1], the receiver antenna height above ground level, is within range; if hg[1] is less than one meter or greater than one kilometer, prop.kwx, the error marker, is set to equal the greater of: the existing value of prop.kwx; or 1.

Line 689:

```
for (j=0; j<2; j++)
if (prop.hg[j]<1.0 || prop.hg[j]>1000.0)
prop.kwx=mymax(prop.kwx,1);
```

137. A **for** statement is initiated with two loops, j=0 and j=1.

a. A three-way **if** statement is initiated to check if the[0], the transmitter antenna take off angle theta, is within range; if either:
   (1) the absolute value of the[0] is greater than 0.2,
   (2) prop.dl[0], the distance from transmitter to horizon,, is < less than 1/10 of propa.dls[0], smooth earth distance from transmitter to horizon,
   (3) prop.dl[0] is > 3.0 * propa.dls[0]

   Then prop.kwx, the error marker, is set to equal the greater of: the existing value of prop.kwx; or 3, a value of kwx = 3 indicating that internal calculations show parameters out of range.

b. A three-way **if** statement is initiated to check if the[1], the receiver antenna take off angle theta, is within range; if either:
   (1) the absolute value of the[1] is greater than 0.2,
   (2) prop.dl[1], the distance from transmitter to horizon,, is < less than 1/10 of propa.dls[1], the smooth earth distance from transmitter to horizon,
   (3) prop.dl[1] is > 3.0 * propa.dls[1]

Then prop.kwx, the error marker, is set to equal the greater of: the existing value of prop.kwx; or 3.

Line 693:

```
for (j=0; j<2; j++)
if (abs(prop.the[j]) >200e-3 || prop.dl[j]<0.1*propa.dls[j] ||
prop.dl[j]>3.0*propa.dls[j] )
prop.kwx=mymax(prop.kwx,3);
```

138. A seven-way **if** statement is initiated to check the ranges of *ens, gme, zgnd,* and *wn*. If either:

a. *prop.ens,* the surface refractivity of the atmosphere, is less than 250.0 or greater than 400;
b. *prop.gme,* the effective earth's curvature, is less than 75e–9 or greater than 250e–9;
c. *prop.zgnd.real,* the surface transfer impedance real, (or resistance) component is less or equal to the absolute value of *prop.zgnd.imag*, the imaginary (or reactance) component;
d. *prop.wn,* the wave number, is less than 0.419 (equal to a frequency of 20 Mhz) or greater than 420 (equal to a frequency of 20 Mhz);

Then prop.kwx, the error marker, is set to 4, indicating parameters out of range.

Line 697:
```
if (prop.ens < 250.0 || prop.ens > 400.0 || prop.gme < 75e-9 || prop.gme >
250e-9 || prop_zgnd.real() <= abs(prop_zgnd.imag()) || prop.wn < 0.419 ||
prop.wn > 420.0)
        prop.kwx=4;
```

139. A **for** statement is initiated with two loops, j=0 and j=1.

a. An **if** statement is initiated to check if hg[0], the transmitter antenna height above ground level, is within its maximum range; if hg[0] is less than one-half meter or greater than three kilometers, prop.kwx, the error marker, is set to equal 4.

b. An **if** statement is initiated to check hg[1], the receiver antenna height above ground level, as in (a.) above.

Line 700:
```
for (j=0; j<2; j++)
if (prop.hg[j]<0.5 || prop.hg[j]>3000.0)
        prop.kwx=4;
```

140. The value of *dmin* is set to be equal to five times the absolute value of [(prop.he[0] – prop.he[1])], i.e. equal to five times the value of the difference in height between the effective height of the transmit antenna and the receive antenna. The *abs* command ignores any negative sign in the result, causing the result to always be a positive value.

Line 704:
```
dmin=abs(prop.he[0]-prop.he[1])/200e-3;
```

<u>From steps 10 through 20, the coefficients for the Diffraction Range are calculated:</u>

141. The program calls ***adiff*** with inputs (0.0,prop,propa) .

The subroutine ***adiff*** returns *adiff*, the "diffraction attenuation" at the distance *d*.

The value of *q* is set to be equal to the returned value *adiff*. (Note: since the input d = 0.0, the returned value of *adiff* will be 0.0 for point-to-point mode. See subroutine ***adiff***.

Line 705: q=adiff(0.0,prop,propa);

142. *xae* is set to be equal to: $(prop.wn*(prop.gme*prop.gme)^{-1/3}$ [Alg. 4.2]

Where:

prop.wn is the wave number, equal to the frequency in MHz/47.7.
prop.gme is the effective earth's curvature.

Line 707: xae=pow(prop.wn*(prop.gme*prop.gme),-THIRD);

143. *d3* is set to be equal to the greater of *propa.dlsa* or (1.3787 * *xae* + *propa.dla*): [Alg. 4.3]

where:
Propa.dlsa is the distance value set at line 680, the total smooth earth horizon distance.
xae value was set at line 707.
Propa.dla is the total horizon distance.

Line 708*:* d3=mymax(propa.dlsa,1.3787*xae+propa.dla);

144. *d4* is set to be equal to *d3* plus 2.7574 times *xae* [Alg. 4.4]

Line 709*:* d4=d3+2.7574*xae;

145. The program calls ***adiff*** with inputs (*d3*,prop,propa) . [Alg. 4.6]

The subroutine ***adiff*** returns *adiff*, the "diffraction attenuation" at the distance *d*.
a3 is set to be equal to *adiff;*

Line 710: a3=adiff(d3,prop,propa);

146. The program calls ***adiff*** with inputs (*d4*,prop,propa) . [Alg. 4.6]

The subroutine ***adiff*** returns *adiff*, the "diffraction attenuation" at the distance *d*.
a4 is set to be equal to *adiff*

Line 711: a4=adiff(d4,prop,propa);

147. *propa.emd* is set to be equal to: (*a4–a3)/(d4*–d*3*) [Alg. 4.7]

Line 712: propa.emd=(a4-a3)/(d4-d3);

148. *propa.aed* is set to be equal to: (*a3 – propa.emd*d3*) [Alg. 4.8]

Line 713: propa.aed=a3-propa.emd*d3;
}

149. The first **if** statement has run its course. A new **if** statement is initiated, as per Section 5 of the ITMD, stating that if prop.mdp is greater than or equal to zero, then:

a. prop.mdp is set to be equal to zero, indicating the area mode has initiated and will continue, and;
b. prop.dist is set to be equal to *d*, the path distance.

Line 716:
```
if (prop.mdp>=0)
{
prop.mdp=0;
prop.dist=d;
}
```

150. A third **if** statement is initiated. It has three embedded **if** statements that check the path distance [see Section 8 of the ITMD]; so if *prop.dist*, the value of *d* the path distance, is greater than zero, and:

a. if *prop.dist* is greater than 1,000 kilometers, then:
prop.kwx, the error value, is set to be equal to the higher value of *prop.kwx* or 1;

b. if *prop.dist* is less than *dmin*,then;
prop.kwx is set to be equal to the higher value of *prop.kwx* or 3;

c. if *prop.dist* is less than 1000 meters, or
prop.dist is greater than 2000 kilometers, then:
prop.kwx is set to be equal to 4;

Line 722:
```
if (prop.dist>0.0)
{
    if (prop.dist>1000e3)
        prop.kwx=mymax(prop.kwx,1);
```

```
        if (prop.dist<dmin)
                prop.kwx=mymax(prop.kwx,3);

        if (prop.dist<1e3 || prop.dist>2000e3)
                prop.kwx=4;
    }
```

151. The third **if** statement, and its three embedded **if** statements, have run their course. A fourth primary **if** statement is initiated, stating that if prop.dist is less than prop.dlsa, then:

```
Line 734:   if (prop.dist<propa.dlsa)
            {
```

<u>In steps 21 through 37, the coefficients for the Line-of-Sight Range, including the line-of-sight path loss, are calculated:</u>

152. The fourth primary **if** statement is follow by a series of embedded **if** and **else** statements; the first of these **if** statements states that: if (*wlos*) is a boolean false, indicating that the line-of-sight coefficients have not yet been calculated, then:

a. Subroutine ***alos*** is called with input (*0.0,prop,propa*). The subroutine ***alos*** returns *alosv,* the value of the line of sight attenuation, and *q* is set to be equal to *alosv.*

b. *d2* is set to be equal to *propa.dlsa*, the sum of the two smooth earth horizon distances;

c. *a2* is set to be equal to the sum of *propa.aed* and (*d2 * propa.emd*);

   where

propa.aed	is defined in step 17, above	[Alg 4.8]
propa.emd	is defined in Step16, above	[Alg. 4.7]

d. d0 is set to be equal to: (1.908**prop.wn*prop.he[0]*prop.he[1]*; [Alg. 4.28]

   where

prop.wn	is the wave number, = frequency /47.7 MHz*meters
prop.he[0]	is the effective height of the transmit antenna
prop.he[1]	is the effective height of the receive antenna

```
Line 736:   if (!wlos)
            {
                q=alos(0.0,prop,propa);
```

```
d2=propa.dlsa;
a2=propa.aed+d2*propa.emd;
d0=1.908*prop.wn*prop.he[0]*prop.he[1];
```

153. The first embedded **if** statement following the fourth primary **if** statement, states that if *propa.aed* is greater than, or equal to, zero, then:

a. *d0* is set to be equal to the lesser of: *d0* or ½ of *propa.dla*; [Alg. 4.28] where *propa.dla* is the sum of the two terminal to horizon distances, and:

b. d1 is set to be equal to: *d0*+0.25*(*propa.dla – d0*); [Alg. 4.29]

Line 743:

```
if (propa.aed>=0.0)
{
        d0=mymin(d0,0.5*propa.dla);
        d1=d0+0.25*(propa.dla-d0);
}
```

154. An ***else*** statement follows, so if *propa.aed* is less than zero, then: d1 is set to be the greater of: [-propa.aed/propa.emd] or [0.25*propa.dla] [Alg. 4.39]

Line 749:

```
else
        d1=mymax(-propa.aed/propa.emd,0.25*propa.dla);
```

155. Subroutine ***alos*** is called with input (*d1*,*prop*,*propa*). [Alg. 4.31]

The subroutine ***alos*** returns *alosv,* the value of the line of sight attenuation, and *a1* is set to be equal to *alosv.*

Line 752:

```
a1=alos(d1,prop,propa);
```

156. *wq* is then set to be equal to Boolean false.

Line 753:

```
wq=false;
```

157. The second embedded **if** statement following the fourth primary **if** statement, states than if *d0* is less than *d1*, then:

a. Subroutine ***alos*** is called with input (*d0*,*prop*,*propa*). The subroutine ***alos*** returns *alosv,* the value of the line of sight attenuation, and *a0* is set to be equal to *alosv.* [Alg. 4.30]

b. q is set to be equal to: log(*d2*/*d0*)

c. propa.ak2 is set to be equal to:

((*d2-d0*)*(*a1-a0*)-(*d1-d0*)*(*a2-a0*))/((*d2-d0*)*log(*d1/d0*)-(*d1-d0*)*q*))

or zero, whichever is greater; [Note: there is an error in Alg. 4.32 here that leaves out the log function. Correct in the code.]

d. if *propa.aed*>=0.0 or *propa.ak*2>0.0
wq is set to be equal to Boolean true;

Line 755:
```
if (d0<d1)
{
    a0=alos(d0,prop,propa);
    q=log(d2/d0);
    propa.ak2=mymax(0.0,((d2-d0)*(a1-a0)-(d1-d0)*(a2-a0))/((d2-
    d0)*log(d1/d0)-(d1-d0)*q));
    wq=propa.aed>=0.0 || propa.ak2>0.0;
```

158. A second **if** statement is embedded within the **if** statement on line 736. If *wlos* is not a Boolean "true", and if *wq* is boolean "true", then: *propa.ak1* is set to be equal to: (*a2-a0-propa.ak2**q)/(*d2-d0*). [Alg. 4.33]

Line 762:
```
if (wq)
{
    propa.ak1=(a2-a0-propa.ak2*q)/(d2-d0);
```

159. An **if** statement is embedded within the **if** statement on line 762. So:

a. If *wlos* is not a Boolean "true", and;
b. if *d0* is less than *d1*;
c. if *wq* is boolean "true", and;
d. if *propa.ak1* is less than zero;
e. then: (1.) *propa.ak1* is set to be equal to zero, and: [Alg. 4.36]
(2.) *propa.ak2* is set to be equal to (*a2 – a0)/*q if *a2* is greater than *a0;* if *a2* is not greater than *a0,* the FORTRAN_DIM function returns zero, and *propa.ak2* is set to be equal to 0.0/*q,* i.e. zero. [Alg. 4.35]

Line 766:
```
if (propa.ak1<0.0)
{
    propa.ak1=0.0;
    propa.ak2=FORTRAN_DIM(a2,a0)/q;
```

160. An **if** statement is embedded within the **if** statement on line 766, So:

a. if *wlos* is not a Boolean "true", and;
b. if *d0* is less than *d1*;
c. if *wq* is boolean "true", and
d. if *propa.ak1* is less than zero, and
e. if *propa.ak2* is equal to zero, then:

f. *propa.ak1* is set to be equal to *propa.emd.* [Alg. 4.37]

```
Line 771:                    if (propa.ak2==0.0)
                                     propa.ak1=propa.emd;
                 }
            }
```

161. At this point, the **if** statements at Lines 771, 766, and 762 have completed their run. The **if** statements at Lines 755 and 736 are still active.

162. An ***else*** statement follows, providing an alternative path to the **if** statement on line 755. Therefore:

a. if *wlos* is not a Boolean "true", and;
b. if *d0* is equal to or greater than *d1*, then:
c. *propa.ak2* is set to be equal to zero, and:
d. *propa.ak1* is set to be equal to (*a2-a1*)/(*d2-d1*);

```
Line 776:    else
                 {
                         propa.ak2=0.0;
                         propa.ak1=(a2-a1)/(d2-d1);
```

163. An **if** statement is embedded within the **else** statement on line 766, so:

a. if *wlos* is not a Boolean "true", and;
b. if *d0* is equal to or greater than *d1*, and;
c. if *propa.ak1* is less than or equal to zero,
d. *propa.ak1* is set to be equal to *propa.emd*.

```
Line 781:                    if (propa.ak1<=0.0)
                               propa.ak1=propa.emd;
                 }
            }
```

164. The **else** statement from line 776 ends its run; the **if** statements on line 755 and 736 are still active. A new ***else*** statement follows on line 786, providing an alternative path to the **if** statement on line 755. So:

a. if *wlos* is not a Boolean "true", and;
b. if *d0* is equal to or greater than *d1*, and;
c. *propa.ak1* is set to be equal to: (*a2-a1*)/(*d2-d1*); [Alg. 4.41]
d. *propa.ak2* is set to be equal to 0.0; [Alg. 4.40]

Line 791: else

```
{
        propa.ak1=(a2-a1)/(d2-d1);
        propa.ak2=0.0;
```

165. An **if** statement is embedded within the **else** statement on line 786, so:

a. if *wlos* is not a Boolean "true", and;
b. if *d0* is equal to or greater than *d1*, and;
c. if *propa.ak1* is less than or equal to zero,
d. *propa.ak1* is set to be equal to *propa.emd*.

Line 791:

```
        if (propa.ak1<=0.0)
                propa.ak1=propa.emd;
}
```

Note: In the ITM version 7.0 released June 26, 2007, the else and if statements in Steps 33 and 34 were removed, and the else statement in Step 33 becomes an if statement. The modification performs the same actions as the old code, in 10 fewer lines;

Alternate to Steps 31 to 34;

The **else** statement from line 776 ends its run; the **if** statements on line 755 and 736 are still active. A new ***if !****wq* statement follows, leading to a FORTRAN_DIM call; So:

e. if *wlos* is not a Boolean "true", and;
f. if *d0* is equal to or greater than *d1*, and;
g. *propa.ak1* is set to be equal to (*a2-a1*)/(*d2-d1*) if a2 is greater than a1; if a2 is equal to or less than a1, *propa.ak1* is set to be equal to zero. [Alg. 4.41]
h. *propa.ak2* is set to be equal to 0.0; [Alg. 4.40]

Alternate code: if (! wq)

```
{
        propa.ak1=FORTRAN_DIM(a2,a1)/(d2-d1);
        propa.ak2=0.0;
```

An **if** statement is embedded within the **if (!** *wq***)** statement, so:

i. if *wlos* is not a Boolean "true", and;
j. if *d0* is equal to or greater than *d1*, and;
k. if *propa.ak1* is equal to zero,
l. *propa.ak1* is set to be equal to *propa.emd*.

Alternate Code:

```
        if (propa.ak1= =0.0)  propa.ak1=propa.emd;
}
```

166. The **else** statement on line 786 has now completed its run. Here:

a. *propa.ael* is set to be equal to *a2 – propa.ak1 * d2 – propa.ak2 ** log(*d2*), and:
b. *wlos* is set to be equal to: Boolean" true", indicating completion of the calculation of the line-of-sight coefficients.

```
Line 795:    propa.ael=a2-propa.ak1*d2-propa.ak2*log(d2);
             wlos=true;
        }
```

<u>The next step calculates the reference attenuation, aref, for the line of sight range.</u>

167. An **if** statement is initiated. The **if** statement on line 736 is still active, so:

a. if *wlos* was not a Boolean "true" when checked by the **if** statement in Step 21, and;
b. if *prop.dist* is greater than zero, then:
c. *prop.aref* is set to be equal to:

$$propa.ael + propa.ak1 * prop.dist + propa.ak2 * \log(prop.dist)$$

[Alg. 4.1]

```
Line  799:  if(prop.dist>0.0)
            prop.aref=propa.ael+propa.ak1*prop.dist+propa.ak2*log(prop.dist);
```

168. The **if** statement on line 736 ends its run. We have finished calculating the coefficients for the Line of Sight range, and have calculated the value of aref if the path is line-of-sight from the transmit to the receive terminals.

```
Line 802:    }
```

<u>In Steps 38 to 41, coefficients are calculated for the Troposcatter (scatter) range.</u>

169. The last primary **if** statement is initiated at line 804. It has an embedded **if** statement immediately following; so if:

a. *prop.dist*, the path distance, is less than or equal to zero, or:
b. *prop.dist*, is greater than *propa.dlsa,* the sum of the calculated distances to the smooth earth horizons. This is the point, for a smooth earth condition, where diffraction mode takes over from line of sight mode.
c. and;
d. if *wscat* is not Boolean *true* (i.e. is Boolean *false),* then:

(1) subroutine ***ascat*** is called with inputs (0.0, *prop,propa*). Subroutine ***ascat*** returns *ascatv*, the value of the "scatter attenuation", and *q* is reset to be equal to *ascatv*
(2) *d5* is set to be equal to *propa.dla* + 200,000 meters. [Alg. 4.52]
(3) *d6* is set to be equal to *d5* + 200,000 meters, i.e. = *propa.dla* + 400,000 meters. [Alg. 4.53]
(4) subroutine ***ascat*** is called with inputs (*d6*, *prop,propa*). Subroutine ***ascat*** returns *ascatv*, the value of the "scatter attenuation", and *a6* is reset to be equal to *ascatv* [Alg. 4.54]
(5) subroutine ***ascat*** is called with inputs (*d5*, *prop,propa*). Subroutine ***ascat*** returns *ascatv*, the value of the "scatter attenuation", and *a5* is reset to be equal to *ascatv* [Alg. 4.55]

```
Line 804:  if  (prop.dist<=0.0 || prop.dist>=propa.dlsa)
           {
                if(!wscat)
                {
                     q=ascat(0.0,prop,propa);
                     d5=propa.dla+200e3;
                     d6=d5+200e3;
                     a6=ascat(d6,prop,propa);
                     a5=ascat(d5,prop,propa);
```

170. An **if** statement, embedded under the **if** statement at line 806, which is embedded under the primary **if** statement at line 804, is initiated. So if:

a. *prop.dist*, the path distance, is less than or equal to zero, or:
b. *prop.dist*, is greater than *propa.dlsa*, and:
c. if *wscat* is not Boolean *true* (i.e. is Boolean *false),*and;
d. if *a5* is less than 1000, then:
(1) *propa.ems* is set to be equal to: (*a6-a5*)/(200000 meters) [Alg. 4.57]
(2) *propa.dx*, the distance where diffraction mode gives way to scatter mode, is set to be equal to the greater of: [*propa.dlsa*] or [the greater of (*propa.dla* + 0.3 * *xae* * log(47.7 * *prop*.wn), or ((*a5-propa.aed-propa.ems**d5)/(*propa.emd-propa.ems*))]; [Alg. 4.58]
(3) *propa.aes* is set to be equal to:
(*propa.emd -propa.ems*) * *propa.dx* + *propa.aed* .
[Alg. 4.59]

```
Line 814:      if (a5<1000.0)
               {
                    propa.ems=(a6-a5)/200e3;
```

```
                propa.dx=mymax(propa.dlsa,mymax(propa.dla+0.3*xae*log(47.7*
                prop.wn),(a5-propa.aed-propa.ems*d5)/(propa.emd-propa.ems)));
                propa.aes=(propa.emd-propa.ems)*propa.dx+propa.aed;
        }
```

171. An **else** statement provides an alternate path to the **if** statement immediately above. So if:

a. *prop.dist*, the path distance, is less than or equal to zero, or:
b. *prop.dist*, is greater than *propa.dlsa,* and;
c. if *wscat* is not Boolean *true* (i.e. is Boolean *false),*and;
d. if *a5* is equal to or greater than 1000, then:
   (1) *propa.ems* is set to be equal to: *propa.emd.*
   (2) *propa.aes* is set to be equal to *propa.aed.*
   (3) *propa.dx* is set to be equal to: 10,000,000. [Alg. 4.56]

```
Line 821:       else
                {
                        propa.ems=propa.emd;
                        propa.aes=propa.aed;
                        propa.dx=10.e6;
                }
```

172. The value of *wscat* is then set to be equal to a Boolean "*true;*" The **if** statement at line 806 then ends its run.

```
Line 828:       wscat=true;  (Scatter coefficients calculated and ready)
        }
```

<u>The coefficients for the Troposcatter (scatter) range have now been calculated. In steps 42, 43, and 44, aref, the reference attenuation, will be computed as per [Alg. 4.1] if the path ends in the scatter (Step 43) or diffraction (Step 42) ranges.</u>

173. An **if** statement, embedded within the **if** statement at line 804, is initiated; so if:

a. *prop.dist*, the path distance, is less than or equal to zero, or:
b. *prop.dist*, is greater than *propa.dlsa,* and;
c. if *prop.dist* is greater than *propa.dx*, indicating that we are in the troposcatter (scatter) range, then:
d. *prop.aref* is set to be equal to: *propa.aes* + *propa.ems* * *prop.dist*;

```
Line 831:       if (prop.dist>propa.dx)
                prop.aref=propa.aes+propa.ems*prop.dist;
```

174. An **else** statement provides an alternative path to the **if** statement directly above, which is still embedded within the the **if** statement in Step 38; so if:

a. *prop.dist*, the path distance, is less than or equal to zero, or:
b. *prop.dist*, is greater than *propa.dlsa,* indicating we are past the line-of-sight range for a smooth earth situation, and;
c. if *prop.dist* is equal to or less than *propa.dx*, indicating that we have not yet arrived at the distance where diffraction dominance rolls over to (tropo)scatter dominance, (the combination of b. and c. therefore indicating that we are in the diffraction dominant area) then:
d. *prop.aref* is set to be equal to: *propa.aed* + *propa.emd* * *prop.dist*;

```
Line 833:       else
                        prop.aref=propa.aed+propa.emd*prop.dist;
        }
```

<u>Here we have a problem; an omission. The else statement in Step 38 only allows the else statement on Line 833 to act to switch the mode from line-of-sight to diffraction mode for a smooth earth situation, at the point where the path length exceeds the sum of the calculated smooth earth horizon distances. There is no provision to switch line-of-sight mode to diffraction mode, at the point when an obstruction blocks the horizon for the transmitter site. Therefore, the string mode propagation status printout from point_to_point, lies in outputting "Diffraction Mode" after passing the first obstacle, as the line-of-site mode actually continues until the path length exceeds the sum of the smooth earth horizon distances.</u>

<u>To make this subroutine operate to match the status printout from point_to_point, and correctly switch from line of sight mode calculation of aref to diffraction mode calculation of aref at the first obstruction, it is only necessary to add the following if statements, just after the last bracket for the if statement that starts on line 804:</u>

```
        if (prop.dist<propa.dx);
                {
                if (prop.dist>propa.dla);
                        prop.aref=propa.aed+propa.emd*prop.dist;
                }
```

175. The subroutine ***lrprop*** then sets *prop.aref,* the reference attenuation, to be equal to the greater of *prop.aref* or zero, and then returns the value of *prop.aref.*.

```
Line 837:       prop.aref=mymax(prop.aref,0.0);
}
```

Chapter 13: Adiff

Attenuation from Diffraction subroutine ***adiff***.

Note: Used with both point-to-point and area modes. Called by ***lrprop.*** Calls ***fht*** and ***aknife.***

From ITMD Section 10, 11, 12:

The function ***adiff*** finds the "diffraction attenuation" at the distance *d*. It uses a convex combination of smooth earth diffraction and double knife-edge diffraction. A call with *d* = 0 sets up initial constants.

Call inputs:

d distance from transmit site at which diffraction attenuation is to be determined.

Prop_type

&prop array *prop* with array elements:

propa_type

&propa array *propa* with array elements:

defines private, or local, arguments:

prop_zgnd an array containing values of the *zgnd* surface transfer impedance, with elements:

prop.zgndreal, the real, (resistive) component of the surface transfer impedance;
prop.zgndimag; the imaginary, (reactive) component.

wdl
xd1
afo,
qk,
aht,
xht;

a
q
pk

ds
th
wa
ar
wd
adiffv attenuation due to diffraction over an obstace at a distance *d.*

This subroutine:

Uses *d*, prop_type, propa_type, and other information in arrays prop and propa in order to calculate the attenuation due to diffraction over an obstacle at a distance d using a convex combination of smooth earth diffraction and double knife-edge diffraction.

176. An **if** statement is initiated; it operates from lines 225 to 254. If *d* is not equal to zero, go to the **else** statement in step 13 below, on line 256. If *d* is equal to zero:

a. *q* is set to be equal to prop.hg[0] times prop.hg[1], the product of the transmit antenna height above ground level, multipled by the receive antenna height above ground level; units in meters, so output is in square meters.
b. *qk* is set to be equal to the product of the effective height of the transmit antenna, multipled by the effective height of the receive antenna, less the value of *q*; output is in square meters.

Line 225:
```
if (d = = 0)
        q=prop.hg[0]*prop.hg[1];
        qk=prop.he[0]*prop.he[1]-q;
```

177. A second **if** statement is initiated within the first; if prop.mdp, the mode of the propagation model, is less than zero, indicating operation in the point-to-point mode, the value of q is increased by 10.

Line 230:
```
if (prop.mdp<0.0)
  q+=10.0;
```

178. *wdl* is set to be equal to the square root of (1 + qk/q).

Line 233:
```
wd1=sqrt(1.0+qk/q);
```

179. *xdl* is set to be equal to propa.dla + propa.tha/prop.gme.

Line 234:
```
xd1=propa.dla+propa.tha/prop.gme;
```

180. *q* is reset to be equal to the terrain irregularity parameter, dh (a.k.a. delta h or Δh), multiplied by the distance compensation term (1.0-0.8*exp(-propa.dlsa/50,000)), using a formula derived from Alg. (3.9). See subroutine ***qlrps*** step 23, for the derivation. At this point, the value of *q* represents Δh(s).

Line 235: q=(1.0-0.8*exp(-propa.dlsa/50e3))*prop.dh;

181. *q* is then further modified by setting it to be equal to the value of *q* obtained on line 235 in step 5 above, multiplied by 0.78*exp(-pow(q/16.0,0.25).

This step utilizes the formula:

$$\sigma_h(s) = 0.78\ \Delta h(s) \exp\left[-\left(\Delta h(s)/H\right)^{1/4}\right] \quad \text{with } H = 16 \text{ meters.''} \qquad \text{Alg. (3.10)}$$

This formula, is found in the Algorithm, shows the relationship between Δh and the terrain roughness factor σ_h used in Tech Note 101. Here it is used to convert the value stored in *q* from the value for Δh(s) to the value for σ_h(s).

Line: 236: q*=0.78*exp(-pow(q/16.0,0.25));

182. The value of *afo* is set to be equal to the lesser of:

a. 15
b. (2.171*log(1.0+4.77e-4*prop.hg[0]*prop.hg[1]*prop.wn*q));
   Where:
   hg[0] is the transmit antenna height above ground in meters;
   hg[1] is the receive antenna height above ground in meters;
   prop.wn is the wave number, (equal to freq. in MHz/47.7)
   q is currently equal to the value of σ_h(s), the terrain roughness factor with distance correction.

183. The value of *qk* is set to be equal to 1/(absolute value of *prop_zgnd*).

prop_zgnd is a complex double, representing the earth's surface transfer impedance with two elements; a real element, the resistance value, and an "imaginary" value, the reactance, which describes the phase mismatch between the voltage and the current, in terms of a capacitive value (current peak leads voltage peak) or a inductive value (current peak lags behind voltage peak).

Line 238: qk=1.0/abs(prop_zgnd);

184. The value of *aht* is set to be equal to 20.0.

Line 239: aht=20.0;

185. The value of x*ht* is set to be equal to 0.0.

Line 240: xht=0.0;

186. A **for** statement is initiated with two loops, j=0 and j=1. The **for** loop starts with j=0:

Line 242:
```
for (int j=0; j<2; ++j)
{
```

a. The value of *a* is set to be equal to: $0.5*(prop.dl[0])^2/prop.he[0]$;
   Where:
   prop.dl[0] is the distance from the transmit site to the horizon
   prop.he[0] is the effective height of the transmit site

Line 245: a=0.5*(prop.dl[j]*prop.dl[j])/prop.he[j];

b. The value of *wa* is set to be equal to $(a*prop.wn)^{1/3}$
   Where:
   a was determined in step 11(a.).
   prop.wn is the wave number, = (frequency in MHz/47.7)

Line 246: wa=pow(a*prop.wn,THIRD);

c. The value of *pk* is set to be equal to *qk/wa.*
   Where:
   qk was determined in step 8.
   wa was determined in the last step, 11(b).

Line 247: pk=qk/wa;

d. The value of *q* is again reset, this time to be equal to:
   ((1.607-pk)*151.0*wa*prop.dl[0]/*a*;

Line 248: q=(1.607-pk)*151.0*wa*prop.dl[j]/a;

e. The value of *xht* is increase by adding the value of *q.*

Line 249: xht+=q;

f. Subroutine ***fht*** is called with inputs *(q,pk).* Subroutine ***fht*** then returns *fhtv,* the height-gain over a smooth spherical earth for use with the three-radii method. The value of *aht* is increased by adding the value returned by ***fht.***.

Line 250:
```
aht+=fht(q,pk);
}
```

The **for** loop then repeats, with j=1:

g. The value of *a* is set to be equal to: $0.5*(prop.dl[1])^2/prop.he[1]$;
   Where:
   prop.dl[0] is the distance from the receive site to the horizon
   prop.he[0] is the effective height of the receive site

h. The value of *wa* is set to be equal to $(a*prop.wn)^{1/3}$
   Where:
   a was determined in step 11(g.).
   prop.wn is the wave number, = (frequency in MHz/47.7)

i. The value of *pk* is set to be equal to *qk/wa.*
   Where:
   qk was determined in step 8.
   wa was determined in the last step, 11(h).

j. The value of *q* is again reset, this time to be equal to:
   ((1.607-pk)*151.0*wa*prop.dl[1]/*a*;

k. The value of *xht* is increase by adding the value of *q.*

l. Subroutine ***fht*** is called with inputs *(q,pk).* Subroutine ***fht*** then returns *fhtv,* the height-gain over a smooth spherical earth for use with the three-radii method. The value of *aht* is increased by adding the value returned by ***fht.***.

The **for** loop completes, and:

187. *adiffv* is then set equal to zero.

Line 253: adiffv=0.0;
}

188. The **if** statement on line 225 has a matching **else** statement on line 256. Therefore, if the input value *d* is not equal to zero:

Line 256: else
{

a. *th* is set to be equal to *propa.tha* + *d*prop.gme*;
   Where:
   propa.tha, the total bending angle, set in ***lrprop;*** .
   d is the distance at which the attenuation is to be calculated.
   gme is the earth's effective radius.

Line 258: th=propa.tha+d*prop.gme;

b. *ds* is set to be equal to $d - propa.dla$;
Where:
d is the distance at which the attenuation is to be calculated.
propa.dla is the total horizon distance.

Line 259: ds=d-propa.dla;

c. *q* is reset to be equal to 0.0795775*prop.wn*ds*th*th; At this point, the value of *q* represents Δh(s).

Line 261: q=0.0795775*prop.wn*ds*th*th;

d. subroutine ***aknfe*** is called twice;
the first time with input: (q*prop.dl[0]/(ds+prop.dl[0])),
and the second time with input: (q*prop.dl[1]/(ds+prop.dl[1]));
in each case, ***aknfe*** reports out *a*, the attenuation due to a single knife edge diffraction; the Fresnel integral (in decibels) as a function of the input, v^2.
Adiffv is then set to equal the sum of the two outputs from ***aknfe***.

Line 262:
adiffv=aknfe(q*prop.dl[0]/(ds+prop.dl[0]))+aknfe(q*prop.dl[1]/(ds+prop.dl[1]));

e. *a* is set to equal *ds/th*

Line 263: a=ds/th;

f. *wa* is set to be equal to $(a*prop.wn)^{1/3}$

Line 264: wa=pow(a*prop.wn,THIRD);

g. *pk* is set to equal the value of *qk*/*wa.*

Line 265: pk=qk/wa;

q is reset to be equal to (1.607-*pk*) * 151.0 * *wa* * *th* + *xht*

Line 266: q=(1.607-pk)*151.0*wa*th+xht;

h. *ar* is set to be equal to 0.05751 * *q* – 4.343 * log(*q*) – *aht*

Line 267: ar=0.05751*q-4.343*log(q)-aht;

i. *q* is reset to be equal to:

(*wd1*+*xd1*/*d*)*mymin(((1.0-0.8*exp(-*d*/50e3))**prop.dh**prop.wn*),6283.2)

Line 268: q=(wd1+xd1/d)*mymin(((1.0-0.8*exp(-d/50e3))*prop.dh*prop.wn),6283.2);

j. wd is set to be equal to: (25.1/(25.1+sqrt(*q*)))

Line 269: wd=25.1/(25.1+sqrt(q));

k. *adiffv* is set to be equal to: *ar* * *wd* + (1.0-*wd*) * *adiffv* + *afo*

Line 270: adiffv=ar*wd+(1.0-wd)*adiffv+afo;
}

189. The subroutine then returns the value of *adiffv,* the "diffraction attenuation" at the distance *d*.

return adiffv;

Chapter 14: Fht

Function Height-Gain for Three-Radii method; subroutine: ***fht***.

Note: Used with both point-to-point mode and area mode. Called by ***adiff.***

From ITMD Section 14:

Calculates the height-gain over a smooth spherical earth – to be used in the "three radii" method. The approximation is that given in [Alg 6.4].

Note that in the Algorithm, in the first paragraph in Section 6, "Addenda – numerical approximations", from which the formulas below are taken, George Hufford states:

> "Part of the algorithm for the ITM consists in approximations for the standard functions that have been used. In these approximations, computational simplicity has often taken greater priority than accuracy."

Call inputs:
& *x*
& *pk*

Declares private, or local, arguments:
w
fhtv height gain over a smooth spherical earth

This subroutine:

190. Initiates an **if** statement. If *x* is less than 200, then *w* is set to be = (–log(*pk*)).

Line 133:
```
if (x<200.0)
        if (x<200.0)
        w=-log(pk);
```

191. An ***if*** statement is nested within the first ***if*** statement. If *x* is less than 200, and; *pk* is less than 1.0e–5, or if ($x*w^3$) is greater than 5,495, then:

a. *fhtv* is set to be equal to –117.0;
b. An ***if*** statement is nested within the second ***if*** statement. Therefore:
   (1) if *x* is less than 200, and:
   (2) *pk* is less than 1.0e–5, or ($x*w^3$) is greater than 5,495, and:
   (3) *x* is greater than 1.0, then:
c. *fhtv* is reset to be equal to 17.372 *log (*x)* + value of *fhtv* from 2(a).

Line 138:
```
if (pk<1.0e-5 || x*w*w*w > 5495.0)
```

```
{
        fhtv=-117.0;

        if (x>1.0)
                fhtv=17.372*log(x)+fhtv                [Alg. 6.5]
}
```

192. The second **if** statement, found on line 138, has an offsetting **else** statement. So if:

a. *x* is less than 200, and:
b. *pk* is not less than 1.0e–5, or ($x*w^3$) is not greater than 5,495, then:

fhtv is reset to be = (2.5e –5)*(x^2/pk) –(8.686*w) –15.0 [Alg. 6.6]

```
Line 145:       else
                        fhtv=2.5e-5*x*x/pk–8.686*w–15.0;
        }
```

193. The first **if** statement, found on line 133, has an offsetting **else** statement. So if *x* is greater than or equal to 200, then:

fhtv is reset to be = 0.05751*x–4.343*log(x) [Alg. 6.3; almost]

The Algorithm states that this equation, in fact, should be G(x) (a.k.a *fhtv*) = .05751x – 10 log(x). There is an undocumented fudge factor of .4343 applied here; see full discussion and mathematical proof of result mismatch in Chapter 16, Aknfe, where the fudge factor first appears (going by line number) in the ITMDLL.cpp.

An ***if*** statement is nested within this **else** statement. So if the value of *x* is greater than or equal to 200, and less than 2000, then:

a. *w* is set to be equal to 0.0134**x*exp(–0.005**x*)
b. *fhtv* is reset to be = (1.0-*w*)* *fhtv*+*w**(17.372*log(x) –117.0) [Alg. 6.4]

```
Line 149:       else
                {
                        fhtv=0.05751*x-4.343*log(x);
                        if (x<2000.0)
                        {
                                w=0.0134*x*exp(-0.005*x);
                                fhtv=(1.0-w)*fhtv+w*(17.372*log(x)-117.0);
                        }
                }
```

194. Subroutine ***fht*** then returns *fhtv,* the height-gain over a smooth spherical earth.

Chapter 15: Aknfe

Attenuation from Knife Edge Diffraction subroutine ***aknfe***.

Note: Used with both point-to-point and area modes. Called by ***adiff.***

From ITMD Section 13:

The function ***aknfe*** computes the attenuation due to a single knife edge – the Fresnel integral (in decibels) as a function of v^2. The approximation is that given in [Alg. 6.1].

Call inputs: & v2

Note: Subroutine ***adiff*** calls subroutine ***aknfe*** twice; the input value is still being calculated in the calling statement. The input value, v2, received from ***adiff*** represents the square of the value v (a.k.a. v^2) found in [TN101 7.2 and Alg. 6.1].

defines private, or local, arguments:

double *a* attenuation due to a single knife edge

This subroutine:

195. An **if** statement is initiated; if v2 is less than 5.76:
a is set to be equal to: 6.02 + 9.11 * sqrt(v2) –1.27 * v2

Line 122: if (v2<5.76)
a=6.02+9.11*sqrt(v2)-1.27*v2;

196. The following ***else*** statement provides that if v2 is ≥ 5.76,
a is set to be equal to: 12.953 + 4.343 * log(v2).

Here we have a major inconsistency.

In Tech Note 101, equation 7.2 states that

"if v is greater than 3, A(v,0) may be expressed by:

A(v,0) is approximately equal to: 12.953 + 20 log v (units in) dB. (7.2)"

In the Algorithm, Section 6., George Hufford states:

"We have (for v>0)

Fn(v) is approximately equal to:

$6.02 + 9.11*v - 1.27v^2$	if $v \leq 2.40$,	
12.953 + 20 log v	otherwise	(6.1)"

Here, the formula in the code does match the equations in TN101 and the Algorithm when v is less than 2.40 (i.e., when $(v)^2$ is less than 5.76). The equation in the code, however, has an additional factor added; instead of the log function being multiplied by 10, it is multiplied by 4.343, i.e. the 10 is being multiplied by a factor of 0.4343. This is contrary to the documentation in Tech Note 101 and in the Algorithm. This unexplained factor, and therefore inconsistent code, appears in the Appendix A to " A Guide to the Use of the ITS Irregular Terrain Model in the Area Prediction Mode", NTIA Report TR-82-100, April 1982; so this factor also exists in the FORTRAN versions of ITM 1.2.2.

To see which is correct, we test the results at the break point, where v = 2.40 and v^2 is equal to 5.76. The break point should provide an equal and smooth transition; the results for both formulas should match. By setting v2 equal to 5.76), and comparing the results from the formulas in the code, for the uncontested formula for v if $v \leq 2.40$, we get:

For v2 = 5.76, a = 6.02 + 9.11 * sqrt(v2) – 1.27*v2 = 20.569 dB

And using the formula found after the else statement:

For v2 = 5.76, a = 12.953 + 4.343*log(v2) = 16.2555 dB

By comparison, using the equation 6.1 in the Algorithm, which is mathematically the same as {TN101 7.2}:

For v2 = 5.76, and therefore v = 2.40; Fn (v) = 12.953 + 20 log *v.* = 20.557 dB

The formula in the code produces an answer that differs by more than 4 dB, a significant difference; the formula in the Algorithm and Tech Note 101 provides an answer that differs by only .012 dB; an essentially identical answer.

This is not a singular error; the same conversion factor is utilized the same way in nine locations. Including its appearance here, it also appears twice in subroutine *ascat*; twice in subroutine *h0f*; and once each in *fht*; *ahd*; *adiff*; and *alos*. Again, where the documentation allows, a 4.343 appears where a 10 should be. All appear in subroutines involved in the calculation of diffraction and scatter losses; sections

that were revised after the issuance of ESSA technical report ERL 79-ITS67,(1968). The factor does not appear in the source code documented in ERL79-ITS67.

There is no mention of this factor in the documentation found. Therefore, the obvious conclusion is that the .4343 factor is a "fudge" factor; inserted to make the program's results more closely match the measured empirical values to which they were compared. This was an obvious attempt to correct program results that are faulty due to the errors here found.

Line 124: else

```
a=12.953+4.343*log(v2);
```

197. Subroutine ***aknfe*** returns the value of *a*, the attenuation due to a single knife edge.

Line 126: return a;

Chapter 16: Alos

Attenuation for Line of Sight subroutine; ***alos***.

Note: Used with both point-to-point and area modes. Called by ***lrprop.*** Calls ***abq_alos, mymin,*** and ***mymax.***

From ITMD Sections 17, 18, 19:

The function ***alos*** computes the line-of-sight attenuation for a distance d. It uses a convex combination of plane earth fields and diffracted fields. A call with d = 0 sets up initial constants.

Call inputs:
double d
prop_type
& prop array with constants
propa_type
& propa array with constants

defines private, or local, arguments:

complex<**double**> prop_zgnd (prop_zgndreal,prop.zgndimag);
static double wls
complex<**double**> r
double s
double sps
double q
double alosv

In this subroutine:

198. An **if** statement is initiated; **if** d is equal to zero, then:

    a. *wls* is set to be equal to: 0.021, divided by (0.021+*prop.wn***prop.dh*), and all of which is divided by the greater of 10,000 or *propa.dlsa*;

where

prop.wn is the wave number, = frequency/47.7 MHz *meters
prop.dh is delta h, or Δh, the terrain irregularity parameter
propa.dlsa is the sum of the two smooth earth horizon distances; from the terminals to the horizon over smooth earth

b. alosv is set to be equal to zero.

Line 405: if (d= = 0.0)

```
{
        wls=0.021/(0.021+prop.wn*prop.dh/mymax(10e3,propa.dlsa));
        alosv=0.0;
}
```

199. An **else** statement follows, so if d is not equal to zero, then:

a. q is set to be equal to $(1.0 - 0.8^{(-d/50,000)})$*$prop.dh$;

where

prop.dh is delta h, or Δh, the terrain irregularity parameter

b. s is set to be equal to $0.78 * q * 10^{(-(q/16.0)^{0.25})}$;

c. q is set to be equal to the sum of *prop.he[0]+prop.he[1]*;

where

prop.he[0] is the effective height of the transmit antenna
prop.he[1] is the effective height of the receive antenna

d. sps is set to be equal to q/sqrt(d*d+q*q);

e. r is set to be equal to:
(sps-prop_zgnd)/(sps+prop_zgnd)*exp(-mymin(10.0,prop.wn*s*sps));

f. The subroutine ***abq_alos*** is called with input (r).

The subroutine ***abq_alos ,*** in its entirety, consists of:

```
double abq_alos (complex<double> r)
{
return r.real()*r.real()+r.imag()*r.imag();
}
```

The subroutine ***abq_alos*** returns r.real()*r.real()+r.imag()*r.imag(),
and q is set to be equal to r.real()*r.real()+r.imag()*r.imag();

```
Line 411:   else
            {
                q=(1.0-0.8*exp(-d/50e3))*prop.dh;
                s=0.78*q*exp(-pow(q/16.0,0.25));
                q=prop.he[0]+prop.he[1];
                sps=q/sqrt(d*d+q*q);
                r=(sps-prop_zgnd)/(sps+prop_zgnd)*exp(-mymin(10.0,prop.wn*s*sps));
                q=abq_alos(r);
```

200. An **if** statement is initiated; if *q* is less than 0.25 or if *q* is less than *sps*, then:

r is set to be equal to $r*(sps/q)^{1/2}$;

```
Line 420:   if (q<0.25 || q<sps)
                r=r*sqrt(sps/q);
```

201. alosv is then set to be equal to [*propa.emd* * *d* + *propa.aed*];

where:

propa.emd has been set equal to (a4-a3)/(d4-d3) in ***lrprop***
d is the path distance
propa.aed has been set equal to *a3–propa.emd*d3*

And *q* is reset to be equal to: *prop.wn*prop.he[0]*prop.he[1]**2.0/*d*;

where:

prop.wn is the wave number
prop.he[0] is the effective height of the transmit antenna
prop.he[1] is the effective height of the transmit antenna
d is the path distance

```
Line 423: alosv=propa.emd*d+propa.aed;
              q=prop.wn*prop.he[0]*prop.he[1]*2.0/d;
```

An **if** statement is initiated; if *q* is greater than 1.57, then *q* is reset to be equal to: 3.14- (2.4649/*q*).

```
Line 426:   if (q>1.57)
                q=3.14-2.4649/q;
```

202.

The subroutine ***abq_alos*** is called with input (complex<double>(cos(q),-sin(q))+r)).

The subroutine ***abq_alos ,*** in its entirety, consists of:

```
double abq_alos (complex<double> r)
{
return r.real()*r.real()+r.imag()*r.imag();
}
```

The subroutine ***abq_alos,*** also described in Chapter 4 as a utility subroutine, returns r.real()*r.real()+r.imag()*r.imag().

alosv is then reset to be equal to:

(-4.343*log((return from ***abq_alos***: r.real()*r.real()+r.imag()*r.imag() - alosv)*wls+alosv;

```
Line 429:    alosv=(-4.343*log(abq_alos(complex<double>(cos(q),-sin(q))+r))-
             alosv)*wls+alosv;
             }
```

NOTE: Here again we find the use of the fudge factor .4343 multiplied to 10, the constant found in The Algorithm, Tech Note 101, and ESSA Technical Report ERL79-ITS 67.

203. The subroutine ***alos*** returns *alosv,* the value of the line of sight attenuation;

```
Line 432:    return alosv;
             }
```

Chapter 17: Ascat

Attenuation from Scatter subroutine, (***A_{scat}***.); ***ascat***

Note: Used with both point-to-point and area modes. Called by ***lrprop.*** Calls ***mymin, mymax, h0f,*** and ***ahd.***

From ITMD Sections 22, 23, and 24:

The function ***ascat*** finds the "scatter attenuation" for the path distance *d*. It uses an approximation to the methods of NBS TN101 with checks for inadmissible situations. For proper operation, the larger distance ($d = d_6$ must be the first called. A call with $d = 0.$ sets up initial constants.

From the Algorithm, Section 4.3.1:

Computation of this function uses an abbreviated version of the methods described in Section 9 and Annex III.5 of NBS Tech Note 101.

Note: THIS SUBROUTINE IS ELIGIBLE FOR UPDATE AND REVISION: On page 11 of the Algorithm, last paragraph, George Hufford stated:

"A difficulty with the present model is that there is not sufficient geometric data in the input variables to determine where the crossover point is. This is resolved by assuming it to be midway between the two horizons."

This statement and concept should be reviewed to determine if the "sheer magnitude" of data available in today's terrain databases is adequate to implement a more accurate geometric determination of the crossover point.

For now, here is how the current version of the ITM.cpp works:

Note: Not defined in the Algorithm section 4.3.1 below, the *k* in [Alg. 4.62] is the wave number, which is equal to the frequency in MHz divided by 47.7.

From the Algorithm, with clarifications:

"First, set:

$\theta = \theta_e + \gamma_e^s$ [Alg. 4.60]

$\theta' = \theta_{e1} + \theta_{e2} + \gamma_e^s$ [Alg. 4.61]

$r_j = 2 * k * \theta' * h_{ej}$ *for* $j = 1, 2.$ [Alg. 4.62]

If both r_1 and r_2 are less than 0.2, the function A_{scat} is not defined, (or is infinite).

Otherwise, we put

$$A_{scat}(s) = 10 * \log(k*H*\theta^4) + F(\theta_s, N_s) + H_0 \qquad (4.63), \text{ i.e. [Alg. 4.63]}$$

Where $F(\theta_s, N_s)$ is the function shown in Figure 9.1 of Tech Note 101, H_0 is the "frequency gain function", and H is 47.7 meters.

The frequency gain function H_0 is a function of:

r_1, defined in {Alg. 4.62]
r_2, defined in {Alg. 4.62]
η_s, the scatter efficiency factor, and
the "asymmetry factor", which we shall here call s_s.

A difficulty with the present model is that there is not sufficient geometric data in the input variables to determine where the crossover point is. This is resolved by assuming it to be midway between the two horizons The asymmetry factor, for example, is found by first defining the distance between horizons

$$d_s = s - d_{L1} - d_{L2} \qquad \text{[Alg. 4.64]}$$

whereupon

$$s_s = (d_{L2} + d_s/2) / (d_{L1} + d_s/2) \qquad \text{[Alg. 4.65]}$$

There then follows that the height of the crossover point is

$$z_0 = (s_s * d * \theta') / (1 + s_s)^2 \qquad \text{[Alg. 4.66]}$$

[Ed. where
d is the total path distance
θ' is the angular distance, defined in [Alg. 4.61], a.k.a. θ_{oo}, as shown on Figure 6.1 of TN101 on page 6 – 8.]

and then

$$\eta_s = (z_0 / Z_0) * [1 + (0.0.031 - N_s * 2.32 * 10^{-3} + N_s^2 * 5.67 * 10^{-6}) e^{-(z0/Z1)^6}]$$

(4.67), i.e. [Alg. 4.67]

where

Z_0 = 1.756 km or 1756 meters
Z1 = 8.0 km or 8000 meters
[Ed. (and N_s is the surface refractivity of the atmosphere, a.k.a *ens* or *prop.ens)]*

The computation of H_0 then proceeds according to the rules in Section 9.3 and Figure 9.3 of Tech Note 101.

The model requires these results at the two distances $s = d_5$, d_6 described above. One further precaution is taken to prevent anomalous results. If, at d_5, calculations show that H_0 will exceed 15 dB, they are replaced by the value it has at d_6. This helps keep the scatter mode slope within reasonable bounds."

Discussion:

TN101 defines "launch angles" for the signal path line as it leaves the transmit antenna toward the receive antenna, and from the receive antenna as received from the transmit antenna. These launch angles are defined in Section 6.4 of TN101, and are designated as θ_{et}, the angular elevation of the transmit horizon ray, and θ_{er}, the angular elevation of the receive horizon ray. They are shown on Figure 6.1 of TN101. They are calculated in subroutine ***hzns*** and provided to ***ascat*** as values stored in *prop.the[0]* and *prop.the[1]*.

The equation [Alg. 4.60] can be confusing here, due to its poorly defined use of θ. Alg. 4.60 is attempting to explain that in the general case, a launch angle must be adjusted for the earth's curvature; and is easier to understand if we give examples:

$$\theta_t = \theta_{et} + \gamma_{et}^{s} \qquad \text{[Alg. 4.60a]}$$
$$\theta_r = \theta_{er} + \gamma_{er}^{s} \qquad \text{[Alg. 4.60a]}$$

where

θ_{et} is the transmit site launch angle as shown on TN101 Figure 6.1.
θ_{er} is the receive site launch angle as shown on TN101 Figure 6.1.
s is either the asymmetry factor, or a distance
at the terminals, $\gamma_{e(t,r)}^{s} = d_{L(t,r)} / a = d_{L(t,r)} * gme$,

where

d_{Lt} is the distance from the transmit site to the horizon or obstacle, in meters.
d_{Lr} is the distance from the receive site to the horizon or obstacle, in meters.
gme is the earths curvature, equal to $1/a$ where a is the effective earths radius shown in Figure 6.1 of TN101. Here, a is in meters, so *gme* is in 1/meters.

In TN101, the angular distance θ is defined as:

$$\theta = \theta_{oo} = d / a + \theta_{et} + \theta_{er} \qquad \text{[TN101 6.14]}$$

where, as described in section 6.4 and on diagram 6.1 of NBS TN101;
d is the total path distance as shown on figure 6.1
a is the effective earth's radius, equal to $1/gme$, the effective earth's curvature..

θ_{et} is the angle between the horizontal, at the transmitter site, and a line between the transmit antenna and the horizon (or the top of the obstruction). This is the value stored in *prop.the[0]*.
θ_{er} is the angle between the horizontal, at the receive site, and a line between the receive antenna and the horizon (or the top of the obstruction). This is the value stored in *prop.the[1]*.

In [Alg. 4.61], we again find γ_e^s replacing the d/a term in [TN101 6.14].

The angular distance θ_{oo} is the angle, as shown on diagram 6.1 of NBS TN101, between a line starting from the transmit antenna and touching the transmit horizon or the top of the receive obstacle, and a line starting from the receive antenna and touching the receive horizon or the top of the receive obstacle.

TN101 also defines "launch angles" for the signal path line as it grazes the horizon, (or crosses the top of the obstacles). These launch angles, or angular elevation of a horizon ray, are defined as the angle between the horizontal at the horizon (or obstacle), and the signal path line grazing the horizon (or touching the top of the obstacle), and in TN101 are designated as θ_{ot}, the angular elevation of the transmit horizon ray, and θ_{or}, the angular elevation of the receive horizon ray, as shown on Figure 6.1 of TN101. These are calculated using:

$\theta_{ot} = \theta_{et} + d_{Lt}/a$ $\theta_{or} = \theta_{er} + d_{Lr}/a$ [TN101 6.16]

where

d is the total path distance as shown on figure 6.1, and input to ***ascat*** as input *d*
θ_{et} is the angle between the horizontal, at the transmitter site, and a line between the transmit antenna and the horizon (or the top of the obstruction). This is the value stored in *prop.the[0]*.
θ_{er} is the angle between the horizontal, at the receive site, and a line between the receive antenna and the horizon (or the top of the obstruction). This is the value stored in *prop.the[1]*.
d_{Lt} is the distance from the transmit site to the horizon or obstruction, stored in *prop.dl[0]*.
d_{Lr} is the distance from the receive site to the horizon or obstruction stored in *prop.dl[1]*.
a is the effective earth's radius, equal to 1/*gme,* the effective earth's curvature, the value of which is stored in *prop.gme.*

If the earth is smooth, θ is approximately equal to D_s/a,
where:
"a" is the effective earth's radius (equal to 1/*gme*), and;

D_s (a.k.a. *ds* in the code) is the distance between the transmitter site horizon (or obstacle) location, and the receive site horizon (or obstacle) location.

and where: $D_s = d - d_{Lt} - d_{Lr}$ [TN101 6.17]

In order to properly calculate tropospheric scatter losses, Longley-Rice generates a "path asymmetry factor, identified in NBS TN101 as "s". In order to do this, TN101 starts by defining the angles α_{oo} and β_{oo}. On page 6.8 of TN101, Volume I, Figure 6.1 shows a graphic representation of the two angles. α_{oo} is the angle at the transmit site between a line drawn from the transmit antenna and grazing the horizon or tallest visible obstacle, and a theoretical line drawn directly from the transmit antenna to the receive antenna (passing through the earth for a beyond-the-horizon path). β_{oo} is the same angle from the point of view of the receive antenna.

In NBS TN101, for the general case of irregular terrain, the angles α_{oo} and β_{oo} are calculated using:

$$\alpha_{oo} = (d/2 * a) + \theta_{et} + (h_{ts} - h_{rs})/d \qquad \text{[TN101 6.18a]}$$

$$\beta_{oo} = (d/2 * a) + \theta_{er} + (h_{rs} - h_{ts})/d \qquad \text{[TN101 6.18b]}$$

where:

d is the total path distance as shown on figure 6.1, and input as *d*
a is the effective earth's radius, equal to 1/*gme,* the effective earth's curvature, the value of which is stored in *prop.gme.*
θ_{et} is the angle between the horizontal, at the transmitter site, and a line between the transmit antenna and the horizon (or the top of the obstruction). This is the value stored in *prop.the[0].*
θ_{er} is the angle between the horizontal, at the receive site, and a line between the receive antenna and the horizon (or the top of the obstruction). This is the value stored in *prop.the[1].*
h_{ts} is the transmit site antenna elevation, stored in *prop.he[0]*
h_{rs} is the receive site antenna elevation, stored in *prop.he[1].*

These angles are positive for beyond-horizon paths. To allow for the effects of a non-linear refractivity gradient, α_{oo} and β_{oo} are modified by corrections $\Delta\alpha_o$ and $\Delta\beta_o$

where:

α_o is defined as $\alpha_{oo} + \Delta\alpha_o$ [TN 6.19a]

and

β_o is defined as $\beta_{oo} + \Delta\beta_o$ [TN 6.19b]

To give the angles α_o and β_o, whose sum is the angular distance theta, θ, and whose ratio defines a path asymmetry factor "s".

$$\theta = \alpha_o + \beta_o \qquad s = \alpha_o / \beta_o \qquad \text{[TN 6.19c]}$$

The corrections $\Delta\alpha_o$ and $\Delta\beta_o$ are functions of the angles θ_{ot} and θ_{or}, (see [TN 6.16], and of the distances d_{st} and d_{sr} from each horizon obstacle to the point where the horizon rays cross over. These distances are approximated as:

$$d_{st} = d\,(\beta_{oo} / \theta_{oo}) - d_{Lt}\,, \quad d_{sr} = d\,(\alpha_{oo} / \theta_{oo}) - d_{Lr} \qquad \text{[TN 6.20]}$$

The sum of distances d_{st} and d_{sr} is the distance D_s between horizon obstacles, defined by [TN 6.17]. Over a smooth earth, $d_{st} = d_{sr} = D_s$.

For small θ_{ot} or θ_{or}, no correction $\Delta\alpha_o$ or $\Delta\beta_o$ is required for values of d_{st} or d_{sr} less than 100,000 meters. When both $\Delta\alpha_o$ or $\Delta\beta_o$ are negligible;

$$\theta = \theta_{oo} = \alpha_{oo} + \beta_{oo} \qquad \text{[TN101 6.22]}$$

which is the same as [TN101 6.14], i.e.:

$$\theta = \theta_{oo} = \alpha_{oo} + \beta_{oo} = d / a + \theta_{et} + \theta_{er} \qquad \text{[TN101 6.14 with 6.22]}$$

If either θ_{ot} or θ_{or} is negative, indicating an obstruction taller than the terminal height, then compute:

$$d'_{st} = d_{st} - |\,\alpha * \theta_{ot}\,| \quad \text{or} \quad d'_{sr} = d_{sr} - |\,\alpha * \theta_{or}\,| \qquad \text{[TN101 6.23]}$$

substitute d'_{st} for d_{st} or d'_{sr} for d_{sr}, and read figure 6.9, on page 6-16 of TN101, using $\theta_{ot} = 0$ or $\theta_{ot} = 0$. [needless to say; the compeuter code cannot do this without using an approximation.

If either θ_{ot} or θ_{or} is greather than 0.1 radian and less than 0.9 radian, determine $\Delta\alpha_o$ or $\Delta\beta_o$ for $\theta_{ot} = 0.1$ radian and add the additional correction term

$$N_s\,(9.97 - \cot\theta_{ot,r})\,[\,1 - \exp(-0.05 * d_{st,r})\,] * 10^{-6} \text{ radians}$$

The bending of radio rays elevated more than 0.9 radian above the horizon and passing all the way through the atmosphere is less than 0.0004 radian, and may be neglected.

Referring to the information obtained from the Algorithm above:

The equation for η_s, the scatter efficiency factor, given in [Alg. 4.67], is derived from an equation in TN101:

$$\eta_s = 0.5696 * h_o * [1 + (0.0.031 - N_s * 2.32 * 10^{-3} + 5.67 * N_s^2 * 10^{-6}) \exp(-3.8 * h_o^6 * 10^{-6})]$$

[TN101 9.3a]:

where

z0 is represented by h_o

Z_0, stated as 1,756 meters in the Algorithm, is replaced by 1.7556 kilometers, so 1/1.7556 = 0.5696;

Z1 = 8.0 km or 8000 meters, and $[1/(8.0)]^6 = (.125)^6 = 3.8$

Call inputs:

d	the total path distance
*P*rop_type	
&prop	array with elements
propa_type	
&propa	array with elements

defines private, or local, arguments:

Note: The following four arguments are static doubles:

ad	absolute value of the difference in distance between: the distance from the transmit site to the horizon, and the distance from the receive site to the horizon.
rr	ratio of the higher terminal's effective height, (transmit or receive site antenna) to the lower terminal's effective height
etq	a term in the equation for η_s
h0s	frequency gain function for (s) smooth earth
h0	frequency gain function
r1	transmit site angle calculated in [Alg. 4.62]
r2	receive site angle calculated in [Alg. 4.62]
z0	the height of the crossover point of the horizon or obstacle grazing lines from the terminal antennas, above a line drawn between the two terminal antennas
ss	a.k.a. s_s the "asymmetry factor"
et	
ett	
th	*a.k.a.* theta prime, or θ', the combined launch angle calculated in [Alg. 4.61]
q	

In this subroutine:

1. An **if** statement is initiated to prepare the initial scatter constants; if *d* is equal to zero, then:

a. *ad* is set to be equal to the difference in distance, in meters, between: the distance from the transmit site to the horizon, *prop.dl[0],* and the distance from the receive site to the horizon, *prop.dl[1]*.

b. *rr* is set to be equal to the effective height of the transmit antenna, *prop.he[0]*, in meters, divided by the effective height of the receive antenna, *prop.he[1]*, in meters. At this moment, the argument *rr* represents the ratio of the transmit antenna effective height to the receive antenna effective height.

Line 282:
```
if (d= =0.0)
{
        ad=prop.dl[0]–prop.dl[1];
        rr=prop.he[1]/prop.he[0];
```

2. A second **if** statement is initiated, nested within the first; so if *d* is equal to zero, and if *ad* is less than zero, then:
   a. *ad* is made equal to *–ad;* as a result, the always positive resulting value stored in *ad* will represent the difference between the distances to the horizon from the transmit site and the receive site, measured in meters.
   b. *rr* is inverted, i.e. made equal to 1/*rr*. The argument *rr* then represents the dimensionless, and always positive, ratio of the higher terminal's effective height, (transmit or receive site antenna) to the lower terminal's effective height.

Line 287:
```
if (ad<0.0)
{
        ad=-ad;
        rr=1.0/rr;
}
```

3. The subroutine then continues under the first if statement, if d=0, to:
   a. Set *etq* equal to [(5.67e–6∗*prop.ens*–2.32e–3)**prop.ens*+0.031].

What is this for? from: [Alg. 4.67]:
$\eta_s = (z_0 / Z_0) * [1 + (0.0.031 - N_s * 2.32 * 10^{-3} + N_s^2 * 5.67 * 10^{-6})e^{-(z0 / Z1)^{\wedge}6}]$;
from this equation, we take the term: $(0.0.031 - N_s * 2.32 * 10^{-3} + N_s^2 * 5.67 * 10^{-6})$;
re-ordering the term, we get: $[(5.67 * 10^{-6} * N_s - 2.32 * 10^{-3})* N_s + 0.0.031]$;
replacing N_s, the surface refractivity of the atmosphere, with the value of N_s a.k.a. *ens*, stored in *prop.ens*, we get: [(5.67e–6∗*prop.ens*–2.32e–3)**prop.ens*+0.031]. This term is later used to calculate the value for argument *et* , a.k.a. η_s, at line 321.

NOTE: There is a QUIRK here, AS *etq*, a **static double** argument, ONLY GETS CALCULATED IF D == 0. LRPROP does call ASCAT first with d = 0, at line 808, before calling with d = d6 and d5.

b. Set *h0s, a.k.a.* $H_0(s)$, or the H_0 frequency gain function over (s), smooth earth, equal to: –15.
c. Set *ascatv* equal to 0.0.

Line 293:
```
etq=(5.67e-6*prop.ens-2.32e-3)*prop.ens+0.031;
h0s=-15.0;
ascatv=0.0;
```

If at line 282, d was equal to zero, the program here jumps to line 345, and the program ends by returning a value of 0.0 for *ascatv.*

If at line 282, d was not equal to zero, the program ignores lines 282 to 297, and proceeds to line 298, where:

4. An ***else*** statement follows, and its action affects line 300 to 343.
If *d* is not equal to zero, then the program proceeds to an **if** statement nested within the ***else*** statement. If *h0s* is greater than 15, then *h0* is set to be equal to *h0s*.

Line 298:
```
else
{
    if (h0s>15.0)
        h0=h0s;
```

QUESTION: *h0s* is a **static double** type argument; is its value retained when the subroutine is not online? Does it come from the calling routine ***lrprop***? Need to determine the source of this value. Unless d=0, there is no value preset for *h0s*.

5. A second ***else*** statement at line 302, nested within the first ***else*** statement, provides an alternate path to the **if** statement nested within the first ***else*** statement. This else statement affects lines 304 to 338. So if *d* is not equal to zero, and if *h0s* is less than or equal to 15, then:

   a. *th*,(a..k.a.theta, or in TN101 θ_{oo}, or in the Algorithm, *θ'*, theta prime), the angular distance, is set to be equal to:

   *prop.the[0]+prop.the[1]+d*prop.gme*; [Alg.4.61] or [TN101 6.14]

   where
   prop.the[0] is the launch angle from the transmit antenna
   prop.the[1] is the launch angle from the receive antenna
   d is the total path distance
   prop.gme is the effective earth's curvature, which is equal to 1/a, where a is the effective earth's radius

b. *r2* is set to be equal to 2 * *prop.wn* * *th*; this is a part of [Alg 4.62), in that *r2* is momentarily set to be equal to the term 2 * *k* * *θ'*, where:

prop.wn, is the wave number, *k*, equal to the frequency in MHz divided by 47.7, as defined in [Alg. 1.1] ;
th is *θ'*, or theta prime, the angular distance calculated in step a. above, as per [Alg.4.61].

c. *r1* is set to be equal to *r2* * *prop.he[0]*; calculated as per [Alg. 4.62].

where
r2 was determined in step b. above
prop.he[0] is the effective height of the transmit site antenna (in meters)

d. *r2* is then reset to be equal to the value of *r2* from step b. above, multiplied by the value of *prop.he[1]*, the effective height of the receive antenna. At this point, *r1* and *r2* have been calculated as per [Alg. 4.61].

This comes from the equations for the Frequency Gain Function, H_o, in Section 9.2 of TN101. On page 9–3 of TN101, the parameters r_1 and r_2 are defined as:

$$r_1 = 4 * \pi * \theta * h_{te,}/\lambda \quad \text{and } r_2 = 4 * \pi * \theta * h_{re,}/\lambda \qquad \text{[TN101 9.4b]}$$

In TN101, Section 9.2, the dimensions can be said to be either meters or kilometers for the wavelength λ, and for the effective antenna heights, $h_{te,}$ and h_{re}, as long as all are specified in either meters or kilometers; these units of measure cancel out. The angular distance θ is specified in radians in TN101 and in the Algorithm. This is not true in the code. Radians, in mathematics, are usually assumed to be the standard unit of angular measure, so the unit "rad" is customarily omitted, contributing to the confusion in attempting to study Longley Rice as it translates from TN101 to the computer code. In the ITM FORTRAN and c++ code, the angular distance θ is specified in a unique ratio; the ratio of vertical distance change to horizontal distance change. The units must cancel out (both the numerator and denominator must be specified in the same units). The change in units for θ is not the only difference in r_1 and r_2. In the equation for r used in the code, the wave number is used instead of the wavelength.

The Algorithm, defines the wave number to be that of the carrier or central frequency. It is defined to be:

$$k = (2 * \pi / \lambda) = f / f_o \quad \text{with } f_o = 47.70 \text{ MHz * meters.} \qquad \text{[Alg. 1.1]}$$

Resorting r_1 and r_2 , we get:

$$r_1 = 2 * \theta * h_{te} * (2 * \pi / \lambda) \quad \text{and } r_2 = 2 * \theta * h_{re} * (2 * \pi / \lambda) \qquad \text{[TN101 9.4b]}$$

Replacing the term $(2 * \pi / \lambda)$ with *k:*

$$r_1 = 2 * \theta * h_{te} * k \quad \text{and } r_2 = 2 * \theta * h_{re} * k \text{, which is the same as [Alg. 4.62]}$$

The distance units, if all in meters, cancel out. So we have now successfully converted from wavelength to wave number. But in the equation [Alg. 4.62], in Section 4.3.1, "The Function A_{scat}." the angular distance θ is still being specified in radians, as becomes clear in Section 6, equations [Alg 6.13 and 6.14}.

How do we convert θ, a.k.a. *th*, to radians? There are 2π radians in a full cycle, or 360°. A radian is defined as the angle subtended at the center of a circle by an arc of circumference that is equal in length to the radius of the circle. Draw this construct on a circle, with one radii of length r on the horizontal plane, and a distance of r on the circumference between the two radii. Now draw a vertical line from the point where the non-horizontal radii touches the circumference of the circle, to a point perpendicular to the horizontal radii, forming a right triangle. The radius then becomes the hypotenuse of a right triangle with an angle, subtended at the center of the circle, between the two radii, of one radian, or 57.2958 degrees. The length of the vertical line is then equal to the sine function of the angle θ, which is equal to the ratio of the length of the vertical line to the length of the hypotenuse of the triangle, which is equal to r. So we can now obtain the length of the vertical line by multiplying sin θ by the hypotenuse length, r. This results in the equation:

$$V = (\sin \theta) * r$$

The length of the horizontal line is then equal to the cosine function of the angle θ, which is equal to the ratio of the length horizontal line of the triangle to the length of the hypotenuse of the triangle, which is equal to r. So we can obtain the length of the horizontal line by multiplying cos θ by the hypotenuse length, r. This results in the equation:

$$H = (\cos \theta) * r$$

We can now obtain the ratio of the vertical length to the horizontal length by dividing the equation for Y by the equation for X. and canceling out the "r" terms:

$$V/H = [(\sin \theta) * r] / [(\cos \theta) * r] = (\sin \theta)/(\cos \theta)$$

In trigonometry, by definition of the tangent function, $\tan x = (\sin x) / (\cos x)$, so the equation becomes:

$$V/H = (\tan \theta) \text{ in radians}$$

This can be used to convert from the angle in radians or degrees, to the ratio used for θ in the code, but we also need to know how to convert from the vertical-distance-to-horizontal-distance ratio (V/H ratio) used for *th*, to radians, in case we later run into a formula that cannot handle the V/H ratio. For this we use the arctan subroutine function:

arctan (V/H) = θ in radians (rads)

We will need these conversion factors later in the subroutine, and in subroutine ***h0f*** when we call it.

Line 298:
```
else
{
        th=prop.the[0]+prop.the[1]+d*prop.gme;
        r2=2.0*prop.wn*th;
        r1=r2*prop.he[0];
        r2*=prop.he[1];
```

6. An **if** statement is nested in the **else** statement at this point. If *r1* is less than 0.2 and if *r2* is less than 0.2, then the function A_{scat} is not defined, (or is infinite), and the subroutine returns 1001.0 as the value of *ascatv*. The subroutine ends early, and returns to the calling subroutine, ***lrprop***.

Line 309:
```
if (r1<0.2 && r2<0.2)
    return 1001.0;  // <==== early return
```

7. If *r1* is equal to or more than 0.2, or if *r2* is equal to or more than 0.2, the program continues under the **else** statement, and:

   a. ss, a.k.a. s_s the "asymmetry factor" over smooth earth, is set to be equal to:
      $=(d-ad)/(d+ad)$;

      where:
      d is the total path distance
      ad is the absolute value of the difference between the distances to the horizon from the transmit site and the receive site, measured in meters.

Which at first glance, appears to bear no relationship to the equations described in the Algorithm. However, In the Algorithm, the asymmetry factor is found by first defining the distance between horizons

$$d_s = s - d_{L1} - d_{L2} \qquad \text{[Alg. 4.64] also [TN101 6.17]}$$

From TN101, page 6-6, it states:

"The sum of d_{st} and d_{sr}, the distances from each horizon obstacle to the crossover of horizon rays, is the distance D_s (this D_s is the same as d_s in the Algorithm and c++ code). Over a smooth earth $d_{st} = d_{sr} = D_s / 2$.

The Algorithm states: "A difficulty with the present model is that there is not sufficient geometric data in the input variables to determine where the crossover point is. This is resolved by assuming it to be midway between the two horizons." Therefore, from this assumption, $d_s/2$ is equal to the distance from each horizon to the crossover point.

whereupon

$$s_s = (d_{L2} + d_s/2) / (d_{L1} + d_s/2) \qquad \text{[Alg. 4.65]}$$

where:

d_{L2} is the distance from the receive site to the horizon
d_{L1} is the distance from the transmit site to the horizon
d_s is the distance between the transmit horizon and the receive horizon

And, $d_s = d - d_{L2} - d_{L1}$, the same as: [Alg. 4.64] also [TN101 6.17]

Or, restated; d, the total path distance, equals the sum of $d_{L2} + d_{L1} + d_s$

Therefore, $d_{L2} = - d_{L1} - d_s + d$, and $d_{L1} = - d_{L2} - d_s + d$,

So, substituting into [Alg. 4.65] we get:

$$s_s = (-d_{L1} - d_s + d + d_s/2) / (- d_{L2} - d_s + d + d_s/2)$$

by reordering: $s_s = (d - d_{L1} - d_s + d_s/2) / (d - d_{L2} - d_s + d_s/2)$
by consolidating: $s_s = (d - d_{L1} - d_s/2) / (d - d_{L2} - d_s/2)$
by multiplying both the numerator and denominator by 2:

$$s_s = (2d - 2d_{L1} - d_s) / (2d - 2\, d_{L2} - d_s)$$

and substituting $d - d_{L2} - d_{L1}$ *for* d_s

$$s_s = (2d - 2d_{L1} - d + d_{L2} + d_{L1}) / (2d - 2\, d_{L2} - d + d_{L2} + d_{L1})$$

by consolidating: $s_s = (d - d_{L1} + d_{L2}) / (d - d_{L2} + d_{L1})$
by reordering: $s_s = [d - (d_{L1} - d_{L2})]/ [d + (d_{L1} - d_{L2})]$

Since *ad* has been made equal to the absolute value of prop.dl[0]–prop.dl[1], where prop.dl[0] = d_{L1} , and prop.dl[1] = d_{L2} , then the value of ad is equal to the term ($d_{L1} - d_{L2}$), and equations [Alg. 4.64 and 4.65] merge to create:

$$s_s = [d - (ad)]/ [d + (ad)]$$

Therefore the value of *ss*, which represents s_s the "asymmetry factor" over smooth earth, is set to be equal to: (*d-ad*)/(*d+ad*).

An interesting point here is that *ad* at this point has been set to be the absolute value of *ad* in step 2 above. This refers to the procedure mentioned in the last paragraph on page 6-7 of TN101, where it states:

"Many of the graphs in this a subsequent sections assume that s is $\leq$ 1 (Ed. here TN101 is referring to "s" as the "asymmetry factor). It is therefore convenient, since the transmission loss is independent of the actual direction of transmission, to denote as the transmitting antenna whichever antenna will make s less than or equal to unity."

By using the absolute, or always positive, value of *ad,* we make sure that *ss* will be less than 1, as the numerator will always be smaller than the denominator.

Line 312: ss=(d-ad)/(d+ad);

b. in the next step, q is set to be equal to the ratio of rr/ss, where:
The argument *rr* represents the dimensionless, and absolute, (i.e. always positive), ratio of the higher terminal's effective height above ground level, (transmit or receive site antenna) to the lower terminal's effective height above ground level.
ss is the asymmetry factor over smooth earth.

Line 313: q=rr/ss;

c. the value of ss (a..k.a. s_s), the asymmetry factor, is then set to be no less than 0.1;

Line 314: ss=mymax(0.1,ss);

In section 9.2 of TN101, "The Frequency Gain Function, H_o " it states:

"For the great majority of transhorizon paths, s is within the range $0.7 \leq s \leq 1$. The effect of very small values of s, with $\alpha_o << \beta_o$, may be seen in figures III.15 to III.19, which have been computed for the special case where effective transmitting and receiving antenna heights are equal."

The effect shown in these charts is that s (or for smooth earth, s_s or ss), has a minor effect for $0.7 \leq s \leq 1$. There is a greater effect as s becomes lower in value, i.e. as the asymmetry increases. The above action limits the maximum effect to be that obtained at an asymmetry ratio of 10 to 1, i.e. s = 0.1.

d. The value of *q* represents the ratio of the dimensionless, and absolute, (i.e. always positive), ratio of the higher terminal's effective height above ground level, (transmit or receive site antenna) to the lower terminal's effective height above ground level, divided by *ss*, the asymmetry factor over smooth earth (whose range has not yet been limited, and has been set to always be a positive value). Here, the range of *q* is limited, so that the value of *q* can be no less than 0.1, and no more than 10.0.

Line 315: q=mymin(mymax(0.1,q),10.0);

e. *z0*, the height of the crossover point, is calculated. The Algorithm states:

"There then follows that the height of the crossover point is:

$$z0 = (s_s * d * \theta')/(1 + s_s)^2 \qquad \text{[Alg. 4.66]"}$$

where

s_s is the asymmetry factor for smooth earth, = *ss*
d is the total path distance, in kilometers
θ' is theta prime, the angular distance; = *th*

A form of this equation is also found in [TN101 9.3b], where units are in kilometers.

However, this leaves out a lot of explanation. From TN101 Section 9.2, where *z0* is referred to as h_o, it is defined as the height of the crossover point *as referenced to a direct line drawn between the transmit antenna and the receive antenna,* not with reference to sea level or ground level. A visual depiction of h_o is shown in Figure 6.1 on page 6 – 8. In TN101, it is utilized in calculating H_o, the "frequency gain function".

Here, the code calculates *z0* to be equal to: (d-ad)*(d+ad)*th*0.25/d. How did the Irregular Terrain Model code writers get to this equation for *z0,* starting from [TN 101 9.3b] and [Alg. 4.66]?

First, we re-order Alg. 4.66;

$$z0 = (s_s * d * \theta')/(1 + s_s)^2 = (\theta' * d * s_s) * 1/(1 + s_s)^2 \text{ [TN 101 9.3b] and [Alg. 4.66]}$$

Then multiply both sides by $(1 + s_s)$:

$$z0 * (1 + s_s) = \theta' * d * s_s * [1/(1 + s_s)]$$

and then substitute the equation for *ss,* a.k.a. s_s the "asymmetry factor" over smooth earth, *ss* = (*d-ad*)/(*d+ad*), derived and used in Step 7 (a.) above, for the three s_s terms:

$$z0 * (1 + (d\text{-}ad)/(d+ad)) = \theta' * d * (d\text{-}ad)/(d+ad) * [1/(1 + (d\text{-}ad)/(d+ad))]$$

multiplying out the terms in the left hand side of the equation, and reordering the terms in the right hand side of the equation:

$$(z0 + z0 * (d\text{-}ad)/(d+ad)) = \theta' * d * [(d\text{-}ad)/(d+ad)] * [1/(1 + (d\text{-}ad)/(d+ad))]$$

recombining the numerator in the right hand side of the equation:

$$(z0 + z0 * (d\text{-}ad)/(d+ad)) = \theta' * d * (d\text{-}ad)/[(d+ad) * (1 + (d\text{-}ad)/(d+ad))]$$

multiplying out the terms in the numerator in the right hand side of the equation:

$$(z0 + z0 * (d\text{-}ad)/(d+ad)) = \theta' * d * (d\text{-}ad)/[(d+ad) + (d\text{-}ad)*(d+ad)/(d+ad))]$$

the term $(d+ad)/(d+ad)$ in the right hand side denominator equals 1 (cancels out), and by adding up the terms in the right hand side denominator, we get:

$$(z0 + z0 * (d\text{-}ad)/(d+ad)) = \theta'* d* (d\text{-}ad)/(d + d + ad - ad) = \theta'* (d\text{-}ad)* d /2d$$

since $d/d = 1$, the d /2d term in the right hand side denominator equals ½, so:

$$(z0 + z0 * (d\text{-}ad)/(d+ad)) = \theta'* (d\text{-}ad)* (1/2)$$

by multiplying both sides of the equation by the term $(d+ad)$, we get:

$$z0* (d+ad) + z0 * (d\text{-}ad) * [(d+ad)/(d+ad)] = \theta'* (d\text{-}ad)* (d+ad) /2$$

the term $(d+ad) /(d+ad)$ equals 1; also, by multiplying out the terms on the left side of the equation, and adding up, we get:

$$z0* (d)+ z0*(ad) + z0 * (d) - z0*(ad) = 2 * d * z0 = \theta'* (d\text{-}ad)* (d+ad) /2$$

by dividing both sides of the equation by 2 * d, we get:

$$z0 (2d/2d) = \theta'* (d\text{-}ad)* (d+ad) /(2 * 2 * d) = \theta'* (d\text{-}ad)* (d+ad) /(4 * d)$$

the term (2d/2d) = 1, and ¼ = 0.25, so the result is:

$z0$ = th* $(d\text{-}ad)$* $(d+ad)$ *(0.25)/d, the equation used in the ITM code.

where:

th represents the angular distance, θ', (theta prime).

Line 316: z0=(d-ad)*(d+ad)*th*0.25/d;

f. the working variable *temp* is set to be equal to *z*0 divided by 8,000 unless the results equal or exceed 1.7; in which case the value of *temp* is limited to 1.7 (limiting the value of *temp* where *z0* exceeds a ceiling of 13,600 feet).

Line 319: temp=mymin(1.7,z0/8.0e3);

g. the value of temp is then set to the value of temp set in step 7 (f.), multiplied to the sixth power, i.e. $temp = (temp)^6$. Now, *temp* represents $(z0/Z_1)^6$, a component of the equation for η_s, where Z_1 = 8,000, except that *temp* is limited to a maximum value of $(1.7)^6$ = 24.138.

Line 320: temp=temp*temp*temp*temp*temp*temp;

h. the value of temp is then used to calculate the value of et to be equal to: (etq*exp(-temp)+1.0)*z0/1.7556e3);

where:

At line 293, in step 3(a.), if d = 0, *etq* was calculated to be equal to [(5.67e–6∗*prop.ens*–2.32e–3)**prop.ens*+0.031].

The full equation we are working toward is:

$$\eta_s = (z_0 / Z_0) * [1 + (0.0.031 - N_s*2.32*10^{-3} + N_s^2 *5.67*10^{-6})e^{-(z0/Z1)\wedge 6}]$$

[Alg. 4.67]

where

η_s is the scatter efficiency factor

Z_0 = 1.756 km or 1756 meters

and now where:

temp represents the value of $(z0/Z_1)^6$, limited to a maximum value of 24.138.

etq is equal to the term: $(0.0.031 - N_s*2.32*10^{-3} + N_s^2 *5.67*10^{-6})$.

This shortens the equation for η_s to be: $\eta_s = (z_0 / Z_0)*[1 + (etq)e^{-(temp)}]$.

Replacing Z_0 with 1756 meters and reordering, allows us to clearly see that *et* has been set to be the calculated value of η_s, the scatter efficiency factor:

$$\eta_s = [(etq)e^{(-temp)} + 1]*(z_0 /1.756e3)$$

Where *temp* has been limited to a maximum value of 24.138.

Line 321: et=(etq*exp(–temp)+1.0)*z0/1.7556e3)

i. The subroutine ***mymax*** is called to set *ett* to be equal to *et* unless *et* is equal to 1 or less; then the value is set to be equal to a minimum value of 1.0.

Line 323: ett=mymax(et,1.0);

From the Algorithm:

"The computation of H_0 then proceeds according to the rules in Section 9.3 and Figure 9.3 of Tech Note 101.

The model requires these results at the two distances $s = d_5$, d_6 described above. One further precaution is taken to prevent anomalous results. If, at d_5, calculations show that H_0 will exceed 15 dB, they are replaced by the value it has at d_6. This helps keep the scatter mode slope within reasonable bounds."

j. here, the subroutine ***h0f*** is called twice; the first time with inputs (r1,ett), and the second time with inputs (r2,ett),

where:

r1 here is defined as twice the angular distance th, times the effective height of the transmitter site in meters, times the wave number in units of 1/meters.

r2 here is defined as twice the angular distance th, times the effective height of the receive antenna in meters, times the wave number in units of 1/meters.

ett is the value of η_s, the scatter efficiency factor, limited to a maximum value of 1.0.

The subroutine ***h0f*** performs a function equal to that stated in TN101 in Section 9.2, on page 9-4, where it states:

"For η_s greater than or equal to 1; Read $H_o(r1)$ and $H_o(r2)$ from figure 9.3; then H_o is

$$H_o = [H_o(r1) + H_o(r2)]/2 + \Delta H_o \qquad \text{[TN101 9.5]}$$

where

$$\Delta H_o = 6 * (0.6 - \log \eta_s) \log s \log q$$

$$s = \alpha_o / \beta_o \quad q = r_2 /(s * r_1)$$ "

If $\eta_s > 5$, the value of H_o for $\eta_s = 5$ is used. The correction term ΔH_o is zero for $\eta_s = 4$, s = 1, or q = 1 and reaches a maximum value, $\Delta H_o = 3.6$ db, for highly asymmetrical paths when $\eta_s = 1$. The value of ΔH_o may be computed as shown."

Since we cannot use the table, the approximation used here, and in subroutine ***h0f***, is described in the Algorithm, section 6, starting with equation [Alg. 6.10], and continuing through [Alg. 6.14], where it states:

"The frequency gain function may be written as

$$H_o = [H_{oo}(r_1, r_2, \eta_s)] + \Delta H_o \qquad \text{[Alg. 6.10]}$$

where

$$\Delta H_o = 6 * (0.6 - \log \eta_s) \log s_s * \log r_2 /(s_s * r_1) \qquad \text{[Alg. 6.11]}$$

and where H_{oo} is obtained by linear interpolation between its values when η_s is an integer.

For $\eta_s = 1,\ldots,5$ we set

$H_{oo}(r_1, r_2, j)] = ½ [H_{01}(r_1, j) + H_{01}(r_2, j)]$ [Alg. 6.12]

With $H_{01}(r_1, j)$ equal to:

$$
\begin{array}{ll}
10 \log (1 + 24r^{-2} + 25r^{-4}) & j = 1 \\
10 \log (1 + 45r^{-2} + 80r^{-4}) & j = 2 \\
10 \log (1 + 68r^{-2} + 177r^{-4}) & j = 3 \\
10 \log (1 + 80r^{-2} + 395r^{-4}) & j = 4 \\
10 \log (1 + 105r^{-2} + 705r^{-4}) & j = 5
\end{array}
$$

[Alg. 6.13]

For $\eta_s > 5$ we use the value for $\eta_s = 5$, and for $\eta_s = 0$ we suppose

$H_{oo}(r_1, r_2, 0) = 10*\log[(1+(2)^{1/2}/ r_1)^2 *(1+(2)^{1/2}/ r_2)^2 *(r_1+r_2)/(r_1+r_2+2*(2)^{1/2}$ [Alg. 6.14]

In all of this, we truncate the values of s_s and $q = r_2 /(s_s*r_1)$ at 0.1 and 10."

The equation given for ΔH_o, in the Algorithm,

$\Delta H_o = 6 * (0.6 – \log \eta_s) \log s_s * \log r_2 /(s_s * r_1)$ [Alg. 6.11]

is the same as the equation for found in TN101:

$\Delta H_o = 6 * (0.6 – \log \eta_s) \log s \log q$ [TN101 9.5]

where $q = r_2 /(s * r_1)$ "

The two calls return the value *hofv*, i.e. the values for $H_o(r1)$ and $H_o(r2)$; h0 is then set to be equal to the average value (½ the sum) of the results of the two calls to ***h0f:***

Line 324: h0=(h0f(r1,ett)+h0f(r2,ett))*0.5;

k. the value of ΔH_o is then calculated and added to the value of h0 using an equation for ΔH_o calibrated for meters, instead of the kilometers used in TN101; the call to subroutine ***mymin*** makes sure that the value of ΔH_o is no greater than the value of h0 obtained at line 324, limiting the maximum value of h0 to be equal to the sum of the two returns from the two calls to ***h0f***. This is in accord with TN101, section 9.2, page 9-4, where it states: "If $\Delta H_o \geq [H_o(r1) + H_o(r2)]/2$, use $H_o = [H_o(r1) + H_o(r2)]$."

Line 325: h0+=mymin(h0,(1.38-log(ett))*log(ss)*log(q)*0.49);

l. The subroutine ***FORTRAN_DIM*** is called with inputs (h0, 0.0); the subroutine returns the value of (h0 – 0.0), or h0, if h0 is greater than

0.0; if 0.0 is greater than h0, the subroutine returns zero. Here, this archaic subroutine could be replaced with ***mymax***.

This is in accord with TN101, section 9.2, page 9-4, where it states: "If ΔH_o would make H_o negative, use $H_o = 0$."

Line 326: h0=FORTRAN_DIM(h0,0.0);

8. A second **if** statement is nested in the **else** statement at this point. If *et*, which holds the value of η_s, the scatter efficiency factor, is less than 1, then:

Line 328: if (et<1.0)
{

a. Here we reuse the working variable *temp*. The value of *temp* is reset to be equal to:

((1.0+1.4142/r1)*(1.0+1.4142/r2));

Line 332: temp=((1.0+1.4142/r1)*(1.0+1.4142/r2));

b. The value of h0 is set to be equal to:

h0 = et*h0+(1.0-et)*4.343*log((temp*temp)*(r1+r2)/(r1+r2+2.8284)).

Step 8(b.), incorporates an interpolation statement, referring to the statement in the Algorithm that: "H_{oo} is obtained by linear interpolation between its values when η_s is an integer." If et, a.k.a. $\eta_s < 1$, then H_{oo} for ($\eta s = 1$) has been calculated at line 324 and the value of ΔHo for ($\eta s = 1$) is added at line 325, to make h0 equal to the value of H_o for ($\eta s = 1$).
At line 326, h0 is set to be the maximum of h0 as calculated on line 325, or zero. So the term (et*h0) is the value of h0 (for $\eta_s = 1$) multiplied by η_s, representing the portion of h0 for ($0 < \eta s < 1$) interpolated from the value of h0 for $\eta_s = 1$. The term (1- et)*4.343*log((temp*temp)*(r1+r2)/(r1+r2+2.8284)) is the value of h0 (for $\eta_s = 0$) multiplied by (1-η_s), representing the portion of h0 for ($0 < \eta s < 1$) interpolated from the value of h0 for $\eta_s = 0$.

The term (4.343*log((temp*temp)*(r1+r2)/(r1+r2+2.8284)) represents the value of h0 for $\eta_s = 0$, where $H_{o(\eta s = 0)} = H_{oo}(r_1, r_2, 0) + \Delta H_{o(\eta s = 0)}$. TN101, section 9.2, states: "The case $\eta_s = 0$ corresponds to the assumption of a constant atmospheric refractive index." The Algorithm, section 6, states:

"for $\eta_s = 0$ we suppose

$$H_{oo}(r_1, r_2, 0) = 10*\log[(1+(2)^{1/2}/ r_1)^2 *(1+(2)^{1/2}/ r_2)^2 *(r_1+r_2)/(r_1+r_2+2*(2)^{1/2})$$

[Alg. 6.14]"

Since $(2)^{1/2} = 1.4142$, the value of *temp* set in step 8 (a.) is equal to the term $(1+(2)^{1/2}/ r_1)^2 *(1+(2)^{1/2}/ r_2)^2$ in [Alg 6.14], simplifying $H_{oo}(r_1, r_2, 0)$ to be:

$$H_{oo}(r_1, r_2, 0) = 10*\log[(temp)*(temp)*(r_1+r_2)/(r_1+r_2+2.8284)]$$

Which we can easily recognize as being a part of the equation for h0 in step 8 (b.) above. In step 8(a.), by removing the interpolation terms, we can derive that h0 for $\eta_s = 0$, as used in the code, is:

$$h0 \text{ for } (\eta_s = 0)= 4.343*\log((temp*temp)*(r1+r2)/(r1+r2+2.8284)).$$

Since h0 is equal to: $H_o = [H_{oo}(r_1, r_2, \eta_s)] + \Delta H_o$ [Alg. 6.10]

For $\eta_s = 0$, $[H_{oo}(r_1, r_2, 0)]$ is equal to:

$$H_{oo}(r_1, r_2, 0)=10*\log[(1+(2)^{1/2}/ r_1)^2 *(1+(2)^{1/2}/ r_2)^2 *(r_1+r_2)/(r_1+r_2+2*(2)^{1/2})]$$

[Alg. 6.14]"

replacing the term $[1+(2)^{1/2}/ r_1)^2 *(1+(2)^{1/2}/ r_2)^2]$ in [Alg. 6.14] with (*temp***temp*), we get:

$$H_{oo}(r_1, r_2, 0)=10*\log[(temp*temp)*(r_1+r_2)/(r_1+r_2+2.8284)].$$

This is calculated as per [Alg 6.14], except that the constant multiplier value 4.343 replaces the constant multiplier value 10, as it did for the equations based on [Alg. 6.13] used in the subroutine ***h0f***, previously called by ***ascat***. Why? I thought the answer was the unusual units of measure of *th*, used in calculating r_1 and r_2 , and I still hold that opinion. But a single change of a constant value, from 10 to 4.343, will not do the job of compensating for the different units properly; as the equations use values of $(r_1)^2$, $(r_2)^2$, $(r_1)^4$ and $(r_2)^4$ multiplied by varying constants.

MAJOR PROBLEM NOTE: ***Therefore, since r_1 and r_2 were calculated with a value of th in the wrong units, it appears to this author that the subroutine will produce errant and erratic results; the replacement of the constant value 10 by 4.343 may have occurred in order to produce results that were somewhat close to the empirical results from the field measurements. The author finds no other solid mathematical basis for the change from 10 to 4.343 in the equations in the code.***

How can this be corrected? By using the equation:

$$\textit{arctan (V/H)} = \theta \textit{ in radians (rads)}$$

To convert the unit value of th to radians prior to its use in calculating r_1 and r_2, at line 305 of this subroutine; and replacing the constant 4.343 with the

correct constant, 10, stated in [Alg. 6.13 and 6.14], in subroutines hof and ascat.

Using a different approach, we try reverse engineering. Solving [Alg. 6.10] for ΔH_o, we derive that:

$$\Delta H_o = H_o - [H_{oo}(r_1, r_2, \eta_s)]$$

therefore $\Delta H_{o(\eta s = 0)} = H_{o(\eta s = 0)} - [H_{oo}(r_1, r_2, 0)]$, and;

$$\Delta H_{o(\eta s = 0)} = [4.343*\log((temp*temp)*(r1+r2)/(r1+r2+2.8284))] - 10*\log[(temp*temp)*(r_1+r_2)/(r_1+r_2+2.8284)]$$

which shortens to become:

$$\Delta H_{o(\eta s = 0)} = (4.343 - 10)*[\log((temp*temp)*(r1+r2)/(r1+r2+2.8284))]$$

and adds up to:

$$\Delta H_{o(\eta s = 0)} = (-5.657)*[\log((temp*temp)*(r1+r2)/(r1+r2+2.8284))]$$

and in the code, ΔH_o is added to $H_{oo}(r_1, r_2, 0)$ by:

```
h0+=mymin(h0,(1.38-log(ett))*log(ss)*log(q)*0.49);
```

Since we cannot use the table, the approximation used here, and in subroutine ***h0f***, is described in the Algorithm, section 6, starting with equation [Alg. 6.10], and continuing through [Alg. 6.14], where it states:

"The frequency gain function may be written as

$$H_o = [H_{oo}(r_1, r_2, \eta_s)] + \Delta H_o \qquad \text{[Alg. 6.10]}$$

where

$$\Delta H_o = 6 * (0.6 - \log \eta_s) \log s_s * \log r_2 /(s_s * r_1) \qquad \text{[Alg. 6.11]}$$

and where H_{oo}
is obtained by linear interpolation between its values when η_s is an integer.
For $\eta_s = 1,\ldots,5$ we set

$$H_{oo}(r_1, r_2, j)] = \tfrac{1}{2} [H_{01}(r_1, j) + H_{01}(r_2, j)] \qquad \text{[Alg. 6.12]}$$

With $H_{01}(r_1, j)$ equal to:

$$10 \log (1 + 24r^{-2} + 25r^{-4}) \quad j = 1 \qquad \text{[Alg. 6.13]}$$
$$10 \log (1 + 45r^{-2} + 80r^{-4}) \quad j = 2$$

$10 \log (1 + 68r^{-2} + 177r^{-4})$ j = 3
$10 \log (1 + 80r^{-2} + 395r^{-4})$ j = 4
$10 \log (1 + 105r^{-2} + 705r^{-4})$ j = 5

For $\eta_s > 5$ we use the value for $\eta_s = 5$, and for $\eta_s = 0$ we suppose

$H_{oo}(r_1, r_2, 0) = 10*\log[(1+(2)^{1/2}/ r_1)^2 *(1+(2)^{1/2}/ r_2)^2 *(r_1+r_2)/(r_1+r_2+2*(2)^{1/2}$ [Alg. 6.14]

In all of this, we truncate the values of s_s and $q = r_2 /(s_s*r_1)$ at 0.1 and 10."

The equation given for ΔH_o, in the Algorithm,

$\Delta H_o = 6 * (0.6 - \log \eta_s) \log s_s * \log r_2 /(s_s * r_1)$ [Alg. 6.11]

is the same as the equation for found in TN101:

$\Delta H_o = 6 * (0.6 - \log \eta_s) \log s \log q$ [TN101 9.5]

where $q = r_2 /(s * r_1)$ "

This is used in lieu of the procedure from TN101 section 9.2(b), where it states:
"For η_s less than 1:
First, obtain H_o for $\eta_s = 1$, as described above, then read H_o for $\eta_s = 0$ from figure 9.5. Figure 9.5b shows H_o ($\eta_s = 0$) for the special case of equal antenna heights. The desired value is found by interpolation:
$H_o(\eta_s < 1) = H_o(\eta_s = 0) + \eta_s [H_o(\eta_s = 1) - (H_o(\eta_s = 0)]$."
Since the program cannot read the value from figure 9.5;

$H_o = [H_o(r1) + H_o(r2)]/2 + \Delta H_o$ [TN101 9.5]

Where:

Line 333: h0=et*h0+(1.0-et)*4.343*log((temp*temp)*(r1+r2)/(r1+r2+2.8284));
}

The above steps 8 (a.) and (b.) are used to approximate the results obtained from the procedure found in TN101 section 9.2(b).

This is calculated as per [Alg 6.13], except that the constant multiplier value 4.343 replaces the constant multiplier value 10, a fudge factor adjustment found in eleven locations in the code, and discussed in the Chapter on Aknfe. A single change of a constant value, from 10 to 4.343, will not do the job properly; as the formula uses values of $(r_1)^2$, $(r_2)^2$, $(r_1)^4$ and $(r_2)^4$ multiplied by varying constants.

MAJOR PROBLEM NOTE: ***Therefore, since r_1 and r_2 were calculated with a value of th in the wrong units, it appears to this author that the subroutine will produce errant***

and erratic results; the replacement of the constant value 10 by 4.343 appears to be a "fudge factor" added in order to produce results that were somewhat close to the empirical results from the field measurements. The author finds no other solid mathematical basis for the change from 10 to 4.343 in the equations in the code.

9. A third **if** statement is nested in the **else** statement at this point. If both: h0 is greater than 15.0, and h0s is greater than or equal to 0.0, then h0 is set to be equal to h0s;

Line 336:
```
if (h0>15.0 && h0s>=0.0)
        h0 = h0s;
```

10. The second **else** statement then ends, but the first **else** statement is still active, and continues;

    a. *h0s* is set to be equal to *h0*. Here *h0s*, the H_o value for smooth earth, is preset to be equal to the value of *h0*, unless h0 was >15 and h0s > 0, where both would have been set to the value for h0s.
    b. *th* is set to be equal to: *propa.tha* + *d* * *prop.gme*;

    where:

propa.tha	is the total bending angle, set in ***lrprop***;
d	is the path distance
prop.gme;	is the effective earth's curvature.

NOTE: HERE, *th IS RECALCULATED USING propa.tha, which may be derived from prop.the[0] and prop.the[1] in step 3(c.) of* ***lrprop.***.

    c. ascatv is set to be equal to:
    ahd(th*d)+4.343*log(47.7*prop.wn*(th*th*th*th))
    – 0.1*(prop.ens-301.0)*exp(-th*d/40e3)+h0;

    The subroutine ***ahd*** is called with input equal to: (th*d). The subroutine ***ahd*** returns the value of F_0 (D), a.k.a. $F(\theta_d, N_s = 301)$, used below in the equation for $F(\theta_d, N_s)$.

From Section 4.3.1 , "**The function** A_{scat},"

"$A_{scat}(s) = 10 \log(k * H * \theta^4) + F(\theta_s, N_s) + H_o$ [Alg. 4.63]

where

$F(\theta_s, N_s)$	is the function shown in Figure 9.1 of TN101
H_o	is the "frequency gain function"
H	is 47.7 meters."

also:

k,	the wave number, is the value stored in *prop.wn;*

th is the angular distance, recalculated in step 10(b.) above;

Figure 9.1 of TN101 is a table; so its function is approximated by the equation listed after III.48 in Annex III, Section 5 – Forward Scatter, where it states:

The function $F(\theta_d)$ may be obtained for any value of N_s, by modifying the value computed for $N_s = 301$:

$$F(\theta_d,\ N_s) = F(\theta_d,\ N_s = 301) - 0.1*(N_s - 301.0)*\exp(-\theta*d/40)$$

Which, with the constant 40 converted to 40,000 to adjust for a change from kilometers to meters as the unit value of *d*, is the same as:

$$F(\theta_s,\ N_s) = -0.1*(\text{prop.ens}-301.0)*\exp(-\text{th}*d/40e3)$$

where

prop.ens is the earth's surface refractivity, a.k.a. N_s
th is the angular distance, θ, recalculated in step 10(b.) above;
d is the path distance

NOTE: 4.343 is being substituted for 10 again, as discussed above.

d. The first **else** statement then ends its run.

```
Line 340:    h0s=h0;
Line 341:    th=propa.tha+d*prop.gme;
Line 343:    ascatv=ahd(th*d)+4.343*log(47.7*prop.wn*(th*th*th*th))-0.1*(prop.ens-
             301.0)*exp(-th*d/40e3)+h0;
        }
```

11. The subroutine then ends by returning the value of *ascatv*, the "scatter attenuation" at a distance *d*.

```
Line 346: return ascatv;
```

Chapter 18: Ahd

Approximate tHeta D function for scatter fields; subroutine: ***ahd***.

Note: Used with both point-to-point mode and area mode. Called by ***ascat.***

From ITMD Section 26:

This is the *F(θd)* function for scatter fields.

As defined in the Algorithm, Section 6, "Addenda – numerical approximations." This section starts by mentioning:

> "Part of the algorithm for the ITM consists in approximations for the standard functions that have been used. In these approximations, computational simplicity has often taken greater priority than accuracy."

The Algorithm later states:

> "we have the two functions, *F(θd)* and H_0 , used for tropospheric scatter. First,
>
> $$F(D, N_s) = F_0(D) - 0.1\,(N_s - 301)e^{-D/Do} \qquad (6.8)$$
>
> where
>
> $$D_0 = 40 \text{ km}$$
>
> And (when D_0 is given in meters)
>
> $$F_0(D) = 133.4 + 0.332 * 10^{-3} * D - 10 * \log D \quad \text{for } 0 < D \leq 10 \text{ km, or}$$
>
> $$F_0(D) = 104.6 + 0.212 * 10^{-3} * D - 2.5 * \log D \quad \text{for } 10 < D \leq 70 \text{ km, or}$$
>
> $$F_0(D) = 71.8 + 0.157 * 10^{-3} * D + 5 * \log D \quad \text{otherwise "} \qquad (6.9)$$

This can also be found in Annex III, Section III.5 "Forward Scatter", of Tech Note 101; from equations [TN101 III.46, 47, and 48] on page III-24.

Call inputs for subroutine ***ahd***:

Td *D*, distance in meters

Declares private, or local, arguments:
i;

a[3]={ 133.4, 104.6, 71.8};
b[3]={ 0.332e–3, 0.212e–3, 0.157e–3};

NOTE: The constants for c below do not match the constants in Alg. 6.9. They have been modified by the multiplication of the original constants by .4343; a part of the "fudge factor" applied in 11 locations.

c*[3]*= { –4.343, – 1.086, 2.171 };

These constants should be:

{ -10, - 2.5, and +5 }, as per [Alg. 6.9]

In this subroutine:

204. Initiates an **if** statement. If *td* is less than or equal to 10,000 meters, then the value of *i* is set to be equal to zero.

```
Line 207:    if (td<=10e3)
                 i=0;
```

205. An ***else if*** statement follows; if *td*, at line 207, was more than 10,000 meters, and less than or equal to 70,000 meters, then the value of *i* is set to be equal to 1.

```
Line 178:    else if (td<=70e3)
                 i=1;
```

206. An ***else*** statement follows; so if *td*, at line 207, was more than 70,000 meters, then:

i is set to be equal to 2.

```
Line 213:    else
                 i=2;
```

207. The subroutine then calculates and returns the value of F_0 (D), using the appropriate formula from [Alg. 6.9]:

```
Line 216:    return a[i] + b[i] *td +c[i] * log(td);
```

Chapter 19: H0f

H0 Frequency gain function for scatter fields; subroutine: ***h0f***.

Note: Used with both point-to-point mode and area mode. Called by ***ascat.***

From ITMD Section 25:

This is the H_{01} function for scatter fields as defined in the Algorithm, Section 6, "Addenda – numerical approximations."

Background:

From the Algorithm, Section 6, "Addenda – numerical approximations."

This section starts by mentioning:

> "Part of the algorithm for the ITM consists in approximations for the standard functions that have been used. In these approximations, computational simplicity has often taken greater priority than accuracy."

The Algorithm later states, following equation (6.9):

> "The frequency gain function may be written as
>
> $$H_0 = H_{00}(r_1, r_2, \eta_s,) + \Delta H_0 \qquad (6.10)$$
>
> where
>
> $$\Delta H_0 = 6\,(0.6 - \log n_s) * \log s_s * \log r_2/s_s\, r_1 \qquad (6.11)$$
>
> and where H_{00} is obtained by linear interpolation between its values when n_s is an integer. For $\eta_s = 1, \ldots, 5$ we set
>
> $$H_{00}(r_1, r_2, j) = \tfrac{1}{2}[\, H_{01}(r_1, j) + H_{01}(r_2, j) \qquad (6.12)$$
>
> with
>
> $$\begin{aligned} H_{01}(r, j) = \; & 10 \log (1 + 24r^{-2} + 25\, r^{-4}) \;\text{ for } j = 1, \\ & 10 \log (1 + 45r^{-2} + 80\, r^{-4}) \;\text{ for } j = 2, \\ & 10 \log (1 + 68r^{-2} + 177\, r^{-4}) \;\text{ for } j = 3, \\ & 10 \log (1 + 80r^{-2} + 395\, r^{-4}) \;\text{ for } j = 4, \\ & 10 \log (1 + 105r^{-2} + 705\, r^{-4}) \;\text{ for } j = 5. \end{aligned} \qquad (6.13)$$
>
> For $\eta_s > 5$, we use the value for $\eta_s = 5$, and for $\eta_s = 0$ we suppose

$$H_{00}(r_1, r_2, 0) = 10 \log [(1 + (2)^{1/2}/ r_1)^2 (1 + (2)^{1/2}/ r_2)^2 * (r_1 + r_2)/(r_1 + r_2 + 2(2)^{1/2})] \quad (6.14)$$

In all of this, we truncate the values of s_s and $q = r_2 / s_s r_1$ at 0.1 and 10."

Call inputs for subroutine ***h0f***:

r twice the angular distance th, (measured in a ratio of meters, vertical to meters, horizontal, not radians as in TN101 Section 9.2) times the effective height of the terminal antenna (r1 = transmit, r2 = receive) in meters, divided by a wavelength at the frequency selected, in meters. Units (for itm.cpp) cancel out to be dimensionless, not radians as in TN101 Section 9.2.

et the value of η_s, the "scatter efficiency factor"

Declares private, or local, arguments:

double *a[5]*={25.0, 80.0, 177.0, 395.0, 705.0};
double *b[5]*={24.0, 45.0, 68.0, 80.0, 105.0};
double *q*,
double *x*;
double *h0fv*,
double *temp*;
int *it*;

This subroutine:

208. Presets *it* to be equal to: the integer value of input value *et,* (which is equal to the value of η_s, the scatter efficiency factor). This follows from the statement in the Algorithm, Section 6, following equation (6.11), which states: "and where H_{00} is obtained by linear interpolation between its values when η_s is an integer.

Line 170: it=(int)et;

209. Initiates an **if** statement. If *it* is less than or equal to zero, then:
   a. the value of *it* is reset to be equal to 1.
   b. *q* is set to be equal to 0.0.

Line 172:
```
if (it<=0)
{
it=1;
q=0.0;
}
```

210. An ***else if*** statement follows; if *it*, at line 172, was equal to or greater than 5, then:
   a. the value of *it* is reset to be equal to 5.

b. *q* is set to be equal to 0.0.

Line 178:
```
else if (it>=5)
{
        it=5;
        q=0.0;
}
```

Steps 2 and 3 prepare for the use of the procedure associated with equations [Alg. 6.12 and Alg 6.13] stated in the background section, above.

211. An ***else*** statement follows; so if *it*, at line 172, was more than 0, and less than 5, then:

q is set to be equal to the value of *et*, less the value of *it*.

Line 184:
```
else
q=et-it;
```

212. The value of *temp* is set to be equal to (1/*r*), and then the value of *x* is set to be equal to the value of $temp^2$. The value of *x* therefore becomes equal to $(1/r)^2$.

Line 189:
```
temp=1.0/r;
x=temp*temp;
```

The value of *h0fv* is set to be equal to: 4.343 * log((*a*[*it*-1]**x*+*b*[*it*-1])**x*+1.0); this calculates the $H_{01}(r, j) = 10 \log (1 + (b)r^{-2} + (a) r^{-4})$ for j = *it* -1, as per [Alg 6.13], except that the constant multiplier value 4.343 replaces the constant multiplier value 10.

Why is the 10 replaced with 4.343? The obvious expectation is that it is due to the nonstandard units of angular measure used for *th*, which was used to calculate the values of r1 and r2 in the ***ascat*** subroutine, which later called this subroutine. The angular distance, th, a.k.a. θ, or theta) is calculated and defined as (vertical distance in meters/horizontal distance in meters) in the code, instead of in radians.

Tech Note 101 states that θ is in units of radians. George Hufford, the author of the Algorithm, does not note the use of any such conversion factor in the Algorithm equations 6.13 and 6.14, which use the constant multiplier value 10, that is correct for r1 and r2 calculated with θ defined in units of radians.

In the equation [Alg. 4.62], in Section 4.3.1, "The Function A_{scat}." the angular distance θ is still being specified in radians, as becomes clear in Section 6, equations [Alg 6.13 and 6.14].

How do we convert θ, a.k.a. *th*, to radians? There are 2π radians in a full cycle, or 360°. A radian is defined as the angle subtended at the center of a circle by an arc of circumference that is equal in length to the radius of the circle. Draw this construct on a circle, with one radii of length r on the horizontal plane, and a distance of r on the circumference between the two radii. Now draw a vertical line from the point where the non-horizontal radii touches the circumference of the circle, to a point perpendicular to the horizontal radii, forming a right triangle. The radius then becomes the hypotenuse of a right triangle with an angle, subtended at the center of the circle, between the two radii, of one radian, or 57.2958 degrees. The length of the vertical line is then equal to the sine function of the angle θ, which is equal to the ratio of the length of the vertical line to the length of the hypotenuse of the triangle, which is equal to r. So we can now obtain the length of the vertical line by multiplying sin θ by the hypotenuse length, r. This results in the equation:

$$V = (\sin \theta) * r$$

The length of the horizontal line is then equal to the cosine function of the angle θ, which is equal to the ratio of the length horizontal line of the triangle to the length of the hypotenuse of the triangle, which is equal to r. So we can obtain the length of the horizontal line by multiplying cos θ by the hypotenuse length, r. This results in the equation:

$$H = (\cos \theta) * r$$

We can now obtain the ratio of the vertical length to the horizontal length by dividing the equation for Y by the equation for X. and canceling out the "r" terms:

$$V/H = [(\sin \theta) * r] / [(\cos \theta) * r] = (\sin \theta)/(\cos \theta)$$

In trigonometry, by definition of the tangent function, tan x = (sin x) / (cos x), so the equation becomes:

$$V/H = (\tan \theta) \text{ in radians}$$

This can be used to convert from the angle in radians or degrees, to the ratio used for θ in the code, but we also need to know how to convert from the vertical-distance-to-horizontal-distance ratio (V/H ratio) used for *th*, to radians, in case we later run into a formula that cannot handle the V/H ratio. For this we use the arctan subroutine function:

$$\arctan (V/H) = \theta \text{ in radians (rads)}$$

Here, we will attempt to utilize these conversion ratios.

Line 192: h0fv=4.343*log((a[it-1]*x+b[it-1])*x+1.0);

213. Initiates an **if** statement. If q , which holds the difference value between (double) *et* and (integer) *it*, is not equal to zero (which would indicate *et* = *it*), then: *h0fv* is set to be equal to:

$$(1.0-q) * h0fv+q*4.343*\log((a[it]*x+b[it])*x+1.0)$$

Here the value of *h0fv* is interpolated. The H_{01} value for *it* is calculated to be: H_{01}(for *it*) = 4.343 * log((*a*[*it*]**x*+*b*[*it*])**x*+1.0). The term (1 – *q*) interpolates the H_{01} value calculated for *it* , the integer value of η_s, the scatter efficiency factor, to approximate the H_{01} value for *et* = η_s.

This is calculated as per [Alg 6.13], except that the constant multiplier value 10 has been multiplied by a fudge factor of .4343. A discussion of this fudge factor, which appears twice in this subroutine, is found in the Chapter on Aknfe. It would be best if this fudge factor can be eliminated after making the other error corrections needed. A single change of a constant value, by multiplying 10 by .4343, will not do the job properly; as the equations use values of $(r_1)^2$, $(r_2)^2$, $(r_1)^4$ and $(r_2)^4$ multiplied by varying constants.

MAJOR PROBLEM NOTE: ***Therefore, since r_1 and r_2 were calculated with a value of th in the wrong units, it appears to this author that the subroutine will produce errant and erratic results; the replacement of the constant value 10 by 4.343 may have occurred in order to produce results that were somewhat close to the empirical results from the field measurements. The author finds no other solid mathematical basis for the change from 10 to 4.343 at several locations in the code.***

How can this be corrected? By using the equation:

arctan (V/H) = θ in radians (rads)

To convert the unit value of th to radians at the end of hzns, prior to its use in calculating r_1 and r_2; and replacing the constant 4.343 with the correct constant, 10, stated in [Alg. 6.13 and 6.14], in the subroutines.

Line 194: if (q!=0.0)

```
h0fv=(1.0-q)*h0fv+q*4.343*log((a[it]*x+b[it])*x+1.0);
```

Here we find the second instance of the application of the fudge factor .4343 in this subroutine.

214. Subroutine ***h0f*** ends by returning the H_{01} (*r*, *et*) function output value stored in *h0fv:*

Line 197: return h0fv;

Chapter 20: Avar

Longley-Rice Analysis of Variants subroutine, ***avar***.

Note: This subroutine is called by subroutine ***point-to-point*** or ***area***. In version 7.0 of the ITMDLL.cpp, released in June of 2007, there are two alternative subroutines, ***point_to_pointDH*** and ***point_to_pointMDH***, that provide improvements and also call ***avar***. Calls ***curve***.

This subroutine calculates the additional loss resulting from statistical analysis of long term variability due to time (reliability), location, and/or situational variability (confidence), depending upon the operating mode determined by the mode of variability code input. The subroutine reports out a single value of loss, *avarv,* in decibels (dB), which includes *aref*, the "reference attenuation" summed with the additional statistical loss.

Derived from the Irregular Terrain Model description by George Hufford, 2002, (ITMD), and from "A manual for ITM, "Irregular Terrain Model", a manual for the FORTRAN user, found at: http://flattop.its.bldrdoc.gov/itm/itm_man.txt, compared to the ITM.cpp prepared by J. D. McDonald and John Magliacane for compilation on unix and linux systems. "Line" numbers refer to the ITM.cpp as line numbered by Bloodshed Software's DevC++ print function. "Alg" numbers refer to the algorithm formulas in "The ITS Irregular Terrain Model, version 1.22, the Algorithm" by G. A. Hufford, 1995. "ITS67" numbers refer to the algorithm formulas in "ESSA Technical Report ERL 79-ITS 67, Prediction of Tropospheric Radio Transmission Over Irregular Terrain, A Computer Method – 1968" by A.G.Longley and P.L.Rice.

From ITMD Section: 27:

Subroutine ***"lrprop*** will stand alone to compute *aref*. To complete the story, however, one must find the quantiles of the attenuation and this is what ***avar*** will do. It, too, is a stand alone subroutine, except that it requires the output from ***lrprop***, as well as values in a 'variability parameters" common block. These latter values consist of :

A control switch *lvar,*
The standard deviation of situation variability (confidence) *sgc*,
The desired mode of variability *mdvar*, and;
The climate indicator *klim.*

Of these, *sgc* is output, and may be used to answer the inverse problem; with what confidence will a threshold signal be exceeded.

Subroutine inputs:

propv array *propv* with elements:
lvar a control switch, indicating a reset of input values is required.
sgc the standard deviation of situation variability (confidence)
mdvar found as the value of *propv.mdvar*, the mode of variability. In subroutine ***point_to_point***, *mdvar* is set to be equal to 12, a combination of modes 10 and 2. The value "10" is added for the point-to-point mode, which causes location variability to be eliminated. [However, this should not be true in version 7, subroutine ***point_to_pointMDH,*** which allows location variability to be set]. The value "2" indicates "Mobile" mode, where reliability is calculated as a combination of time and location variability. Confidence in mode 2 (or 12) is given by the situation variability.

NOTE: It is interesting to note that for TV or FM broadcast use, in point-to-point mode, the mdvar is set to Mobile mode, not Broadcast. The code for mdvar, the variability mode, which sets the mode of operation, is a tens and single digit code. The single digits represent:

0 - Single message mode. Time, location and situation variability are combined together to give a confidence level.
1 – Accidental mode. Reliability is given by time availability. Confidence is a combination of location and situation variability.
2 – Mobile mode. Reliability is a combination of time and location variability. Confidence is given by the situation variability.
3 – Broadcast mode. Reliability is given by the statement of –at least- qT of the time in qL of the locations. Confidence is given by the situation variability.

The tens code is: No tens; default to area mode: combined code is 0 to 3.
10 – For the point-to-point mode. Location variability is eliminated.
20 – For interference problems. Direct situation variability is eliminated. Note that there may be a small residual situational variability.

So the setting of mdvar equal to 12, hard-coded into the ITMDLL.cpp, means that the current version of ITMDLL.cpp is never intended to be used for broadcast unless the value of mdvar is changed in the source code and the ITMDLL is re-compiled.

Note: Therefore, for broadcast reception prediction use, the ITMDLL should be modified to allow external resetting of the mdvar; this is especially critical in light of the new optional subroutine *point_to_pointMDP*, which allows for setting the percentage value for location.

klim, or Radio_climate the radio climate code; set by the user or obtained from a preset list. Customary Default value is 5. The climate codes are:

1. Equatorial; (Africa, along the equator)
2. Continental Subtropical; (Sudan region)
3. Subtropical (a.k.a. Maritime Subtropical (West Coast of Africa);
4. Desert (Death Valley, NV; Sahara);
5. Continental Temperate (usual general U.S. default);
6. Maritime Temperate Over Land (In U. S., California to State of Washington; West Coast of Europe including U.K.),
7. Maritime Temperate, Over Sea.

Call inputs:

zzt, a.k.a. zr, qerfi (rel), ztime, time; rel; reliability; time reliability; a statistical percentage of time availability; set as .01 to .99 Usual default setting; for NTSC (analog) TV, FM broadcast and most FM analog transmissions, set to 0.50 (50% for FCC 50,50); for Digital FM IBOC sidebands, set to .90 to .98, or for television (DTV), set to 0.97 (97% for FCC 50, 97).

zzl, a.k.a., zloc, loc, location, location variability; a statistical percentage of location availability; set to 0.01 to 0.99. Internally fixed to zero in ***point_to_point***; as ***point_to_point*** calls subroutine ***avar*** with the location variable set at 0.0. In the new version 7, the optional alternative subroutine ***point_to_pointMDH*** allows this to be set by the user. Usual user default setting; 0.50 (50%) for 50% of locations.

zzc, a.k.a. zc, zconf, qerfi(conf), conf, confidence; a statistical percentage of confidence in the situation; set as .01 to .99 . In ***avar,*** the definition of confidence varies with the value of mdvar, the mode of variability; for mdvar = 0, for example, time, location, and situation variability are combined together. For point_to_point mode, mdvar = -1. See chapter on ***avar*** or itm.man. Usual default setting; 0.50 (50%) (Note: often used instead of location calculation, i.e. to approximate 50% of locations; however, ***avar*** has separate inputs for confidence and location, and point_to_point calls ***avar*** to calculate the reference attenuation with the location variable set at 0.0. See note at input loc below regarding optional subroutine ***point_to_pointMDH***.)

prop_type &prop, array prop with elements:

a. *Prop.wn* wave number, = freq. in MHz/47.7 MHz*m; units in 1/meters
b. *prop.ens* surface refractivity (refractivity of the atmosphere)
c. *prop.gme* effective earth curvature
d. *prop.zgnd* surface impedance array, consisting of:
e. *prop.zgndreal* real surface impedance (resistance component)
f. *prop.zgndimag* imaginary surface impedance (reactive component)

propv_type &propv, array propv with elements:
Outputs:

defines private, or local, arguments:

```
static   int kdv;
static   doubles:
         dexa,
         de,
         vmd,
         vs0,
         sgl,
         sgtm,
         sgtp,
         sgtd,
         tgtd,
         gm,
         gp,
         cv1,
         cv2,
         yv1,
         yv2,
         yv3,
         csm1,
         csm2,
         ysm1,
         ysm2,
         ysm3,
         csp1,
         csp2,
         ysp1,
         ysp2,
         ysp3,
         csd1,
         zd,
         cfm1,
         cfm2,
         cfm3,
         cfp1,
         cfp2,
         cfp3;
```

The following tables are on lines 863 to 885;

```
double bv1[7]={-9.67,-0.62,1.26,-9.21,-0.62,-0.39,3.15};
double bv2[7]={12.7,9.19,15.5,9.05,9.19,2.86,857.9};
double xv1[7]={144.9e3,228.9e3,262.6e3,84.1e3,228.9e3,141.7e3,2222.e3};
double xv2[7]={190.3e3,205.2e3,185.2e3,101.1e3,205.2e3,315.9e3,164.8e3};
double xv3[7]={133.8e3,143.6e3,99.8e3,98.6e3,143.6e3,167.4e3,116.3e3};
double bsm1[7]={2.13,2.66,6.11,1.98,2.68,6.86,8.51};
double bsm2[7]={159.5,7.67,6.65,13.11,7.16,10.38,169.8};
double xsm1[7]={762.2e3,100.4e3,138.2e3,139.1e3,93.7e3,187.8e3,609.8e3};
double xsm2[7]={123.6e3,172.5e3,242.2e3,132.7e3,186.8e3,169.6e3,119.9e3};
double xsm3[7]={94.5e3,136.4e3,178.6e3,193.5e3,133.5e3,108.9e3,106.6e3};
double bsp1[7]={2.11,6.87,10.08,3.68,4.75,8.58,8.43};
double bsp2[7]={102.3,15.53,9.60,159.3,8.12,13.97,8.19};
double xsp1[7]={636.9e3,138.7e3,165.3e3,464.4e3,93.2e3,216.0e3,136.2e3};
double xsp2[7]={134.8e3,143.7e3,225.7e3,93.1e3,135.9e3,152.0e3,188.5e3};
double xsp3[7]={95.6e3,98.6e3,129.7e3,94.2e3,113.4e3,122.7e3,122.9e3};
double bsd1[7]={1.224,0.801,1.380,1.000,1.224,1.518,1.518};
double bzd1[7]={1.282,2.161,1.282,20.,1.282,1.282,1.282};
double bfm1[7]={1.0,1.0,1.0,1.0,0.92,1.0,1.0};
double bfm2[7]={0.0,0.0,0.0,0.0,0.25,0.0,0.0};
double bfm3[7]={0.0,0.0,0.0,0.0,1.77,0.0,0.0};
double bfp1[7]={1.0,0.93,1.0,0.93,0.93,1.0,1.0};
double bfp2[7]={0.0,0.31,0.0,0.19,0.31,0.0,0.0};
double bfp3[7]={0.0,2.00,0.0,1.79,2.00,0.0,0.0};

static bool ws,
static bool w1;
double rt=7.8,
        rl=24.0,
        avarv,
        q,
        vs,
        zt,
        zl,
        zc;
double sgt,
        yr,
        temp1,
        temp2;
int temp_klim=propv.klim-1; i.e. is equal to one less than the radio climate code klim;
```

This subroutine:

1. An **if** statement is initiated. If the value of propa.lvar, the switch code, is greater than zero, then:

Line 891:

```
if (propv.lvar>0)
{
```

2. A **switch** statement is initiated with input of the value of *prop.lvar*. The switch statement selects the case to act on, based on the integer value read from *prop.lvar*.

Line 893:

```
switch (propv.lvar)
{
```

3. A **default** statement is initiated, with an if statement following, so if *prop.lvar* is not equal to either: 1, 2, 3, or 4, and if *propv.klim*, the radio climate variable, is outside of its normal range of 1 to 7, then:
   a. *propv.klim* is set to be equal to the normal default of 5;
   b. *temp_klim* is set to be equal to 4, or one less than *propv.klim*, and
   c. *prop.kwx,* the error indicator, is set to be the greater of *prop.kwx* or 2, indicating that impossible parameters have been replaced with default values.

Line 895:

```
default:
if (propv.klim<=0 || propv.klim>7)
{
        propv.klim=5;
        temp_klim=4;
        prop.kwx=mymax(prop.kwx,2);
}
```

4. The values of the arguments, *cv1*, *cv2*, etc., are populated from the corresponding values of table arrays *bv1*[*temp_klim*], *bv2*[*temp_klim*], etc. on lines 863 to 885, where *temp_klim* is equal to an integer value between zero and six.

Line 903:

```
cv1=bv1[temp_klim];
cv2=bv2[temp_klim];
yv1=xv1[temp_klim];
yv2=xv2[temp_klim];
yv3=xv3[temp_klim];
csm1=bsm1[temp_klim];
csm2=bsm2[temp_klim];
ysm1=xsm1[temp_klim];
ysm2=xsm2[temp_klim];
ysm3=xsm3[temp_klim];
csp1=bsp1[temp_klim];
```

```
csp2=bsp2[temp_klim];
ysp1=xsp1[temp_klim];
ysp2=xsp2[temp_klim];
ysp3=xsp3[temp_klim];
csd1=bsd1[temp_klim];
zd=bzd1[temp_klim];
cfm1=bfm1[temp_klim];
cfm2=bfm2[temp_klim];
cfm3=bfm3[temp_klim];
cfp1=bfp1[temp_klim];
cfp2=bfp2[temp_klim];
cfp3=bfp3[temp_klim];
```

5. If *lvar* was equal to 4, then case "four" applies, and:
    a. *Kdv* is set to be equal to *prop.mdvar.*
    b. *Ws* has its Boolean value set depending upon whether kdv is greater that or equal to 20; therefore, *ws* will be true for a value of *mdvar* greater than 20, indicating a interference study is in progress.

Line 927:
```
case 4:
kdv=propv.mdvar;
ws=kdv>=20;
```

6. If *ws* is true, 20 is subtracted from the value of *kdv*, leaving only a single digit value of *mdvar*; either 0, 1, 2, or 3.

Line 931:
```
if (ws)
   kdv-=20;
```

7. *w1* has its Boolean value set depending upon whether *kdv* is greater than or equal to 10. w1 will be true, indicating a point-to-point calculation is in progress.

Line 934:
```
w1=kdv>=10;
```

8. If *w1* is true, 10 is subtracted from the value of kdv, leaving only a single digit value of *mdvar*; either 0, 1, 2, or 3.

Line 936:
```
if (w1)
   kdv-=10;
```

9. If the value of *kdv* is now less than zero, or more than 3, either case being outside the correct range of zero to 3, then:
    a. *kdv* is set to be equal to zero.
    b. *Prop.kwx* is set to be equal to the greater of *prop.kwx* or 2, indicating that an impossible parameter has been replaced with a default value.

Line 939:

```
if (kdv<0 || kdv>3)
{
        kdv=0;
        prop.kwx=mymax(prop.kwx,2);
}
```

10. If *lvar* was equal to 3, then case "three" applies, and:

a. *q* is set to be equal to log(0.133**prop.wn*)
   where
   prop.wn is the wave number, equal to:
   (the frequency in MHz / 47.7 MHz*meters)
b. *gm* is set to be equal to: $cfm1 + cfm2/(cfm3*q)^2 + 1$
c. *gp* is set to be equal to: $cfp1 + cfp2/(cfp3*q)^2 + 1$

Line 945:

```
 case 3:
q=log(0.133*prop.wn);

/* gm=cfm1+cfm2/(pow(cfm3*q,2.0)+1.0); */
/* gp=cfp1+cfp2/(pow(cfp3*q,2.0)+1.0); */

gm=cfm1+cfm2/((cfm3*q*cfm3*q)+1.0);
gp=cfp1+cfp2/((cfp3*q*cfp3*q)+1.0);
```

11. If *lvar* was equal to 2, then case "two" applies, and the value of dexa is set to be equal to:

$(18e6*prop.he[0])^{1/2} + (18e6*prop.he[1])^{1/2} + (575.7e12/prop.wn)^{1/3}$

where

prop.he[0] is the transmitter antenna effective height above ground
prop.he[1] is the receive antenna effective height above ground
prop.wn is the wave number, equal to:
(freq. in MHz / 47.7 MHz*meters)

Line 954:

```
case 2:

dexa=sqrt(18e6*prop.he[0])+sqrt(18e6*prop.he[1])+pow((575.7e1
2/prop.wn),THIRD);
```

12. If *lvar* was equal to 1, then case "one" applies, and;

a. if *prop.dist* is less than *dexa*, then *de* is set to be equal to 130,000**prop.dist/dexa.*
b. An else statement follows; therefore, if *prop.dist* is equal to or greater than *dexa*, then *de* is set to be equal to 130,000 + *prop.dist* – *dexa.*

Line 957:

```
case 1:
if (prop.dist<dexa)
        de=130e3*prop.dist/dexa;
else
        de=130e3+prop.dist-dexa;
}
```

13. Subroutine ***curve*** is called with inputs (*cv1*,*cv2*,*yv1*,*yv2*,*yv3*,*de*);
The subroutine inserts the parameters chosen by the climate code from the table into the curve-fitting equation:

$$curv(c1,c2,x1,x2,x3) = (c1 + c2)/(1. + ((de - x2)/x3)^2))*((de/x1)^2)/(1. + ((de/x1)^2))$$

Subroutine ***curve*** then returns the value calculated by the equation, and *vmd* is set to be equal to the value returned.

Line 964: vmd=curve(cv1,cv2,yv1,yv2,yv3,de);

14. Subroutine ***curve*** is called with inputs (*csm1*,*csm2*,*ysm1*,*ysm2*,*ysm3*,*de*);
The subroutine inserts the parameters chosen by the climate code from the table into the curve-fitting equation:

$$curv(c1,c2,x1,x2,x3) = (c1 + c2)/(1. + ((de - x2)/x3)^2))*((de/x1)^2)/(1. + ((de/x1)^2))$$

Subroutine ***curve*** then returns the value calculated by the equation, and *sgtm* is set to be equal to this value multiplied by *gm*.

Line 965: sgtm=curve(csm1,csm2,ysm1,ysm2,ysm3,de)*gm;

15. Subroutine ***curve*** is called with inputs (*csp1*,*csp2*,*ysp1*,*ysp2*,*ysp3*,*de*);
The subroutine inserts the parameters chosen by the climate code from the table into the curve-fitting equation:

$$curv(c1,c2,x1,x2,x3) = (c1 + c2)/(1. + ((de - x2)/x3)^2))*((de/x1)^2)/(1. + ((de/x1)^2))$$

Subroutine ***curve*** then returns the value calculated by the equation, and *sgtp* is set to be equal to this value multiplied by *gp*.

Line 966: sgtp=curve(csp1,csp2,ysp1,ysp2,ysp3,de)*gp;

16. The value of *sgtd* is set to be equal to *sgtp*csd1*;

Line 967: sgtd=sgtp*csd1;

17. The value of t*gtd* is set to be equal to (*sgtp – sgtd)*zd*;

Line 968: tgtd=(sgtp-sgtd)*zd;

18. If *wl* is true, a point to point calculation is in progress, and *sgl*, the standard deviation of location variability, is set to be equal to zero.

Line 970: if (w1)
sgl=0.0;

19. An else statement follows, so if *wl* is false, indicating an area mode or an interference problem mode calculation is in progress, then:

a. *q* is set to be equal to: $1.0-0.8^{(-prop.dist/50000)})*prop.dh*prop.wn$
where
prop.dist is the total RF path length,
prop.dh is the terrain irregularity factor delta h (Δh),
prop.wn is the wave number, = (freq. MHz/47.7 MHz*m.)

b. then *sgl*, the standard deviation of location variability, is set to be equal to: 10.0 * *q*/(*q*+13.0);

Line 972:
```
else
{
        q=(1.0-0.8*exp(-prop.dist/50e3))*prop.dh*prop.wn;
        sgl=10.0*q/(q+13.0);
}
```

20. If *ws* is true, indicating that an interference study is in progress, then *vs0* is set to be equal to zero.

Line 978:
```
if (ws)
   vs0=0.0;
```

21. An else statement follows, so for an area mode or point-to-point mode calculation, then:

a. *temp1* is set to be equal to $(5 + 3^{(-de/100000)})$, and is then used to calculate *vs0*:
b. *vs0* is set to be equal to $(5 + 3^{(-de/100000)})^2$

Line 980:
```
        else
{
        /* vs0=pow(5.0+3.0*exp(-de/100e3),2.0); */
        temp1=(5.0+3.0*exp(-de/100e3));
        vs0=temp1*temp1;

}
```

22. The setup adjustments required by the *lvar* switch code have been accomplished. Therefore, *propv.lvar* is reset to zero.

propv.lvar=0;

Line 988: propv.lvar=0;
}

23. The internal argument variables zt, zl, and zc are set to be equal to the percentages of the time, location, and confidence variants stated in the input values:

zt=zzt;	time variant (.01 to .99)
zl=zzl;	location variant (.01 to .99), or 0.0 for a call from the ***point_to_point*** subroutine.
zc=zzc;	confidence variant (.01 to .99)

Line 991: zt=zzt;
zl=zzl;
zc=zzc;

24. The mix of statistical variants to be used for the current operating mode are set up using a **switch** command, with the active case selected by the value of *kdv*. The value of *kdv* was set in steps 5 to 8 to be equal to the units value of the mode of variability (mdvar) code, and checked in step 9 to make sure that it was one of the following four integers; if it was not, zero was substituted. For:

a. *kdv* = 0 - Single message mode. Time, location and situation variability are combined together to give a confidence level. Case zero applies, and:
   i. the value of *zt* (time variant) is set to be equal to the value of *zc* (confidence variant).
   ii. the value of *zl* (location variant) is set to be equal to the value of *zc* (confidence variant).
b. *kdv* = 1 – Accidental mode. Reliability is given by time availability. Confidence is a combination of location and situation variability. Case one applies, and the value of *zl* (location variant) is set to be equal to the value of *zc* (confidence variant).
c. *kdv* = 2 – Mobile mode. Reliability is a combination of time and location variability. Confidence is given by the situation variability. Case two applies, and: the value of *zl* (location variant) is set to be equal to the value of *zt* (time variant).
d. *kdv* = 3 – Broadcast mode. Reliability is given by the statement of [at least] qT of the time in qL of the locations. Confidence is given by the situation variability. Case three applies; the variants are independent.

```
Line 995:       switch (kdv)
                {
                        case 0:
                        zt=zc;
                        zl=zc;
                        break;

                        case 1:
                        zl=zc;
                        break;

                        case 2:
                        zl=zt;
                }
```

25. An **if** statement is initiated. If the value of either *fabs(zt)*, *fabs(zl)* or *fabs(zc)* is greater than 3.1, then the value of *prop.kwx*, (the error indicator *errnum*), is set to be equal to the greater of *prop.kwx* or 1, indicating "caution, parameters are close to limits".

```
Line 1010: if (fabs(zt)>3.1 || fabs(zl)>3.1 || fabs(zc)>3.1)
                prop.kwx=mymax(prop.kwx,1);
```

26. An **if** statement is initiated; if the value of *zt*, the time variant, is less than zero, then the value of *sgt* is set to be equal to the value of *sgtm* calculated in step 14.

```
Line 1013:      if (zt<0.0)
                sgt=sgtm;
```

27. An **else if** statement follows, so if the value of *zt*, the time variant, is equal to or greater than zero and less than or equal to *zd*, then the value of *sgt* is set to be equal to the value of *sgtp* calculated in step 15.

```
                sgt=sgtp;

Line 1016: else if (zt<=zd)
                sgt=sgtp;
```

28. An **else** statement follows, so if the value of *zt*, the time variant, is equal to or greater than zero and greater than *zd*, then the value of *sgt* is set to be equal to the value of: *sgtd + tgtd / zt*..

```
Line 1019:      else
                sgt=sgtd+tgtd/zt;
```

29. The value of *temp1* is set to be equal to *sgt*zt*, the value of *temp2* is set to be equal to *sgl*zl*, and then the values of *temp1* and *temp2* are used to calculate the value of *vs* to be equal to:

$$= vs0 + (sgt*zt)^2/(rt+zc*zc) + (sgl*zt)^2/(rl+zc*zc);$$

Line 1024:
```
temp1=sgt*zt;
temp2=sgl*zl;

vs=vs0+(temp1*temp1)/(rt+zc*zc)+(temp2*temp2)/(rl+zc*zc);
```

30. An **if** statement is inititated. If kdv is equal to zero, indicating an area mode calculation, then:

a. The value of *yr* is set to be equal to zero.
b. The value of *propv.sgc* is set to be equal to: $((sgt)^2 + (sgl)^2 + vs)^{1/2}$

Line 1029:
```
if (kdv==0)
{
yr=0.0;
propv.sgc=sqrt(sgt*sgt+sgl*sgl+vs);
}
```

31. An **else if** statement follows; so if kdv is equal to one, indicating a point-to-point mode calculation, then:

a. The value of *yr* is set to be equal to *sgt * zt*.
b. The value of *propv.sgc* is set to be equal to: $((sgl)^2 + vs)^{1/2}$

Line 1035:
```
else if (kdv==1)
{
yr=sgt*zt;
propv.sgc=sqrt(sgl*sgl+vs);
}
```

32. An **else if** statement follows; so if kdv is equal to two, indicating a interference study calculation, then:

a. The value of *yr* is set to be equal to $((sgt)^2 + (sgl)^2)^{1/2} * zt$.
b. The value of *propv.sgc* is set to be equal to: $(vs)^{1/2}$

Line 1041:
```
else if (kdv==2)
{
yr=sqrt(sgt*sgt+sgl*sgl)*zt;
propv.sgc=sqrt(vs);
}
```

33. An **else** statement follows; so if if *kdv* is not equal to 0, 1, or 2, it is by default equal to 3. This indicates a broadcast mode calculation, so:

a. The value of *yr* is set to be equal to *sgt*zt* + *sgl* zl*.
b. The value of *propv.sgc* is set to be equal to: $(vs)^{1/2}$

```
Line 1047:      else
        {
                yr = sgt*zt + sgl*zl;
                propv.sgc=sqrt(vs);
        }
```

34. The value of *avarv* is then set to be equal to:

*prop.aref – vmd – yr – propv.sgc*zc*;

where

prop.aref
vmd
yr
propv.sgc
zc;

```
Line 1053:      avarv=prop.aref-vmd-yr-propv.sgc*zc;
```

35. If the value of *avarv* is less than zero, then the value of *avarv* is reset to be:

avarv = *avarv**(29.0-*avarv*)/(29.0-10.0**avarv*);

```
Line 1055:      if (avarv<0.0)
                avarv=avarv*(29.0-avarv)/(29.0-10.0*avarv);
```

36. The subroutine ***avar*** then returns the value of *avarv*.

```
Line 1058:      return avarv;
        }
```

Chapter 21: Curve

<u>Curve</u> (a.k.a. CURV) subroutine ***curve***

Note: Used with point-to-point and area modes. Called by ***avar***, mid-routine.

From ITMD Section 30:

Function *curv:*

$curv(c1,c2,x1,x2,x3) = (c1 + c2)/(1. + ((de - x2)/x3)^2))*((de/x1)^2)/(1. + ((de/x1)^2))$

Subroutine Call inputs:

Double constants:

&c1
&c2
&x1
&x2
&x3
&de

defines private, or local, arguments:

double:

temp1
temp2

This subroutine:

215. Sets temp1 to be equal to: $((de - x2)/x3)^2$
216. Sets temp2 to be equal to: $(de/x1)^2$
217. Calculates and returns the value of $(c1 + c2/(1 + temp1))*temp2/(1 + temp2)$.

Line 846:
```
temp1=(de-x2)/x3;
temp2=de/x1;

temp1*=temp1;
temp2*=temp2;

return (c1+c2/(1.0+temp1))*temp2/(1.0+temp2);
```

In the Area Mode:

Note this comment as to the use of the area mode:

In 1982, from "A Guide to th Use of the ITS Irregular Terrain Model in the Area Prediction Mode", by George Hufford, Anita Longley, and W.A. Kissick wrote:

"In the case of a specific operational area or a specific coverage area, one is confronted with a different problem. Here one has a large multitude of possible propagation paths, each of which can presumably be described in detail. One might, therefore, want to consider point-to-point calculations for each of them. But the sheer magnitude of the required input data makes one hesitant. An alternative, which requires far less input data, is to use an area prediction model, particularly if the model provides by itself the required statistics. Even in the case of a specific communications link, the required detailed information for the propagation path may be unobtainable so that one is forced to use the less demanding area prediction model. Of course, in doing so, one expects to lose in precision and in the dependability of the results."

Today, SPLAT!, SPLAT with PLOP, and many commercial ITM software packages, can do these detailed, all-points calculations, using any of several available detailed databases. The average desktop PC can run the program. So the use of area mode is essentially obsolete. The subroutine "Area", and its associated area-mode only subroutines, is included only so the book will be a complete reference to the ITMDLL.cpp.

Chapter 22: Area

"Area Mode Calculations" subroutine, ***area***.

Note: This is the main subroutine for the area mode. Calls ***qerfi***, ***qlrps***, ***qlra***, ***lrprop,*** and then ***avar***.

Uses information in arrays and input values listed (no terrain elevations) as inputs above in order to calculate dbloss, the path attenuation (radio signal strength loss) along the path between two terminals, a transmit site and a receive location.

From ITMD Sections 1, 2, and 42:

Call inputs:

long ModVar,	mode of variability
double deltaH,	delta H, (Δh), the terrain irregularity factor
double tht_m,	transmitter antenna height above ground level, in meters
double rht_m,	receive antenna height above ground level, in meters
double dist_km,	total RF path distance, in kilometers
int TSiteCriteria,	transmitter site selection criteria
int RSiteCriteria,	receive site selection criteria
double eps_dielect,	earth dielectric constant
double sgm_conductivity,	earth conductivity
double eno_ns_surfref,	refractivity of the atmosphere (a.k.a. atmospheric bending constant)
double frq_mhz,	frequency in MHz, range 20 to 20000 MHz
int radio_climate,	klim, or radio climate code (1 to 7; the nominal default is 5, Continental Temperate)
int pol,	polarity of the RF signal and receive antenna 0=H, 1=V
double pctTime,	time variant, expressed as a decimal percentage (.01 to .99)
double pctLoc,	location variant, as a decimal percentage (.01 to .99)
double pctConf,	situational confidence variant, decimal percent (.01 to .99)

User Selected Input Parameters:

pol:	0-Horizontal, 1-Vertical
TsiteCriteria:	0 - random, 1 - careful, 2 - very careful
RSiteCriteria:	0 - random, 1 - careful, 2 - very careful

radio_climate: 1-Equatorial,
2-Continental Subtropical,
3-Maritime Tropical,
4-Desert,
5-Continental Temperate, (nominal default)
6-Maritime Temperate, Over Land,
7-Maritime Temperate, Over Sea

ModVar: 0 - Single:
pctConf is "Time/Situation/Location";
[pctTime, pctLoc not used]
1 - Individual:
pctTime is "Situation/Location",
pctConf is "Confidence",
[pctLoc not used]
2 - Mobile:
pctTime is "Time/Locations (Reliability)",
pctConf is "Confidence",
[pctLoc not used]
3 - Broadcast:
pctTime is "Time",
pctLoc is "Location",
pctConf is "Confidence"

pctTime, .01 to .99
pctLoc, .01 to .99
pctConf: .01 to .99

Outputs:
errnum: (a.k.a. kwx) Error indicator code:
0- No Error.
1- Warning: Some parameters are nearly out of range.
Results should be used with caution.
2- Note: Default parameters have been substituted for impossible ones.
3- Warning: A combination of parameters is out of range.
Results are probably invalid.
Other - Warning: Some parameters are out of range.
Results are probably invalid.

Outputs:

double &dbloss, db loss, a.k.a. aref, the reference attenuation
char *strmode, string mode for printout: not used at this time
int &errnum error number, a.k.a. kwx, the error indicator

Declares private, or local, arguments:

prop_type prop array *prop* with array elements:
propv_type propv array *propv* with array elements:
propa_type propa array *propa* with array elements:

```
double zt,      time variant
       zl,      location variant
       zc,      confidence variant (situational variant)
       xlb;
double fs;      free space loss
long ivar;
double eps,     earth dielectric constant
       eno,     atmospheric bending constant (refractivity of the atmosphere)
       sgm;
long ipol;      polarity
int kst[2];
```

This subroutine:

218. Sets the value of array kst[0] to be equal to the user input value set for TsiteCriteria, and the value of array kst[1] to be equal to the user input value set for RsiteCriteria.

```
Line 1510:   kst[0]=(int)TSiteCriteria;
             kst[1]=(int)RSiteCriteria;
```

219. Subroutine ***qerfi*** is called with input (*pctTime*/100.0). Subroutine ***qerfi*** returns *v*, the standard normal deviate as a function of the complementary probability. The value of zt is then set to equal the value of *v*.

```
Line 1512: zt=qerfi(pctTime/100.0);
```

220. Subroutine ***qerfi*** is called with input (*pctLoc*/100.0). Subroutine ***qerfi*** returns *v*, the standard normal deviate as a function of the complementary probability. The value of zl is then set to equal the value of *v*.

```
Line 1513: zl=qerfi(pctLoc/100.0);
```

221. Subroutine ***qerfi*** is called with input (*pctConf*/100.0). Subroutine ***qerfi*** returns *v*, the standard normal deviate as a function of the complementary probability. The value of zc is then set to equal the value of *v*.

```
Line 1514: zc=qerfi(pctConf/100.0);
```

222. The values of the following local arguments and array values are set:

eps = eps_dielect;	earth dielectric constant
sgm = sgm_conductivity;	earth conductivity
eno = eno_ns_surfref;	atmospheric bending constant (refractivity)
prop.dh = deltaH;	delta h (Δh) the terrain irregularity factor
prop.hg[0] = tht_m;	transmitter antenna height AGL, in meters
prop.hg[1] = rht_m;	receiver antenna height AGL, in meters
propv.klim = (long)radio_climate;	radio climate code; as used in itm.cpp
propv.klim = (int32) radio_climate;	radio climate code; as used in ITMDLL.cpp,
prop.ens = eno;	ens is refractivity of atmosphere reduced to sea level (atmospheric bending constant) here set equal to eno.
prop.kwx = 0;	error code; reset to zero
ivar = (long)ModVar;	variable mode;
ipol = (long)pol;	polarity; 0 for Horizontal, 1 for Vertical

Line 1515:

```
eps=eps_dielect;
sgm=sgm_conductivity;
eno=eno_ns_surfref;
prop.dh=deltaH;
prop.hg[0]=tht_m;
prop.hg[1]=rht_m;
/* propv.klim = (__int32) radio_climate; -KD2BD replaced below */
propv.klim=(long)radio_climate;
prop.ens=eno;
prop.kwx=0;
ivar=(long)ModVar;
ipol=(long)pol;
```

223. Subroutine ***qlrps*** is called with inputs:

a. frq_mhz, frequency in MHz
b. 0.0, general system elevation, here set equal to zero.
c. eno, surface refractivity reduced to sea level (atmospheric bending constant) nominally 301.000 N-units
d. ipol, polarity
e. eps, earth's dielectric constant, a.k.a. polarization constant or relative permittivity
f. sgm, earth conductivity, a.k.a. ground constant, nominally .005 Siemens/meter

Subroutine ***qlrps*** outputs to prop_type structure (prop_type & prop) :

g. *Prop.wn* wave number
h. *prop.ens* surface refractivity (updated by ***qlrps***)
i. *prop.gme* effective earth curvature
j. *prop.zgnd* surface impedance
k. *prop.zgndreal* real surface impedance (resistance component)

l. *prop.zgndimag* imaginary surface impedance (reactive component)

Line 1527: qlrps(frq_mhz, 0.0, eno, ipol, eps, sgm, prop);

224. Subroutine ***qlra*** is called with inputs:

a. kst, array holding transmit & receive site selection criteria
b. propv.klim, the radio climate code
c. ivar, variation mode
d. array prop, with elements:

prop.hg[1]	receive site ground height ASL in meters
prop.hg[0]	transmit site ground height above sea level(ASL) m.
prop.dh	delta h (Δh), the terrain irregularity factor
prop.gme	effective earth curvature = 1/effective earth radius

Subroutine ***qlra*** populates array prop with calculated values for:

prop.he[0]	transmit antenna effective height above ground level (AGL) in meters.
prop.he[1]	receive antenna effective height AGL in meters
prop.dl[0]	distance to horizon(obstacle) from transmitter site
prop.dl[1]	distance to horizon(obstacle) from receiver site
prop.the[0]	take-off angle from transmitter antenna to horizon or the top of the highest visible obstacle
prop.the[1]	take-off angle from receiver antenna to horizon or the top of the highest visible obstacle
prop.mdp	mode of propagation (set to 1)
prop.lvar	mode reset level switch
prop.klim	radio climate code

Line 1528: qlra(kst, propv.klim, ivar, prop, propv);

225. An **if** statement is initiated. If propv.lvar, the mode reset level switch, is less than one, then prop.lvar is set to be equal to one.

Line 1530: if (propv.lvar<1)
propv.lvar=1;

226. Subroutine ***lrprop*** is called with inputs:

a. (*dist_km**1000.0) for *d*, the distance input, required in meters
b. array *prop,* and;
c. array *propa*.

Subroutine ***lrprop*** the populates *prop.aref* with the calculated value of *aref*, the reference attenuation.

Line 1533: lrprop(dist_km*1000.0, prop, propa);

227. The free space attenuation *fs*, is the amount of loss of signal transmitted from an isotropic antenna centered in a sphere, as received by a frequency tuned antenna embedded in the inner surface of the sphere. The free space attenuation is calculated using this equation, where the inputs are frequency in MHz and radius of the sphere (path distance) in kilometers, and the output is in decibels (dB):

$fs=32.45+20.0*\log_{10}(prop.dist/1000.0) +20.0*\log_{10}(frq\_mhz);$

Line 1534: fs=32.45+20.0*log10(frq_mhz)+20.0*log10(prop.dist/1000.0);

228. Subroutine ***avar*** is called with inputs:

a. zt, time variant
b. zl, location variant
c. zc, confidence (situational) variant
d. array prop,
e. array propv.

Subroutine ***avar*** returns the value of *avarv,* the reference attenuation with additional attenuation due to time, location, and situational variations, in decibels (dB). The value of *xlb* is then set to equal the sum of the free space attenuation *fs* and the value of *avarv*.

Line 1535: xlb=fs+avar(zt, zl, zc, prop, propv);

229. *dbloss* is then set to be equal to the value of *xlb*.

Line 1536: dbloss=xlb;

230. An **if** statement is initiated. If *prop.kwx*, the error indicator, is equal to zero, then the value of *errnum*, (error number), is set to be equal to zero.

```
Line 1538:  if (prop.kwx==0)
            errnum=0;
```

231. An **else** statement follows, so if *prop.kwx* is not equal to zero, then the value of *errnum* is set to equal the value of *prop.kwx*.

```
Line 1540:  else
            errnum=prop.kwx;
}
```

Chapter 23: Qerfi

QERF Inverted. The inverse of QERF. Subroutine ***qerfi***.

Note: This is the parameter preparation subroutine for the area mode. Called by ***area***. Calls ***mymax.***

This is used to prepare the model in the area prediction mode. The inverse of qerf, gives the standard normal deviate as a function of the complementary probability truncated at 0.000001 and 0.999999 .

From ITMD Section 51:

"The inverse of *qerf* ---the solution for x to q = Q(x). The approximation is due to C. Hastings, Jr. ("Approximations for digital computers," Princeton Univ. Press, 1955) and the maximum error should be $4.5e10^{-4}$."

Call inputs:

```
double q
```

Declares private, or local, arguments:

```
double x,
       t,
       v;
double c0=2.515516698;
double c1=0.802853;
double c2=0.010328;
double d1=1.432788;
double d2=0.189269;
double d3=0.001308;
```

In this subroutine:

232. The value of *x* is set to be equal to 0.5 – *q*.

Line 359: x=0.5-q;

233. The value of *t* is set to be equal to the greater of (0.5 – fabs(*x*)), or 0.000001 .

[Note; fabs is a math.h function that returns the absolute value of *x* (/x/).]

Line 360: t=mymax(0.5-fabs(x),0.000001);

234. The value of *t* is reset to be equal to $(-2.0*\log(t))^{1/2}$

Line 361: t=sqrt(-2.0*log(t));

235. The value of *v* is set to be equal to:

$t-((c2*t+c1)*t+c0)/(((d3*t+d2)*t+d1)*t+1.0)$

Line 362: v=t-((c2*t+c1)*t+c0)/(((d3*t+d2)*t+d1)*t+1.0);

236. An **if** statement is initiated. If the value of x is less than zero, then the value of v is reset to be equal to – v.

Line 364:
```
if (x<0.0)
v=-v;
```

237. The subroutine ***qerfi*** then returns the value of *v*.
return v;

Line 367:
```
return v;
}
```

Chapter 24: Qlra

Quick Longley Rice Area parameter preparation subroutine, ***qlra***.

Note: This is the parameter preparation subroutine for the area mode. Calls ***mymin*** and ***mymax***.

From ITMD Sections 42:

This is used to prepare the model in the area prediction mode. Normally, one calls ***qlrps*** and then ***qlra***.

Call inputs:

kst,	array holding transmit & receive site selection criteria
klimx	a.k.a. propv.klim, the radio climate code
mdvarx	a.k.a. ivar, the variation mode code
prop_type & prop	array prop, with elements:
prop.hg[1]	receive site ground height ASL in meters
prop.hg[0]	transmit site ground height above sea level(ASL) m.
prop.dh	delta h (Δh), the terrain irregularity factor
prop.gme	effective earth curvature = 1/effective earth radius

propv_type &propv)

Outputs:

Subroutine ***qlra*** populates array prop with calculated values for:

prop.he[0]	transmit antenna effective height above ground level (AGL) in meters.
prop.he[1]	receive antenna effective height AGL in meters
prop.dl[0]	distance to horizon(obstacle) from transmitter site
prop.dl[1]	distance to horizon(obstacle) from receiver site
prop.the[0]	take-off angle from transmitter antenna to horizon or the top of the highest visible obstacle
prop.the[1]	take-off angle from receiver antenna to horizon or the top of the highest visible obstacle
prop.mdp	mode of propagation (set to 1)
prop.lvar	mode reset level switch
prop.klim	radio climate code

Declares private, or local, arguments:

double q

In this subroutine:

238. A two loop **for** loop is initiated. For *j* = 0 and then *j* = 1,

```
Line 440:    for (int j=0; j<2; ++j)
             {
```

239. An **if** statement is embedded in the **for** loop. If *kst[0]*, the transmitter site selection criteria, is less than or equal to zero, then *prop.he[0]*, the transmitter site effective height, is set to be equal to *prop.hg[0]*, the transmitter antenna height above ground level.

```
Line 442                if (kst[j]<=0)
                         prop.he[j]=prop.hg[j];
```

240. An **else** statement is embedded in the for loop. On the first **for** loop, if kst[0], the transmitter site selection criteria, is greater than zero, then:
   a. The value of q is set to be equal to four.
   b. An **if** statement is initiated, embedded within the **else** statement. If *kst[0]* is greater than zero, but is not equal to one, then the value of *q* is set to equal 9.0.
   c. A second **if** statement is initiated, embedded within the **else** statement. If *kst[0]* is greater than zero, and if the value of *prop.hg[0]* is less than 5.0, then the value of q is reset to be equal to the previous value of q multiplied by: sin(0.3141593**prop.hg*[0]).
   d. The value of prop.he[0] is then set to be equal to:

```
prop.hg[0]+(1.0+q)*exp(-mymin(20.0,2.0*prop.hg[0]/mymax(1e-3,prop.dh)));
```

```
Line 444:    else
             {
                     q=4.0;

                     if (kst[j]!=1)
                             q=9.0;

                     if (prop.hg[j]<5.0)
                             q*=sin(0.3141593*prop.hg[j]);

                     prop.he[j]=prop.hg[j]+(1.0+q)*exp(-
mymin(20.0,2.0*prop.hg[j]/mymax(1e-3,prop.dh)));
             }
```

241. The **for** loop continues its first loop, and:

a. The value of *q* is reset to be equal to: $(2.0*prop.he[0]/prop.gme)^{1/2}$
b. The value of *prop.dl[0]* is set to be equal to: $q*\exp(-0.07*(prop.dh/\text{mymax}(prop.he[0],5.0))^{1/2})$
c. The value of *prop.the[0]* is set to be equal to: $(0.65*prop.dh*(q/prop.dl[0]-1.0)-2.0*prop.he[0])/q$;

```
Line 457:   q=sqrt(2.0*prop.he[j]/prop.gme);
            prop.dl[j]=q*exp(-0.07*sqrt(prop.dh/mymax(prop.he[j],5.0)));
            prop.the[j]=(0.65*prop.dh*(q/prop.dl[j]-1.0)-2.0*prop.he[j])/q;
        }
```

242. The **for** loop then starts its second loop, and we return to the first **if** statement embedded in the **for** loop. On the second **for** loop, If kst[1], the receive site selection criteria is less than or equal to zero then prop.he[1], the receive site effective height, is set to be equal to prop.hg[1], the receive antenna height above ground level.

243. An **else** statement is embedded in the for loop. On the second **for** loop, if *kst[1]*, the receive site selection criteria, is greater than zero, then:

a. The value of q is set to be equal to four.
b. An **if** statement is initiated, embedded within the **else** statement. If *kst[1]* is greater than zero, but is not equal to one, then the value of *q* is set to equal 9.0.
c. A second **if** statement is initiated, embedded within the **else** statement. If *kst[1]* is greater than zero, and if the value of *prop.hg[1]* is less than 5.0, then the value of *q* is reset to be equal to the previous value of *q* multiplied by: $\sin(0.3141593*prop.hg[1])$.
d. The value of *prop.he[1]* is then set to be equal to:

```
prop.hg[0]+(1.0+q)*exp(-mymin(20.0,2.0*prop.hg[0]/mymax(1e-3,prop.dh)));
```

244. The **for** loop continues its second loop, and:

a. The value of *q* is reset to be equal to: $(2.0*prop.he[1]/prop.gme)^{1/2}$
b. The value of *prop.dl[1]* is set to be equal to: $q*\exp(-0.07*(prop.dh/\text{mymax}(prop.he[1],5.0))^{1/2})$
c. The value of *prop.the[1]* is set to be equal to: $(0.65*prop.dh*(q/prop.dl[1]-1.0)-2.0*prop.he[1])/q$;

245. The **for** loops are now complete. The value of prop.mdp, the mode of propagation, is set to be equal to one.

```
Line 462:   prop.mdp=1;
```

246. The value of prop.lvar, the level of variation mode setup switch value that controls the setup of subroutine ***avar***, is now reset to be the greater of either three, or the existing value of propv.lvar.

Line 463: propv.lvar=mymax(propv.lvar,3);

247. An **if** statement is initiated. If the value of mdvarx, the mode of variation, is greater than or equal to zero, then:
   a. The value of *propv.mdvar* is set to be equal to the value of *mdvarx*.
   b. The value of *propv.lvar*, the level of variation mode setup switch value, is reset to be equal to the greater of four, or the existing value of *propv.lvar*.

```
Line 465:   if (mdvarx>=0)
            {
                propv.mdvar=mdvarx;
                propv.lvar=mymax(propv.lvar,4);
            }
```

248. An **if** statement is initiated. If the value of *klimx*, the climate code, is greater than zero, then:
   a. The value of *propv.klim* is set to be equal to the value of *klimx*.
   b. The value of *propv.lvar* is set to be equal to five.

```
Line 471:   if (klimx>0)
            {
                propv.klim=klimx;
                propv.lvar=5;
            }
}
```

Chapter 25: Qerf

Q_{erf} The standard normal complementary probability subroutine ***qerf***.

Note: This is an independent, optional subroutine; not directly used in Longley–Rice. However, subroutine ***qerfi*** is the inverse of this function, and is called by ***area***. Calls ***mymax.***

The standard normal complementary probability approximation.

From ITMD Section 50:

"The standard normal complementary probability --- the function:

$Q(x) = [1/ (2*\pi)^{1/2}] *$ {integral from x to infinity of} $e^{-(t*t)/2} *dt$

The approximation is due to C. Hastings, Jr. ("Approximations for digital computers," Princeton Univ. Press, 1955) and the maximum error should be $7.5e10^{-8}$."

Call inputs:

```
const double &z
```

Declares private, or local, arguments:

```
double b1=0.319381530,
       b2=-0.356563782,
       b3=1.781477937;
double b4=-1.821255987,
       b5=1.330274429;
double rp=4.317008,
       rrt2pi=0.398942280;

double t,
       x,
       qerfv;
```

In this subroutine:

249. The value of *x* is set to equal the value of *z*.

Line 1226: x=z;

250. The value of *t* is set to be equal to the (absolute) value of /*x*/.

Line 1227: t=fabs(x);

251. An **if** statement is initiated. If the value of **t** is greater than or equal to 10, then the value of qerfv is set to be equal to zero.

Line 1229: if (t>=10.0)
qerfv=0.0;

252. An **else** statement follows the **if** statement. Therefore, if the value of *t* is less than 10, then:

a. The value of *t* is reset to be equal to 4.317008/(*t* + 4.317008).
b. The value of *qerfv* is set to be equal to:

exp(-0.5*x*x)*rrt2pi*((((b5*t+b4)*t+b3)*t+b2)*t+b1)*t;

where:

b1=0.319381530, b2=-0.356563782,
b3=1.781477937, b4=-1.821255987,
b5=1.330274429, rrt2pi=0.398942280;

Line 1231: else
{
t=rp/(t+rp);
qerfv=exp(-0.5*x*x)*rrt2pi*((((b5*t+b4)*t+b3)*t+b2)*t+b1)*t;
}

253. An if statement is initiated. If the value of x is less than zero, then the value of *qerfv* is reset to be equal to 1 – *qerfv*.

Line 1237: if (x<0.0)
qerfv=1.0-qerfv;

254. Subroutine ***qerf*** then returns the value of *qerfv*.

Line 1240: return qerfv;
}

Version 7; The June, 2007 update of the ITMDLL.cpp

Chapter 26: The dissapearance of Fred's Lrprop and changes to the Paul M Lrprop.

The c++ source code has been made available since 2003 as file: ITMDLL.cpp, located on the NTIA's website at http://flattop.its.bldrdoc.gov/itm.html. This port is a direct as possible conversion of the FORTRAN code found in the appendix to NTIA Report TR-82-100, "A Guide to the Use of the ITS Irregular Terrain Model in the Area Prediction Mode" [April, 1982], at http://www.its.bldrdoc.gov/pub/ntia-rpt/82-100/, converted to a Windows DLL-friendly code. The published source code had remained unchanged from at least November 5, 2003 until June 26, 2007. On June 26, 2007, the NTIA quietly released an update to the ITMDLL.cpp software code, via its website.

What happened to Fred's Lrprop?

In the new version 7 of the ITMDLL.cpp, released vial the NTIA website on June 29, 2007, the optional subroutine "freds lrprop", found in the previous version, had been removed. The NTIA made no mention on the website as to why, and there is no note in the source code to explain the removal.

Other Changes to the ITMDLL.cpp:

The earlier version, 1.2.2, is still available at ftp://flattop.its.bldrdoc.gov/itm, as file: ITMDLL_old11-5-03.cpp. The old code does not have a version number embedded in the source code; the new one reports out version 7.0. Caution may be required in making use of the updated version, as the FCC has specifically stated, in OET 69, "Longley Rice Methodology for Evaluation TV Coverage and Interference", that version 1.2.2 is the "version used by the FCC for its evaluations". The FCC also relies on Longley–Rice in estimating television reception coverage for satellite reception waivers, and accepts submissions using Longley-Rice to show whether Cities of License receive city-grade coverage from far-away terrain-limited transmitter sites.

What changes in the new version? I found 205 lines that have been removed, of which 2 are consolidated. ***freds lrprop***, formerly occupying lines 370 to 543, is 174 lines of the 205 removed; 8 lines of code, and 21 lines of comments, also have been removed. There are eight lines of active, mandatory code that have been affected by changes, and four new subroutines, occupying 182 lines, have been added. The first two, ***point_to_pointMDH*** and ***point_to_pointDH***, provide welcome optional alternatives to the normally called ***point_to_point*** subroutine. The other two new subroutines are ***ITMAreadBLoss***, and ***ITMDLLVersion( )***. Including the spaces and bracket changes,

the old version 1.2.2 c++ code has 1,263 lines; the updated version 7.0 code has 1,239 lines.

I will be referring to the changes by line number; to follow along, I suggest that you download the new and old versions of the files and print them out; both are available at: ftp://flattop.its.bldrdoc.gov/itm. To keep track of the line numbers, I load the file into Bloodshed Software's freeware Dev-C++ version 5 (beta) IDE compiler, available at www.bloodshed.net/devcpp.html, and use the print setup and print functions to identify functions by color, word wrap, add line numbers in the margin, and then print.

As to the total active changes to the non-optional subroutines, the action all occurs in one place: the remaining version of subroutine lrprop (a.k.a PaulM_lrprop). Here there are changes between former lines 657 to 674. These changes include: (a.) the if statements and actions on lines 657 and 658, and on lines 665 and 666, are consolidated onto a single line each (a minor change), (b.) the else statement on line 670 becomes an **if** (! wq) statement (an action suggested in a former comment, removed in the updated version, that was on line 669); (c.) line 671, which states: propa.ak1=(a2-a1)/(d2-d1) is changed to propa.ak1=FORTRAN_DIM(a2-a1)/(d2-d1), and (d.) the else statement and its following actions, on lines 661 to 667, are removed. In the new code, the changed section occupies lines 466 to 473, so what occupied 18 lines now takes up 8 lines.

What does this accomplish? The changes to lrprop replace two "else" statements with a single "if" statement that does the same job; executing the calculations specified in equations 4.40, 4.41, and 4.42 of "The ITS Irregular Terrain Model, version 1.2.2 The Algorithm" by George Hufford, found at: http://flattop.its.bldrdoc.gov/itm/itm_alg.pdf . It appears that no code or math errors are repaired, or will be caused by, this change to ***lrprop***. It accomplishes the same task with fewer lines, so perhaps the code will execute a tiny bit faster.

Chapter 27: The Unveiling of: Point_to_PointDH, Point_to_PointMDH, ITMAreadBLoss, and ITMDLLVersion( )

What do the new subroutines do? The first two, ***point_to_pointDH*** and ***point_to_pointMDH***, provide optional alternatives to the ***point_to_point*** subroutine normally called by the separate input-output wraparound software for point-to-point mode signal path loss profiles and detailed area signal level mapping calculations. ***point_to_pointDH***, the "Point-to-point calculations, Delta H" subroutine has only two changes; (1.) It changes "char *strmode to char strmode[100] and moves it out of the line of declarations on line 920 to have its own place on line 1095. (2.) In its place on line 920 is a new declaration, **double** &deltaH, and at line 1132, a new line stating: deltaH = prop.dh; the second set of changes therefore declare a new local argument, deltaH, that can be read by a wraparound program without reading array "prop", in the same way that "errnum" can be read, and set its value equal to the value held by prop.dh, which represents delta H, (ΔH), the terrain irregularity factor.

point_to_pointMDH, "Point-to-point calculations with Mode numbers and Delta H", includes the second set of changes in ***point_to_pointDH*** above, and provides two additional significant, and welcome, changes. (1.) The string mode (strmode) setup to allow printing out the mode, which causes, for example, the reports in SPLAT to print out "Line-of-Sight Mode" hundreds or thousands of times, followed by "Single Horizon" etc., has been replace with a simple numerical code. The code is: -1, undefined; 0; line-of-sight, 5, Single Horizon, Diffraction; 6, Single Horizon, Troposcatter; 9, Double Horizon, Diffraction; and 10, Double Horizon, Troposcatter. The new comments included in the source code explain the numerical codes. The numerical value is loaded into a new argument: propmode, which is declared as **int** &propmode in line 994, allowing it to be read by the calling wraparound software in the same way as errnum can be read. Last but not least, (2.) The existing **point_to_point** subroutine calculates variable amounts of statistical reliability for percentage of time, and confidence, but calls subroutine ***avar*** with its location percentage set to zero. The new ***point_to_pointMDH*** subroutine declares a new argument, zloc, for location percentage, sets it to be equal to the value stored in array qerfi (locpct), and allows subroutine ***avar*** to be called with this value as the location percentage input, allowing statistics to be generated for time, location and confidence in the point_to_point mode, instead of the former limitation of only time and confidence.

But note this:

Line 1445: propv.mdvar=12;

Making use of the new access to the horizontal percent input requires resettin the mdvar =12 statement in the new point_to_pointMDH to 3, from the current setting of 12, prior to compiling the source code. That is awkward, at best.

NOTE: For TV or FM broadcast use, in point_to_point mode, the mdvar is set to Mobile mode, not Broadcast. The code for mdvar, the variability mode, which sets the mode of operation, is a tens and single digit code. The single digits represent:

0 - Single message mode. Time, location and situation variability are combined together to give a confidence level.

1 – Accidental mode. Reliability is given by time availability. Confidence is a combination of location and situation variability.

2 – Mobile mode. Reliability is a combination of time and location variability. Confidence is given by the situation variability.

3 – Broadcast mode. Reliability is given by the statement of –at least- qT of the time in qL of the locations. Confidence is given by the situation variability.

The tens code is: No tens; default to area mode; combined code is 0 to 3.

10 – For the point-to-point mode. Location variability is eliminated.

20 – For interference problems. Direct situation variability is eliminated. Note that there may be a small residual situational variability.

Note that in the point-to-point mode, *location variability is supposed to be eliminated,* according to "The Algorithm". In fact, there is a minor cautionary bug that causes the horizontal to be set equal to the time in area-mobile and interference-mobile situations; since the mdvar is hard coded in point-to-point, it is not considered serious enough to warrant inclusion in the seven significant sicknesses. See discussion in chapter 20, Avar, as this occurs in the subroutine ***avar*** in response to the setting of the mdvar code. So the hard-coded setting of mdvar equal to 12, hard-coded into all three of the point_to_point subroutines in the ITMDLL.cpp, means that the current version of ITMDLL.cpp is never intended to be used for broadcast calculation of location variability, unless the value of mdvar is changed in the source code and the ITMDLL is re-compiled.

Note: Therefore, for broadcast reception prediction use, the ITMDLL should be modified to allow external resetting of the mdvar; this is especially critical in light of the new optional subroutine point_to_pointMDP, which allows for setting the percentage value for location.

ITMAreadBLoss, "ITM Area dB Loss", when called, calls subroutine ***area,*** "Area mode calculations", and reports out the value of "dbloss" calculated by subroutine ***area. ITMDLLVersion( )***, "ITMDLL Version Number" when called, reports out "7.0".

The ITM predicts signal strength loss in three "modes". The first, starting at the transmission site, is the line of sight mode, which switches to the edge diffraction mode where the path threatens to graze the earth for smooth earth (at the horizon) or at an obstruction. As the path length increases, the edge diffraction dominant mode fades into the troposcatter dominant mode, and goes to full troposcatter mode at far distances. Even

after thirty years in use, there are still mathematical and coding errors and obsolete computer approximations in the ITM code that affect the quality of the predictions in all three modes. These problems have not been addressed, and remain uncorrected, in version 7.0. Many of these errors track back to, and can be seen in the FORTRAN version found in Appendix A of NTIA Report TR-82-100, and have therefore existed for a quarter of a century. In the following articles, I will address the coding and mathematical errors in the subroutines found in both c++ versions.