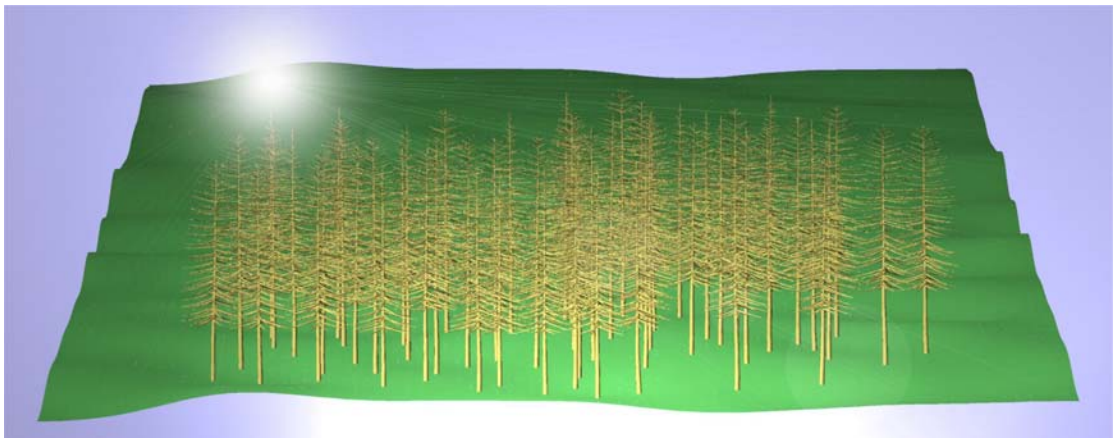
PolSARProSim

A Coherent, Polarimetric SAR Simulation of Forests for PolSARPro

(DRAFT) Design Document (v1.0)

Dr Mark L. Williams

August 2006



Contents

Executive Summary

This document describes the design of the software package **PolSARProSim**, intended to calculate simulated synthetic aperture radar (SAR) imagery of model forest stands. It is delivered to the University of Rennes as the first milestone of contract "PolInSAR Coherent Scattering and Imaging Code Development". This document serves as a reference, and as a general guide to the construction and operation of **PolSARProSim**. This document is not definitive but serves as a broad definition of overall capability, and contains many details of the planned implementation.

The document is divided into a number of sections. The introduction contains a statement of capability; a high-level description of **PolSARProSim**, describing the concept, its operation and the user-interface in general terms.

Following the introduction is a detailed, task-oriented description of the implementation of **PolSARProSim**. This is naturally divided into a number of stages: receipt of parameters describing the forest and SAR imaging process, construction of the forest, and calculation and output of the SAR images. Forest construction and SAR imaging processes are described first and a list of required parameters is generated during the discussion. This list is summarized at the end of the section on implementation.

PolSARProSim is designed to be an educational tool: providing simulated test data of sufficient fidelity to be used within the tutorial package of **PolSARPro v2.0**. In designing **PolSARProSim** and drawing up this document it has been anticipated that the user is sophisticated, yet not an expert in either forestry or PolInSAR techniques. The user will therefore have limited but sufficient influence, via a simple interface, over those parameters controlling the nature of the forest and the SAR imaging sensor.

The users' interface is to be implemented within **PolSARPro v2.0**, and numerical parameter values collected within **PolSARPro v2.0** will be passed to **PolSARProSim** via a file (either text or binary). **PolSARProSim** will then perform the requested simulations, generating an interferometric pair of fully polarimetric SAR images, and writing them to file, in an agreed format, for input back into **PolSARPro v2.0** for post-processing using PolInSAR techniques.

The goal is to make available **PolSARProSim** to run on any machine capable of running **PolSARPro v2.0**. Thus code will be written in ANSI standard C, and source code will be delivered that may be compiled as required for delivery on different platforms. Since the full variety of platforms may not be anticipated **PolSARProSim** is designed to have the smallest possible memory footprint, whilst maintaining efficiency of execution.

MLW. May, 2006.

1. Introduction

1.1 Concept

1.1.1 **PolSARProSim** is designed to be an educational tool: providing simulated test data of sufficient fidelity to be used within the tutorial package of **PolSARPro v2.0**. The purpose of the simulation is to provide simulated SAR images to illustrate the concepts of the PolInSAR lecture course.

1.1.2 The simulation is designed to overcome the limited availability of suitable data, and thereby remove an obstacle to learning. Whilst it would be possible to incorporate many examples of data with **PolSARPro v2.0** software, it is not feasible to anticipate all the situations of interest to potential users of the software: hence the simulation capability.

1.1.3 The user will be provided a simple interface that permits the ready description of a realistic forest model, and a flexible description of SAR imaging parameters. Having designed the forest and set the imaging scenario the simulation will be called from within **PolSARPro v2.0**, and in a short space of time (depending upon platform) the user will have available simulated polarimetric, interferometric SAR data for the scene and sensor of their choice.

1.1.4 The graphical user interface (GUI) used to assign parameter values will be implemented within **PolSARPro v2.0**. A list of parameter values chosen by the user will be communicated to the SAR simulation software via a text or binary file. **PolSARProSim** will then be called from within **PolSARPro v2.0** (with the output location under user control).

1.1.5 In addition to the final polarimetric, interferometric SAR imagery, **PolSARProSim** may generate additional files based upon user specifications. For example a digital elevation map (DEM) file may be generated, and/or a map of stem locations and heights with crown dimensions. When notified that the images are ready, the user may proceed to analyse the imagery using the algorithms from within **PolSARPro v2.0**.

1.1.6 The following is a short, high-level statement of capability for **PolSARProSim**:

1.2 Capability Statement

1.2.1 **PolSARProSim** will be called from within **PolSARPro v2.0** to perform SAR imaging calculations of a model forest.

1.2.2 Parameters controlling the operation of **PolSARProSim** will be passed via a file from **PolSARPro v2.0** having been gathered from within **PolSARPro v2.0** via a graphical user interface.

1.2.3 **PolSARProSim** will use the parameter file values to construct a scene to be imaged, including a digital elevation map, and an array of trees, with specified surface and forest properties, such as ground roughness and the mean height of the forest stand. The description of the forest scene may be reported to some extent as requested by the user.

1.2.4 **PolSARProSim** will then proceed to use the parameter file values to calculate an interferometric pair of SAR images of the forest using those SAR sensor values and interferometric baseline requested by the user. These images will be output in a format suitable for analysis within **PolSARPro v2.0.**, along with a record of ancillary information describing the imaging scenario, such as range to scene centre, centre frequency and baseline etc., into a text file.

1.3 Notable Features of PolSARProSim

1.3.1 Key features in the design of **PolSARProSim** are intended ease of use, minimum memory requirements, maximum speed of execution, and fidelity of simulated imagery sufficient for the educational requirement.

1.3.2 **PolSARProSim** is to be written mostly in ANSI C to help maintain portability across platforms.

1.3.3 Memory requirements are to be minimized during execution by storing only a single realization of tree architecture at any time, and by treating small plant scattering elements using stochastic techniques that preserve interferometric coherence.

1.3.4 Execution speed will as far as possible be optimized by using simplified forms for scattering by plant elements determined by size, and by modeling tree crowns as having uniform permittivity, thereby reducing the computational load associated with estimating the attenuation properties of the canopy, yet still preserving the inhomogeneity of the canopy in a realistic fashion.

1.3.5 Trees are to be represented and stored in parametric form, and realized using procedures coded according to chosen tree species, and dependent

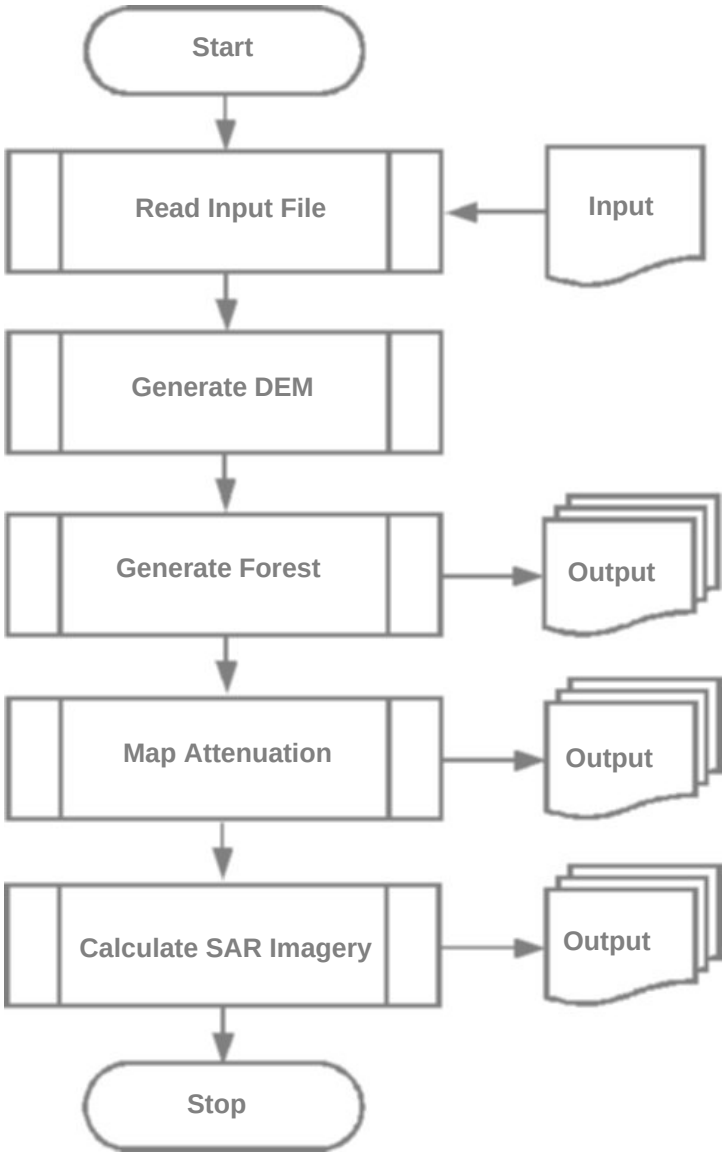
upon chosen tree heights through allometric equations. Tree realizations are subsequently divided into scattering elements for the SAR imaging calculation.

1.3.6 A layer of understorey vegetation is to be accommodated at a simple level. It will be homogeneous and of constant depth above local ground level. The layer is to be comprised of small scattering elements, with uniform spatial and orientational distributions, yielding an azimuthally symmetric, homogeneous layer.

1.3.7 A two-scale model is to be adopted for the underlying ground surface. The DEM generated using user-supplied parameters will provide the large-scale surface. This is to be divided into facets, taking their orientation from the large-scale surface, and assigned small-scale roughness for the scattering calculation.

1.3.8 Images are to be output in ground range and azimuth, and the backscattering alignment (BSA) convention is to be adopted.

1.4 Flow Chart for PolSARProSim Operation



1.5 Document Description

1.5.1 The remainder of this document describes the details of **PolSARProSim** algorithms and their implementation.

2. General Considerations

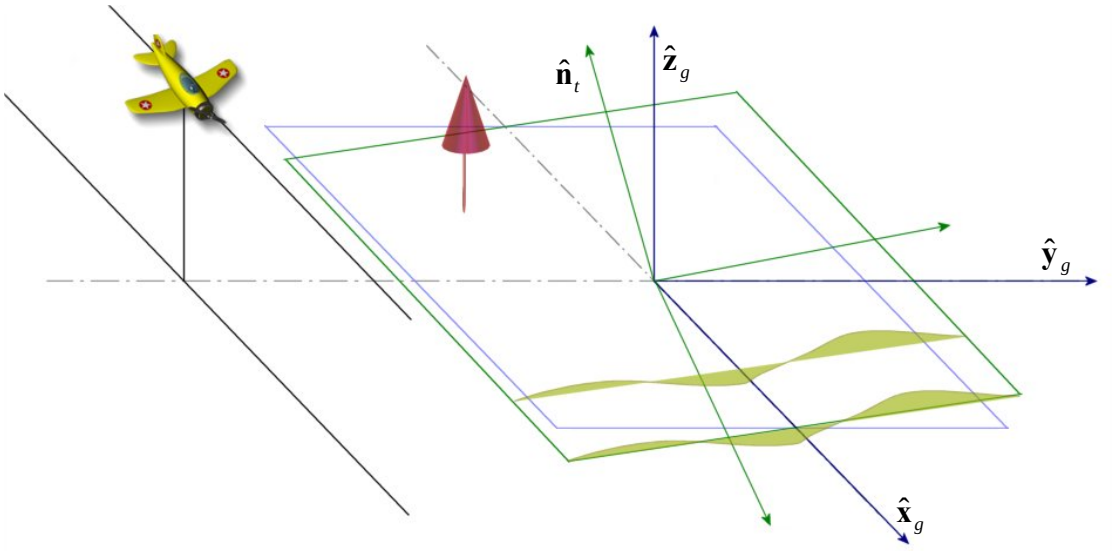


Figure 2.1 Terrain and Imaging Geometry

2.1 Terrain and Imaging Geometry

2.1.2 In the SAR simulation we imagine the imaging radar travelling in the positive *x*-direction, with the antenna pointing portside. The SAR imaging plane is the global $x_g y_g$-plane ($z_g = 0$). The image area lies between $-L_x/2.0 \le x_g \le L_x/2.0$ and $-L_y/2.0 \le y_g \le L_y/2.0$, where x_g is the azimuth coordinate, and y_g is the ground range coordinate, which increases away from the platform trajectory. The mean terrain height at the scene centre is zero.

2.1.3 The forest stand will occupy a circular area. In addition to the tree species, the mean height of the stand, the stem density (trees/hectare) and the area of the forest stand will be supplied by the user. The scene area will be calculated to comfortably accommodate the forest stand, taking into account layover and shadowing areas based on the user-supplied imaging geometry.

2.1.4 Slant range and global incidence angle are defined with reference to the (horizontal) SAR image plane. The platform trajectory is assumed uniform and parallel to the SAR image plane.

2.1.5 The large-scale ground surface, $g(x_g, y_g)$, is expressed as displacements about the mean terrain surface which is flat and tilted. The mean terrain surface has slope s_x in the x_g direction and s_y in the y_g direction. The

ground surface height displacement about the mean terrain surface is periodic in x_g and y_g.

2.1.6 Imagery will be output in the azimuth and ground range space, with far range appearing at the top of the images, as from the point of view of the radar.

3. DEM Generation

3.1 Introduction

3.1.1 This section describes the technique used for the generation of the large-scale digital elevation model. Arguments are given for the 2D case first, and extended to 3D with examples of generated surfaces.

3.2 Theory

3.2.1 Let's describe the desired surface realization as

$$h(x) = \frac{1}{L}\sum_{n=-N}^{N} h_n \exp(i2\pi nx / L) \tag{3.2.1}$$

3.2.2 In (1) L is the length of the surface to be generated, and the surface is periodic with length L such that $h(x+nL) = h(x)$. Now writing $x' = 2\pi x / L$ we have

$$h(x') = \frac{1}{L}\sum_{n=-N}^{N} h_n \exp(inx'), \tag{3.2.2}$$

$$h(x' + 2n\pi) = h(x'), \text{ and} \tag{3.2.3}$$

$$h_m = \frac{L}{2\pi}\int_0^{2\pi} h(x')\exp(-imx')dx' \equiv H(k_m = 2m\pi / L) \tag{3.2.4}$$

3.2.3 The underlying surface correlation function is

$$c_h(s) = \int h(x)h(x+s)dx \tag{3.2.5}$$

3.2.4 Suppose we have a random function $f(x)$ with a very narrow correlation function. A realisation of such a function may be generated with a suitable random number generator of which there are many. Let's write the Fourier transform (*FT*) of our desired function as

$$H(k) = W(k)F(k),\ F(k) = FT\left[f(x)\right] \text{ etc.} \tag{3.2.6}$$

3.2.5 The desired correlation can then be written

$$\begin{aligned} c_h(s) &= \int h(x)h(x+s)dx \\ &= FT^{-1}\left[|H(k)|^2\right] \\ &= FT^{-1}\left[|W(k)|^2|F(k)|^2\right] \end{aligned} \tag{3.2.7}$$

3.2.6 In like manner the correlation of our random function is expressed as

$$c_f(s) = FT^{-1}\left[\,|F(k)|^2\right] \tag{3.2.8}$$

3.2.7 This correlation is very narrow, and in the limit of completely random noise we expect $c_f(s) \to \delta(s)$, when we have, after suitable normalisation,

$$C_h(k) = FT\left[c_h(s)\right] = |W(k)|^2 \tag{3.2.9}$$

3.2.8 from which it follows that

$$H(k) = \left[C_h(k)\right]^{1/2} F(k) \tag{3.2.10}$$

3.2.9 or in the discrete model

$$H(k_m) = \left[C_h(k_m)\right]^{1/2} F(k_m), \tag{3.2.11a}$$

$$H(k_m = 2m\pi / L) = \frac{L}{2\pi}\int_0^{2\pi} h(x')\exp(-imx')dx' = h_m \tag{3.2.11b}$$

3.2.10 Now the slopes on the mean flat terrain surface are s_x and s_y such that for any point in the SAR image plane $\mathbf{r} = (x, y, 0)$, the height of the mean flat surface is given as $xs_x + ys_y$. The mean terrain surface has surface normal vector:

$$\hat{\mathbf{n}}_t = (\hat{\mathbf{z}}_g - s_x\hat{\mathbf{x}}_g - s_y\hat{\mathbf{y}}_g)/(1 + s_x^2 + s_y^2)^{1/2} \tag{3.2.12}$$

3.2.11 which makes an angle θ_t with the global z-direction $\hat{\mathbf{z}}_g$ such that

$$\cos\theta_t = \hat{\mathbf{n}}_t.\hat{\mathbf{z}}_g = 1/(1 + s_x^2 + s_y^2)^{1/2} \tag{3.2.13}$$

3.2.12 The displacements $h(x, y)$ calculated using the technique described in the following text are in the direction $\hat{\mathbf{n}}_t$ away from the mean surface at $\mathbf{t} = (x, y, xs_x + ys_y)$. To convert to the global frame make the simple *approximation* that the displacement h in the direction $\hat{\mathbf{n}}_t$ corresponds to a displacement in the global *z*-direction $\hat{\mathbf{z}}_g$ of $h \times (1 + s_x^2 + s_y^2)^{1/2}$. Thus the final height of the ground surface at azimuth *x* and ground range *y* is:

$$g(x, y) = xs_x + ys_y + h(x, y)(1 + s_x^2 + s_y^2)^{1/2} \tag{3.2.14}$$

3.3 Algorithm

3.3.1 The process of constructing the realisation of the surface proceeds as follows (now in 3D). In the frame of the local sloping terrain:

3.3.2 Create a random function $f(x, y)$.

3.3.3 Determine the coefficients,

$$F(k_m, k_n) = \int_0^{2\pi}\int_0^{2\pi} f(x', y')\exp(-imx')\exp(-imy')dx'dy' = f_{mn}, \quad (3.3.15)$$

3.3.4 where the lengths are increased by a factor $a = (1 + s_x^2 + s_y^2)^{1/2}$. The overall scaling can wait until the end. The integrals may be approximated as discrete sums for convenience:

$$F(k_{xm}, k_{yn}) \approx \sum_{i=0}^{N}\sum_{j=0}^{N} f(x_i', y_j')\exp(-imx_i')\exp(-iny_j') = f_{mn} \quad (3.3.16)$$

3.3.5 where $x_i' = i2\pi\Delta x / L_x$ and $y_j' = j2\pi\Delta y / L_y$, and all lengths are in the local terrain frame. Note here that $-N \le m, n \le N$ and $k_{xm} = 2m\pi / L_x$, whilst $k_{yn} = 2n\pi / L_y$.

3.3.6 Determine the coefficients, $\left[C_h(k_{xm}, k_{yn})\right]^{1/2} = c_{mn}$. In our Gaussian correlation example below we have

$$\left[C_h(k_{xm}, k_{yn})\right]^{1/2} = \frac{l^2}{2}\exp(-k_{xm}^2 l^2 / 8)\exp(-k_{yn}^2 l^2 / 8) \quad (3.3.17)$$

3.3.7 Form the products, $h_{mn} = c_{mn} f_{mn}$.

3.3.8 Form the surface using,

$$h(x, y) = \sum_{m=-N}^{N}\sum_{n=-N}^{N} h_{mn}\exp(ik_{xm}x)\exp(ik_{yn}y) \quad (3.3.18)$$

3.3.9 Zero the mean height ($h(x, y) \to h(x, y) - \bar{h}$) and normalize the standard deviation to the desired value ($h(x, y) \to \alpha h(x, y)$).

3.3.10 Finally the heights $h(x, y)$ may be interpreted as displacements about a tilted ground plane, and converted to global SAR frame (see section 2 above) displacements using

$$g(x_m, y_n) = x_m s_x + y_n s_y + h(x_m, y_m)(1 + s_x^2 + s_y^2)^{1/2}. \quad (3.3.19)$$

3.4 Example

3.4.1 Choose:

$$c_h(s) = \exp(-s^2 / l^2) \tag{3.4.20}$$

3.4.2 Then using

$$C_h(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} c_h(s) \exp(iks)\, ds \tag{3.4.21}$$

3.4.3 Find

$$\left[C_h(k)\right]^{1/2} = \frac{l}{\sqrt{2}} \exp(-k^2 l^2 / 8) \tag{3.4.22}$$

3.4.4 Let's consider imaging an area with projection in the SAR image plane 100m in azimuth by 100m in ground range. Suppose that ground range resolution is equal to azimuth resolution, which in turn is 1.5m (specified as width at half power). Let's sample at two thirds of the resolution, i.e. with pixels separated in the SAR image plane by 1m in both azimuth and ground range. As a convenience I like to arrange for the number of pixels to be odd, so that we have a pixel at the scene centre. So first calculate the number of pixels, e.g. in azimuth, as $n_x = L_x / (f_x \sigma_x) = 100$ where $\sigma_x = 1.5m$ is the resolution and $f_x = 2/3$ is the sampling rate. Now we have an even number of pixels so we can force this to become odd by employing the transformation $n_x \to 2(n_x / 2) + 1$, where the integer part only of $(n/2)$ is used. With this transformation we have $n_x = n_y = 101$, and our pixel spacing is $\Delta x = \Delta y = 0.9901m$.

3.4.5 Suppose that we have opted for a small (2%) slope in the positive ground range direction, $s_y = 0.02$. The height scaling factor for the transformation to the local mean terrain system is $a = (1 + s_x^2 + s_y^2)^{1/2} = 1.0002$, lengths in azimuth are unaltered, whilst those in ground range are scaled by the same amount as heights. Thus samples in the local terrain surface are separated by distances $\Delta x' = (1 + s_x^2)^{1/2} \Delta x = \Delta x$ and $\Delta y' = (1 + s_y^2)^{1/2} \Delta y = 1.0002 \Delta y = 0.9903m$.

3.4.6 We may now proceed to calculate the displacements as described in the algorithm, on a grid such that $h(x_m, y_n) = h(m\Delta x', n\Delta y')$, and then normalise and convert to the global SAR frame as indicated.

3.4.7 Note that our algorithm yields a periodic set of displacements. To calculate the ground height outside of the SAR image area:

$$g(x + nL_x, y + mL_y) = g(x, y) + nL_x s_x + mL_y s_y \tag{3.4.23}$$

3.5 DEM Generation Summary

3.5.1 The following parameters are required to generate the large-scale DEM:

3.5.2 The circular forest stand area in square metres, A, under direct control of the user. The image area dimensions in metres, L_x and L_y, are generated from this area using knowledge of the imaging geometry and stand height.

3.5.3 Image dimensions in pixels, n_x and n_y, under the indirect control of the user, resulting from specified resolutions and image dimensions in metres.

3.5.4 Terrain slopes, s_x and s_y, under control of the user.

3.5.5 Large scale surface correlation length in metres, l, under indirect control of the user.

3.5.6 Larg-scale surface height standard deviation in metres, σ_l, under the indirect control of the user.

3.5.7 Number of terms in Fourier expansion, N, specified by the programmer.

3.5.8 Where a parameter is considered to be under *direct* user control, it is anticipated that the user will choose a numerical value.

3.5.9 A parameter choice under *indirect* control will imply a value resulting either from the user's choice of other parameter values, or from the user specifying a *type* of choice.

3.5.10 For example. If we ask the user to specify the image dimensions in metres, and the sensor resolutions in metres, then the image dimensions in pixels may be calculated using a suitable sampling rate specified by the programmer.

3.5.11 As another example, we might ask the user to choose between surface roughness on the basis of "very smooth", "smooth", "rough" or "very rough" *types*. Roughness depends upon wavelength and incidence angle. Thus the user's choice of surface roughness *type* would translate, in a hidden fashion, into a numerical value for surface height standard deviation in metres. The user does not need to know this value. The manner of choice, i.e. the algorithm for translating the roughness type into a numerical parameter value will be refined during the development process.

3.6 Recovering Ground Heights

3.2.1 Ground heights are stored at the center of grid squares. To generate surface facets requires the ground heights on the corners of the grid squares. To recover these heights we use a linear interpolation. For points within the

periodic surface at coordinates (x, y) first find the integer coordinates of the offset into the ground height map according to

i = (int)((x + (Lx - deltax)/2.0)/deltax); (3.2.1)

j = (int)((-y + (Ly - deltay)/2.0)/deltay); (3.2.2)

3.2.2 Note that in the image space small iy indexes correspond to large ground range, i.e. $iy = 0$ and $y = Ly/2$ at the top of the image.

3.2.3 For any point at which we choose to recover the ground height we choose 4 points in the grid space such that (ix, iy) corresponds to the upper left corner:

(i, j) $(i+1, j)$ (x_i, y_j) $(x_i + dx, y_j)$

$(i, j+1)$ $(i+1, j+1)$ $(x_i, y_j - dy)$ $(x_i + dx, y_j - dy)$

3.2.4 We then expand the height function linearly about the top left corner and write:

$$z(\delta x, \delta y) = a_0 + a_1\delta x + a_2\delta y + a_3\delta x\delta y \quad (3.2.3)$$

3.2.5 where

$$-dx/2 \le \delta x \le +3dx/2 \quad (3.2.4)$$

3.2.6 and

$$+dy/2 \ge \delta y \ge -3dy/2\,. \quad (3.2.5)$$

3.2.7 Numbering the corners in order, starting from the top left as zero, and working around clockwise we recover 4 known heights as:

$$z_0 = a_0 \qquad a_0 = z_0 \quad (3.2.6)$$

$$z_1 = a_0 + dxa_1 \qquad a_1 = (z_1 - z_0)/dx \quad (3.2.7)$$

$$z_2 = a_0 + dxa_1 - dya_2 - dxdya_3 \qquad a_2 = (z_0 - z_3)/dy \quad (3.2.8)$$

$$z_3 = a_0 - dya_2 \qquad a_3 = (z_1 + z_3 - z_2 - z_0)/(dxdy) \quad (3.2.9)$$

3.2.8 writing

$$x_i = i\Delta x - (Lx - \Delta x)/2 \qquad \delta x = x - x_i \quad (3.2.10)$$

$$y_j = -(j\Delta y - (Ly - \Delta y)/2) \qquad \delta y = y - y_j \quad (3.2.11)$$

3.2.9 Then the height is recovered for any point in the periodic region.

4. Tree Location Map Generation

4.1 Introduction

4.1.1 This section describes the generation of the tree location map. The tree location map consists of a list of azimuth and ground range coordinates for the base point of each tree. This point is the stem base for a tree with a single stem.

4.1.2 For each coordinate pair there will be an associated tree height. The tree height will be drawn from a normal distribution. The user will directly specify the mean tree height in metres. The tree height standard deviation will be under programmer control, and typically set to 5% of the mean value.

4.1.3 The tree height is a fundamental quantity. All allometric equations will be fundtions of tree species (specified by the user) and tree height. Thus the tree global crown radius (see later definition of the Tree "object"), will be calculated from tree height using a species-specific allometric equation. This crown radius will be used in the generation of tree locations.

4.1.4 The process of tree location map generation begins by determining the number of trees in the scene, and then initializing them in a regular pattern, and realizing their heights and global crown radii. The trees are subsequently "shuffled" around in random, collision-avoiding walks using Monte Carlo techniques, to reach a more realistic distribution of tree positions.

4.2 Algorithm Details

4.2.1 The user has provided the area dimensions, tree species, and mean height of the forest stand. From the last two quantities we derive the mean crown radius using the encoded allometric equation.

4.2.2 Given the area and the mean crown radius we calculate the maximum number of trees that we could expect within that area if they are close-packed in 2-dimensions (with discs of mean crown radius occupying a fraction 0.907 of the total area).

4.2.3 We permit the user to specify stand density, perhaps in terms of fractional crown cover, or as a *type*, or even simply in stems/hectare. If the requested stand density is greater than the calculated maximum by a certain amount the actual stand density is the smaller of the two, otherwise the actual stand density is the requested density.

4.2.4 From the actual stem density in stems/hectare we recover the stem line density in stems/100m as the square root of the stem density. Initially trees are to be located on grid coordinates, and the grid dimensions are now

recovered using the area dimensions and the stem line density. Note that since the final number of trees is the product of two integers, so the requested stem density will not be matched precisely, only closely.

4.2.5 Having determined the grid dimensions the trees are located on grid points with alternate lines shifted by half a grid spacing to offset the initial tree positions. Tree locations are then “shuffled” using an annealing Monte Carlo algorithm.

4.2.6 The shuffling is controlled using the crown radii and a cost function, which rises steeply as the trees approach to a separation within their combined radii, and has a small attractive effect for greater tree separations. The technique yields realistic, non-uniform distributions of tree locations, which appear quite natural.

4.2.7 The shuffling treats the image area as periodic, so that trees shuffling out of the area to the right, shuffle into the area from the left, and similarly from top to bottom, and vice versa. Exploiting periodicity permits the expansion of the simulated area, whilst preserving a natural tree separation distribution, and avoiding trees that are too close together.

4.2.8 Finally, from the full list of tree positions, a circular area is excised that corresponds to the forest stand area requested by the user. This circular area is offset slightly in ground-range in order to accommodate the extent of layover. The tree locations, heights and crown radii will be output to a text file.

4.2.9 The scene geometry accommodates layover and shadow regions:

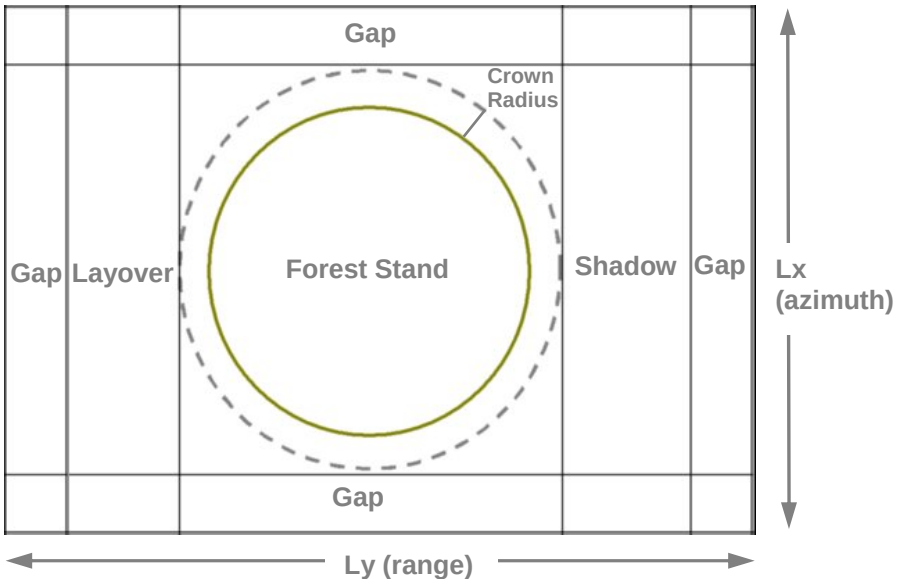


Figure 4.1 Scene geometry

4.3 Tree Location Map Summary

4.3.1 The following parameters are required to generate the tree location map:

4.3.2 Forest stand area, *A*, under the direct control of the user.

4.3.3 The mean tree height in metres, under the direct control of the user.

4.3.4 The requested stand density, specified either directly or indirectly by the user.

4.4 Special Note on the HEDGE Option

4.4.1 In keeping with the previous version of PolSARPro, it will be possible for the user to specify a single **HEDGE** model of the canopy, rather than a forest stand. In this case no shuffling will be required. Choosing the **HEDGE** option will result in a single, cylindrical tree-crown at scene center, with no stems, primary or secondary branches, but only tertiary branches and leaves. Details of this option are given in later sections.

5. Tree Species and Architecture

5.1 Introduction

5.1.1 To complete the description of the forest stand we require a description of the tree architectures. Each tree, when realized, will appear unique: with its own arrangement of stem, primary and secondary branches, and a crown volume filled with tertiary branch elements and leaves.

5.1.2 The description of the tree then resides in the procedures and algorithms used to generate these realizations. A broad description of these algorithms, and the structures upon which they operate, is given in the following sections. Actual parameter values controlling their operation will be refined during the development process to ensure that predicted SAR backscattering coefficients are within observational limits.

5.1.3 Time constraints permit only two species options in the initial version of **PolSARProSim**. These are the hedge model (**HEDGE** option) and a tree similar either to a Scots Pine (**PINE001/2/3** option) or a broad-leaved tree (**DECIDUOUS001** option). Future versions of the software may include all options.

5.2 Trees as Objects

5.2.1 Although programming in C it is useful to think, in object-oriented terms, of trees represented in the software as objects. In practice trees will be represented as records, such as the following prototype definition:

```
typdef  struct tree_object_tag {
 int            species;        * Species of tree                          */
 d3Vector       base;           /* Stem base position                       */
 double         height;         /* Length of stem in the stem direction     */
 double         radius;         /* The nominal maximum crown radius         */
 Branch_List    Stem;           /* List of tree trunks, can be more than one */
 Branch_List    Primary;        /* List of primary branches                 */
 Branch_List    Secondary;      /* List of secondary branches               */
 Crown_List     CrownVolume;    /* List of volumes occupied by tertiary     */
                                /* branches and leaves.                     */
} Tree;
```

5.2.2 The definition of a tree record, as anticipated in the final code, already contains some familiar fields.

5.2.3 For example an integer field represents the species, a vector the location, and double precision floating point numbers (“doubles”) the tree height and radius in metres. These quantities were defined and used in the tree shuffling procedure.

5.2.4 In addition the record contains lists of stems, primary and secondary branch objects, whose definitions are considered shortly. The definition of

a tree record also contains a list of crown volumes occupied by tertiary branches and leaves (or needles in the case of the **PINE001** option).

5.2.5 For the **HEDGE** option we will have only a single tree, with a single crown, represented by a finite cylinder with axis vertical in the global frame, truncated by planes parallel to the mean ground surface. The Stem_List, Primary_Branch_List and Secondary_Branch_List of a **HEDGE** option Tree will be empty, whilst the crown list will have a single entry. Crowns, Branches and Cylinders will be discussed in due course. The hedge will be extended from earlier versions and will be comprised of both deciduous leaves and branches.

5.2.6 Option **PINE001** will resemble Scots Pine. These trees will have two crowns: an upper living crown, and a lower dry crown without green vegetation. Primary branch structure will be predominantly radial, with branch angle varying with height. The living crowns of **PINE001** option Trees will be occupied by tertiary branches and needles. Other options for Scots Pine (**PINE002** and **PINE003**) having different crown shapes, may be implemented in the future.

5.2.7 Option **DECIDUOUS001** does not resemble anything in particular at this stage. This species is experimental and may not be implemented in the final code. The idea is to create these trees first by defining a stem, and a large crown. Primary branches will then be created from the stem to points within the crown. Secondary branches may then be created from primaries and associated crowns. The crowns of **DECIDUOUS001** trees will also be occupied by tertiary branches and deciduous leaves.

5.2.8 Delivery of **HEDGE** and one of **PINE001**, **PINE002**, **PINE003** or **DECIDUOUS001** will be included in the present contract.

5.3 Branches as Objects

5.3.1 In practice the tapering branch will be converted to a series of cylinders for the purposes of the scattering calculation. This permits us the luxury of approximating curved branches. However with this option comes the challenge of their implementation, and their creation. This topic turns out to be quite involved, so let's first look at how shaped branches may be described and then go on to define the Branch object structure.

5.3.1 Branch Shape Definition

5.3.1.1 The branch shape, in the absence of tropism (photo/gravi), is defined in terms of the location of the centre of the branch:

$$\mathbf{b}(t) = \mathbf{b}_0 + l\left\{(1-t)\left[\int_0^t C_x(u)\,du\right]\hat{\mathbf{x}}_0 + (1-t)\left[\int_0^t C_y(u)\,du\right]\hat{\mathbf{y}}_0 + t\hat{\mathbf{z}}_0\right\} \qquad (5.3.1)$$

5.3.1.2 where the two curvature functions are:

$$C_x(t) = \gamma\left[\sin(2\pi\, t/\lambda_{cx} + \phi_x) + \sin(2\pi\, t/(\alpha_g \lambda_{cx}) + \phi_{ax})\right] \tag{5.3.2}$$

$$C_y(t) = \gamma\left[\sin(2\pi\, t/\lambda_{cy} + \phi_y) + \sin(2\pi\, t/(\alpha_g \lambda_{cy}) + \phi_{ay})\right] \tag{5.3.3}$$

5.3.1.2 Here $\alpha_g = 1.61803399$ is the golden ratio, the phase angles $\phi_{x,y}$ and $\phi_{\alpha x,y}$ are random on $[0,2\pi]$, l is the "straight" branch length, γ is an overall amplitude scaling and $\lambda_{cx,y}$ is a wavelength of the crookedness which is large compared with 1 for gently undulating branches and small for rapidly undulating branches.

5.3.1.3 The values for $\lambda_{cx,y}$ and γ are species dependent and defined as fractions of the "straight" stem length.

5.3.1.4 Here t is the parametric value, which is zero at the start of the branch and unity at the tip of the branch. The (1-*t*) scaling on the crookedness correction ensures that the crookedness correction is zero at the branch tip. These functions define the shape of the branch in the absence of child branches and tropisms.

5.3.1.5 The unit vector $\hat{\mathbf{z}}_0$ defines the initial direction of the branch. In the absence of tropisms and crookedness the branch is straight and maintains its initial direction and $\mathbf{b}(t) = \mathbf{b}_0 + lt\hat{\mathbf{z}}_0$.

5.3.1.6 Let's introduce a tropism, first in the absence of crookedness. The effect of a tropism is to bend the straight branch to the tropism direction so that it increasingly points that way as the parametric value approaches unity. In other words the branch direction becomes a function of the parameter *t*. The way I have chosen to describe this is

$$\mathbf{b}(t) = \mathbf{b}_0 + l\left[t\,\hat{\mathbf{z}}_0 + \frac{d_p}{2(1-d_p^2)^{1/2}} t^2\,\hat{\mathbf{p}} + \mathbf{c}(t)\right] \tag{5.3.4}$$

$$\begin{aligned}\mathbf{b}(t) &= \mathbf{b}_0 + l\left[t\,\hat{\mathbf{z}}_0 + p(t)\,\hat{\mathbf{p}} + \mathbf{c}(t)\right] \\ &= \mathbf{b}_0 + lt\,\hat{\mathbf{z}}_0 + l\int_0^t p'(\tau)d\tau\,\hat{\mathbf{p}} \\ &\quad + l(1-t)\left[\int_0^t C_x(u)\,du\right]\hat{\mathbf{x}}(t) + l(1-t)\left[\int_0^t C_y(u)\,du\right]\hat{\mathbf{y}}(t)\end{aligned} \tag{5.3.5}$$

5.3.1.7 Taking the crookedness correction to be minor, the approximate direction of the branch is

$$\hat{\mathbf{z}}(t) = \frac{\hat{\mathbf{z}}_0 + p'(t)\,\hat{\mathbf{p}}}{[1 + p'^2(t)]^{1/2}} \tag{5.3.6}$$

5.3.1.8 A simple, quadratic form for the tropism function is adopted such that

$$\hat{\mathbf{z}}(t) = \frac{(1-d_p^2)^{1/2}\hat{\mathbf{z}}_0 + td_p\,\hat{\mathbf{p}}}{[1+(t^2-1)d_p^2]^{1/2}} \tag{5.3.7}$$

5.3.1.9 where $0 \le d_p < 1$ is a kind of "tropism factor" (species and branching order dependent) controlling the amount of tropism in the tropism direction $\hat{\mathbf{p}}$.

5.3.1.10 The orthogonal vectors are defined using the current direction and the global z-direction as

$$\hat{\mathbf{x}}(t) = \frac{\hat{\mathbf{z}}_g \times \hat{\mathbf{z}}(t)}{|\,\hat{\mathbf{z}}_g \times \hat{\mathbf{z}}(t)\,|} \quad \mathbf{y}(t) = \hat{\mathbf{x}}(t) \times \hat{\mathbf{z}}(t) \tag{5.3.8}$$

5.3.1.11 In principle the tropism factor and direction are dependent upon both species and branching order. However, to keep things as simple as possible, the only tropism to be coded will be a phototropism in the positive global z-direction. Tertiary branches will be modelled as straight.

5.3.1.12 We are now in a position to establish our branch structure:

```
typedef struct branch_object_tag {
 double      length;          /* The "straight" length of the branch, l,  in metres     */
 double      start_radius;    /* The start radius of the branch in metres               */
 double      end_radius;      /* The end radius of the branch in metres                 */
 d3Vector    b0;              /* The branch beginning                                   */
 d3Vector    z0;              /* The initial branch direction                           */
 double      dp;              /* The tropism coefficient                                */
 double      phix;            /* A crookedness random angle                             */
 double      phiy;            /* A crookedness random angle                             */
 double      phicx;           /* A crookedness random angle                             */
double       phicy;           /* A crookedness random angle                             */
 double      lamdacx;         /* A crookedness random wavelength                        */
 double      lamdacy;         /* A crookedness random wavelength                        */
 double      gamma;           /* The crookedness strength                               */
 double      moisture;        /* Fractional moisture content of the branch              */
 Complex     permittivity;    /* The effective dielectric permittivity of the branch    */
} Branch;
```

5.3.1.13 Parameter values will be drawn from species-specific distributions. For example the start radius will depend upon the radius of the parent branch at the branching point etc. The branch radius will decay linearly from the start to end values with the parameter *t*.

5.3.1.14 Branches will be subdivided into segments (Cylinders) in the code for drawing and scattering calculations, and the branch equation will be used to determine the Cylinder properties for each branch segment.

5.3.1.15 A prototype definition of a cylinder object is:

```
typedef struct cylinder_object_tag {
```

```
double      length;         /* The length of the cylinder in metres                */
double      radius;         /* The radius of the cylinder in metres                */
d3Vector    base;           /* A point in the centre of the end of the cylinder    */
d3Vector    axis;           /* A unit vector in the cylinder axial direction       */
d3Vector    x;              /* A unit vector normal to the axial direction         */
d3Vector    y;              /* A unit vector normal to both axis and x             */
Complex     permittivity;   /* The effective dielectric permittivity of the cylinder */
} Cylinder;
```

5.4 Crowns as Objects

5.4.1 All crown objects will relate to volumes of type cylinder, cone or truncated ellipsoid:

```
typedef struct crown_tag {
int         shape;  /* Type of crown shape (Cone, Spheroid or Cylinder)          */
double      beta;   /* The angle between the crown axis and surface              */
double      d1;     /* The semi-major axis of the spheroid or cone/cylinder length */
double      d2;     /* The semi-minor axis of the spheroid or cone/cylinder base radius */
double      d3;     /* The truncation length from spheroid tip or cone/cylinder length */
d3Vector    base;   /* A point in the centre of the base of the crown volume     */
d3Vector    axis;   /* A unit vector in the major axial direction                */
d3Vector    x;      /* A unit vector normal to the major axial direction         */
d3Vector    y;      /* A unit vector normal to both axis and x                   */
double      sx;     /* The slope in the global x direction of a bounding plane   */
double      sy;     /* The slope in the global y direction of a bounding plane   */
} Crown;
```

5.4.2 Crowns are to be populated by leaves and twigs (tertiary branches) and these require dimensions, permittivities, orientations and volume fractions. These will all be hardwired by species, so that we need only define the crown type, which is defined by the species of the associated Tree (crowns never appearing in isolation).

5.4.3 Finally note that crowns may be truncated by bounding planes. For example in the case of the HEDGE, the crown is a vertical cylinder terminated at each end by a flat surface parallel to the mean sloping terrain. A conical crown is terminated at the wide end by a similar plane surface, and an spheroidal crown may be truncated, again using a similar surface. A Plane object may be defined in the code according to:

```
typedef struct plane_tag {
d3Vector    p0;     /* A known point in the plane                  */
double      sx;     /* The slope of the plane in the global x direction */
double      sy;     /* The slope of the plane in the global y direction */
d3Vector    np;     /* A unit vector normal to the plane           */
d3Vector    xp;     /* A unit vector in the plane                  */
d3Vector    yp;     /* A unit vector in the plane normal to xp and np */
} Plane;
```

5.4.4 Note that since Crown bounding surfaces are always taken parallel to the mean sloping terrain and known to pass through specific points they are not currently stored explicitly as part of Crown structures. Instead Crown structures store the slopes of the bounding planes, and Plane objects are constructed from Crown field data as required.

5.4.5 As it will be necessary to understand the intersection of rays with planar and crown surfaces in order to implement the estimation of attenuation when performing the SAR calculation. So in addition to a planar surface a Ray object is also defined, according to:

```
typedef struct ray_tag {
 d3Vector    s0;     /* The starting point of the ray                    */
 double      theta;  /* The polar angle of the ray direction             */
 double      phi;    /* The azimuth angle of the ray direction           */
 d3Vector    a;      /* The unit vector in the ray direction             */
} Ray;
```

5.4.6 The details of Ray-Crown and Ray-Plane intersections are given in an appendix.

5.5 Realization of Trees

5.5.1 The **HEDGE** will, as in the previous data included with **PolSARPro v1.0**, have no stems, primary or secondary branches, and will consist of a single cylindrical living crown centred at the origin. The base of the cylinder will be at the local ground height.

5.5.2 A **HEDGE** Tree has the following assignments:

```
Tree.species        =    0;
Tree.base           =     (0.0, 0.0, ground_height (0.0, 0.0));
Tree.height         =    user specified mean tree height;
Tree.radius;        =     minimum of Lx/4.0, Ly/4.0 or
                         HEDGE_RADIUS_FACTOR*Tree.height;
Tree.Stem           =    empty Branch_List;
Tree.Primary        =    empty Branch_List;
Tree.Secondary      =    empty Branch_List;
Tree.CrownVolume    =    list with a single cylindrical crown defined as:

Crown.shape         =    CROWN_SHAPE_CYLINDER;
Crown.beta          =    atan2 (Tree.radius, Tree.height);
Crown.d1            =    Tree.height;
Crown.d2            =    Tree.radius;
Crown.d3            =    Tree.height
Crown.base          =    Tree.base;
Crown.axis          =    zg, the global up direction;
Crown.x             =    xg, the global x-direction;
Crown.y             =    yg, the global y-direction.
```

5.5.3 Note that tertiary elements are not included in the realization. These are only realized on demand for the SAR calculation, and are otherwise not called into existence. So the creation of the Tree record for the **HEDGE** is particularly straightforward.

5.5.4 The **PINE001** model has a single, near-vertical stem. There are two crowns: an upper living crown with an **SPHEROIDAL** shape, truncated in half, and a lower dry crown, with a **CYLINDRICAL** shape having the same radius as the upper crown maximum radius. If **PINE001** trees are

tall enough then the terminating planes may be made horizontal, otherwise they will again be parallel with the mean sloping terrain, although this has consequences for matching the radius of the cylindrical dry crown layer to the upper living crown spheroid. So a **PINE001** Tree will have the following assignments:

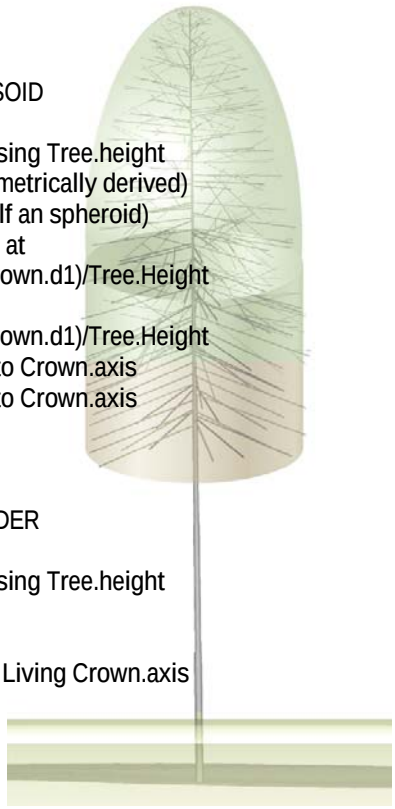
Tree.species	=	1;
Tree.base	=	(0.0, 0.0, ground_height (0.0, 0.0));
Tree.height	=	user specified mean tree height;
Tree.radius;	=	Allometric distribution using Tree.height;
Tree.Stem	=	list with a single stem;
Tree.Primary	=	list of dry and living branches radiating from the stem and terminating at points on the crown surfaces;
Tree.Secondary	=	list of living branches radiating from primaries and terminating on points on sub-crown surfaces;
Tree.CrownVolume	=	list containing two crowns: a cylindrical dry crown and a spheroidal living crown defined as follows:

Spheroidal living crown:

Crown.shape	=	CROWN_SHAPE_ELLIPSOID
Crown.beta	=	atan2 (d2, d1)
Crown.d1	=	Allometric distribution using Tree.height
Crown.d2	=	Tree.radius (already allometrically derived)
Crown.d3	=	Crown.d1 (i.e. exactly half an spheroid)
Crown.base	=	Tree.Stem branch center at t = (Tree.Height – Crown.d1)/Tree.Height
Crown.axis	=	Tree.Stem direction at t = (Tree.Height – Crown.d1)/Tree.Height
Crown.x	=	x-direction , orthogonal to Crown.axis
Crown.y	=	y-direction , orthogonal to Crown.axis

Cylindrical dry crown:

Crown.shape	=	CROWN_SHAPE_CYLINDER
Crown.beta	=	atan2 (d2, d1)
Crown.d1	=	Allometric distribution using Tree.height
Crown.d2	=	Living Crown.d2
Crown.d3	=	d1
Crown.base	=	Living Crown.base –d1 * Living Crown.axis
Crown.axis	=	Living Crown axis
Crown.x	=	Living Crown.x
Crown.y	=	Living crown.y



5.5.5 A possible **DECIDUOUS001** model has a single, highly curved stem and one living crown. Primaries are large but few in number, almost like stems themselves, and generated using the large global crown. Secondaries are created starting from the primary and reaching the crown surface. The crown will be populated with tertiary elements as required. The crown is spheroidal. Thus a **DECIDUOUS001** Tree will have the following assignments:

Tree.species	=	2;
Tree.base	=	(0.0, 0.0, ground_height (0.0, 0.0));
Tree.height	=	user specified mean tree height;
Tree.radius;	=	recovered from DECIDUOUS allometric distribution using Tree.height: global tree dimension;
Tree.Stem	=	list with a single stem;
Tree.Primary	=	list living branches radiating from the stem and terminating at points on the global crown surface;
Tree.Secondary	=	list of living branches radiating from primaries and Terminating on points on sub-crown surfaces;
Tree.CrownVolume	=	list containing spheroidal crown;

5.5.6 The crown associated with the stem in the **DECIDUOUS001** model has the following assignments:

Crown.shape	=	CROWN_SHAPE_ELLIPSOID;
Crown.beta	=	atan2 (d2, d1);
Crown.d1	=	Allometric based;
Crown.d2	=	Allometric based;
Crown.d3	=	2 * d1 (a full spheroid)
Crown.base	=	Asscoiated branch position at t = (Stem length –d3)/Stem length;
Crown.axis	=	Associated primary direction at t = (Stem length –d3)/Stem length;
Crown.x	=	x-direction , orthogonal to this Crown.axis;
Crown.y	=	x-direction , orthogonal to this Crown.axis;

5.5.7 Together with species definitions, ground generation, tree location generation and short vegetation description, this completes the model of the forest.

5.6 Generating Stems and Branches

5.6.1 The definition of a branch is based around an equation describing the curved branch centre. This equation has the form (repeated here for convenience):

$$\mathbf{b}(t) = \mathbf{b}_0 + l\left[t\,\hat{\mathbf{z}}_0 + \frac{d_p}{2(1-d_p^2)^{1/2}} t^2\,\hat{\mathbf{p}} + \mathbf{c}(t) \right] \tag{5.6.9}$$

5.6.2 where the minor “crookedness correction” is given as

$$\mathbf{c}(t) = (1-t)\left[\int_0^t C_x(u)\,du \right]\hat{\mathbf{x}}(t) + (1-t)\left[\int_0^t C_y(u)\,du \right]\hat{\mathbf{y}}(t) \tag{5.6.10}$$

5.6.3 wherein the crookedness functions are

$$C_\gamma(u) = \gamma\left[\sin(2\pi\,u/\lambda_{c\gamma} + \phi_\gamma) + \sin(2\pi\,u/(\alpha_g\lambda_{c\gamma}) + \phi_{a\gamma})\right] \;,\; \gamma = x, y \tag{5.6.11}$$

5.6.4 The non-crooked “direction” of the branch is

$$\hat{\mathbf{z}}(t) = \frac{(1-d_p^2)^{1/2}\hat{\mathbf{z}}_0 + t d_p\,\hat{\mathbf{p}}}{[1+(t^2-1)d_p^2]^{1/2}} \tag{5.6.12}$$

5.6.5 where $0 \le d_p < 1$ is a kind of "tropism factor" (species and branching order dependent) controlling the amount of tropism in the tropism direction $\hat{\mathbf{p}}$ and

$$\hat{\mathbf{x}}(t) = \frac{\hat{\mathbf{z}}_g \times \hat{\mathbf{z}}(t)}{|\,\hat{\mathbf{z}}_g \times \hat{\mathbf{z}}(t)\,|}, \qquad \mathbf{y}(t) = \hat{\mathbf{x}}(t) \times \hat{\mathbf{z}}(t) \tag{5.6.13}$$

5.6.6 The crookedness functions may be integrated to yield

$$\begin{aligned}\int_0^t C_x(u)\,du &= \gamma \int_0^t \left[\sin(2\pi\, u/\lambda_{c\gamma} + \phi_\gamma) + \sin(2\pi\, u/(\alpha_g \lambda_{c\gamma}) + \phi_{a\gamma})\right] du \\ &= -\frac{\gamma \lambda_{c\gamma}}{2\pi}\left[\cos(2\pi\, t/\lambda_{c\gamma} + \phi_\gamma) - \cos(\phi_\gamma)\right] \\ &\quad -\frac{\gamma \alpha_g \lambda_{c\gamma}}{2\pi}\left[\cos(2\pi\, t/\alpha_g\lambda_{c\gamma} + \phi_{a\gamma}) - \cos(\phi_{a\gamma})\right]\end{aligned} \tag{5.6.14}$$

5.6.7 which takes care of this component which we imagine as a correction to the central branch position, and parameter values are chosen appropriately (species and branching order dependencies are involved for the wavelengths and overall scaling, but the angles are uniformly random on $[0,2\pi]$).

5.6.8 The simplest way to find the remaining parameter, l, is to specify the branch termination on the crown volume surface, i.e.

$$\mathbf{b}(1) = \mathbf{b}_0 + l\left[\hat{\mathbf{z}}_0 + \frac{d_p}{2(1-d_p^2)^{1/2}}\,\hat{\mathbf{p}}\right] = \mathbf{b}_0 + l\left[\hat{\mathbf{z}}_0 + f_p\,\hat{\mathbf{p}}\right] \tag{5.6.15}$$

5.6.9 This point may be found by starting at the point $\mathbf{b}_0$, and searching for the intersection of a ray with the crown volume in the direction $\hat{\mathbf{a}} = (\hat{\mathbf{z}}_0 + f_p\,\hat{\mathbf{p}})/|\,\hat{\mathbf{z}}_0 + f_p\,\hat{\mathbf{p}}\,|$ to find a point, $\mathbf{b}_1$, from which the "straight length" l (so called since it would be equal to the length of the branch in the absence of tropism and crookedness) may be recovered as

$$l = \frac{(\mathbf{b}_1 - \mathbf{b}_0).\hat{\mathbf{z}}_0}{\left[1 + f_p\,(\hat{\mathbf{p}}.\hat{\mathbf{z}}_0)\right]} \qquad f_p = \frac{d_p}{2(1-d_p^2)^{1/2}} \tag{5.6.16}$$

5.6.10 If we are able, and we are, to specify the base, initial direction and tropism vectors at the outset, then the branch length may be found by seeking the intersection of the ray with the canopy surface.

5.6.11 The same process will be adopted for stems, primaries and secondary branches. Thus, given the appropriate rules, and techniques for finding ray intersections with crown volume surfaces, we have all that is necessary to construct the Trees.

5.6.12 The algorithms for finding ray intersections with different geometric objects are given in Appendix A.

6. Interferometric SAR Image Calculation

6.1 Introduction

6.1.1 It might appear to be most efficient to calculate both images at the same time, using a “dual-antenna” approach rather than a “repeat-pass” approach. This would have the advantage of only requiring realization of the forest a single time. However there are a number of overheads associated with this idea.

6.1.2 The first and most obvious is the memory requirement. Not only is this doubled for the images themselves but other structures, such as the attenuation grid, to be discussed in the following sections, must be reproduced.

6.1.3 Nor is there a great saving in computational load. Whilst the “repeat-pass” method doubles the realization cost, and requires care and attention to exact reproduction of the scene, this computational load is not a critical factor. From experience the heaviest computational load arises in the calculation of effective permittivity by voxel, scattering amplitude calculations using sophisticated models for cylinder scattering, and determination of focus points, particularly for the ground-volume interactions. All of these overheads are present in both the “dual-antenna” and “repeat-pass” approaches.

6.1.4 Thus the approach to be adopted is “repeat-pass”, which in some respects presents a less complex coding challenge, with a greater concomitant potential for success. This will in effect involve calling a single routine twice, each time with a slightly different imaging geometry.

6.1.5 The SAR image calculation phase may be broken down into stages. The first of these is the calculation of canopy tree crown and understorey permittivities. Crowns will be taken to have the same mean permittivity, and to be statistically isotropic. Thus tree crown effective permittivities will be calculated in the Foldy-Lax approximation by averaging the forward scattering amplitudes of their constituent elements. The same approach will be adopted for the short vegetation layer, which will also be taken to be homogeneous and isotropic, and have constant depth above local ground height.

6.1.6 The next stage will be to construct attenuation grids, which will supply look-up tables for attenuation by spatial location. The attenuation grids will be used when calculating scattering to provide approximate estimates of attenuation by location in a very efficient way that will enhance the simulation performance, yet still maintain the effects of the inhomogeneity of the tree crown distribution.

6.1.7 Having determined the attenuation grids the next stage is the calculation of SAR images. This will start with the direct-ground (DG) calculation, which will employ the techniques described in [ref] and reproduced in the following sections. Essentially the ground surface is divided into numerous facets, and these are assigned complex scattering amplitudes, and focused in the image one after the other, each being scaled by the attenuation estimate for its location in the scene.

6.1.8 The next calculations, direct-volume (DV) and ground-volume (GV) for the short vegetation layer are conducted together to reduce computational load. The technique uses the hybrid stochastic-deterministic approach described in [ref]. A subset of plant elements is realized, and their scattering amplitudes scaled to yield the backscattering coefficient anticipated by the full sample. The number of realizations per resolution cell is sufficient to provide fully-developed speckle, and the actual realizations are preserved for the second interferometric SAR image. Performing the DG and GV calculations together saves on overheads, although this does mean that the separate images will not be available. However, for the purposes of tutorial education in PolInSAR techniques this is not an issue.

6.1.9 The final calculation will be conducted for the trees. Each tree location will be addressed in turn. A unique distribution of stem, primary and secondary branches will be created and subdivided into short cylindrical elements. Both DV and GV terms will be calculated for these elements. Finally the hybrid technique adopted for short vegetation will be adopted for scattering from the smaller tree crown elements of leaves, twigs and pine needles.

6.1.10 After writing out the first set of SAR imagery to an appropriately named file, the whole process is repeated for the repeat-pass scenario, taking care to preserve random number sequences where appropriate and ensure consistency of scene realization.

6.1.11 The full process is described in greater detail in the following sections.

6.2 Representation of the SAR Image

6.2.1 The forward SAR model will employ a representation of the SAR image as a coherent (phase-preserving) transform of a radar reflectivity function. This model is derived in [ref], and the derivation is repeated here for convenience.

6.2.2 The SAR image is taken to be formed using transmission and reception of regular pulses of the form:

$$\mathbf{p}(\tau) = \int_{-\infty}^{\infty} \hat{p}(\omega) \exp(-i\omega\tau)\, d\omega\, \hat{\mathbf{e}} = p(\tau)\, \hat{\mathbf{e}} \tag{6.2.1}$$

6.2.3 The received signal when the platform is at "position" x is

$$\boldsymbol{\sigma}(\tau, x) = \int_{-\infty}^{\infty} \underline{\mathbf{B}}(\omega, x) . \hat{\mathbf{e}}(x)\, \hat{p}(\omega) \exp(-i\omega\tau)\, d\omega\,, \tag{6.2.2}$$

6.2.4 i.e. the total return from the illuminated scene, where

$$\underline{\mathbf{B}}(\omega, x) = \int \underline{\mathbf{F}}(\omega, x, \mathbf{s}) \exp(i\omega 2r(x, \mathbf{s})/c)\, d\mathbf{s} \tag{6.2.3}$$

6.2.5 and $\underline{\mathbf{F}}(\omega, x, \mathbf{s})$ is the reflectivity density at the point $\mathbf{s}$ with range $r(x, \mathbf{s})$. The correlation of the received signal with the return anticipated from a delayed pulse is:

$$\begin{aligned} \underline{\mathbf{I}}(t, y) &= \iint \boldsymbol{\sigma}(\tau, x)\; p^*(\tau - \tau'(x, t, y))\, d\tau\, dx \\ &= \iint \boldsymbol{\sigma}(\tau, x)\; p^*(\tau - 2\,|\,\mathbf{r}_p(x) - \mathbf{r}(t, y)\,|\,/c)\, d\tau\, dx \end{aligned} \tag{6.2.4}$$

6.2.6 Here $\mathbf{r}_p(x)$ is the platform location at x, and $\mathbf{r}(t, y)$ is the location of the scattering centre at the point of interest in the image. Using the definition of the pulse in terms of its Fourier spectrum

$$p^*(\tau - \tau') = \int_{-\infty}^{\infty} \hat{p}^*(\omega') \exp(i\omega'(\tau - \tau'))\, d\omega' \tag{6.2.5}$$

6.2.7 in the expression for the correlated image leads to

$$\underline{\mathbf{I}}(t, y) = \int \int_{-\infty}^{\infty} \underline{\mathbf{B}}(\omega, x)\,|\,\hat{p}(\omega)\,|^2 \exp(-i\omega\tau'(x, t, y))\, d\omega\, dx \tag{6.2.6}$$

6.2.8 Substituting for $\underline{\mathbf{B}}$ the SAR image function is:

$$\underline{\mathbf{I}}(t, y) = \int \int \int_{-\infty}^{\infty} \left[\underline{\mathbf{F}}(\omega, x, \mathbf{s})\,|\,\hat{p}(\omega)\,|^2\; e^{i\omega[2r(x,\mathbf{s})/c - \tau'(x,t,y)]}\right] d\omega\, dx\, d\mathbf{s} \tag{6.2.7}$$

6.2.9 The approximation of constant reflectivity across the SAR bandwidth and aperture is adopted so that

$$\underline{\mathbf{I}}(t, y) = \int \underline{\mathbf{F}}(\mathbf{s})\, \hat{Q}(t - r(\mathbf{s}), y - x)\, d\mathbf{s}\,, \tag{6.2.8}$$

6.2.10 and $\hat{Q} = \int \int_{-\infty}^{\infty} |\,\hat{p}(\omega)\,|^2\; e^{i\omega[2r(x,\mathbf{s})/c - \tau'(x,t,y)]}\, d\omega\, dx$ is the system impulse response or point spread function. This quantity will be approximated in the simulation using a Gaussian, or similar, product in azimuth and ground range, with appropriate range phase.

6.2.11 This result is valid for direct scattering from a surface or volume scattering, and higher orders of scattering involving surface-surface, or volume-surface, or surface-volume-surface etc. The point, **s**, need not be a location either in or on a scattering object, but rather in general it is an *effective* scattering centre associated with the reflectivity determined from interacting scene elements that are spatially separated: examples follow in the next sections.

6.2.12 A discrete approximation will be employed:

$$\underline{\mathbf{I}}(r', x') = \sum_n \underline{\mathbf{F}}_n \hat{Q}(r' - r(\mathbf{s}_n), x' - x(\mathbf{s}_n)) \tag{6.2.9}$$

6.2.13 where the polarimetric scattering amplitude, $\underline{\mathbf{F}}_n$, associated with the n^{th} scattering event, arising from one or more elements of the scene, and incorporating attenuation, has an *effective* scattering centre, $\mathbf{s}_n$.

6.2.14 SAR image prediction using this model will proceed as follows: describe the scene geometrically and physically, as described in the preceding sections, then divide the scene into small (relative to system resolution) elements. Interrogate the scene and determine scattering events (divided by class), calculating their polarimetric scattering amplitudes, and associated *effective* scattering centres. Sum their contributions *coherently* into the image accumulator, $\underline{\mathbf{I}}(r', x')$.

6.2.15 Whilst simple in concept the procedure entails a heavy computational load mitigated only by the use of approximations that preserve important features of the SAR imagery such as clutter distribution, shadowing and interferometric coherence.

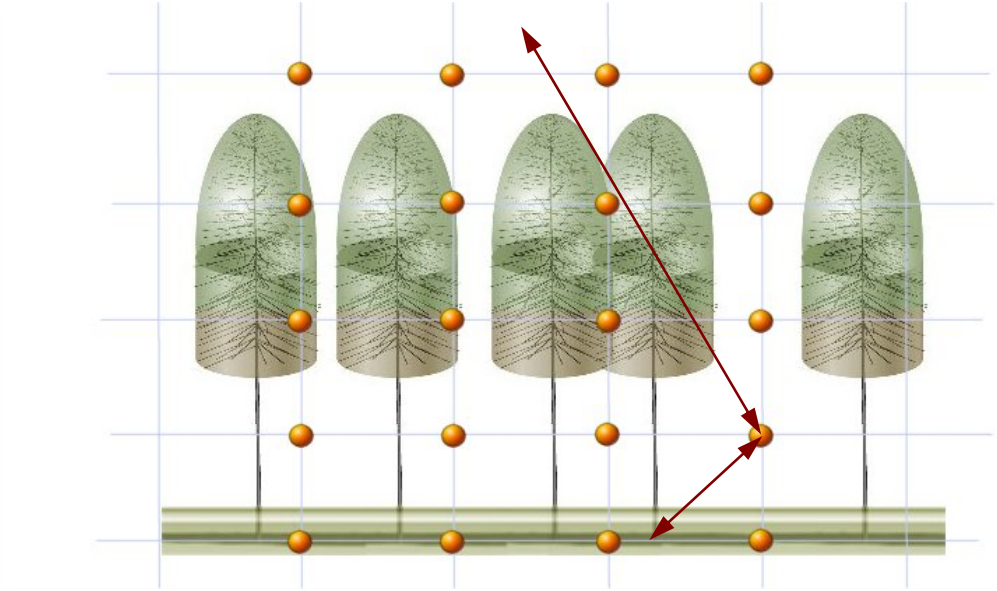
6.3 Modelling Attenuation

6.3.1 This requires estimation of the complex permittivity of the random media comprising the tree crowns and the short vegetation. In previous work of this kind the technique was to divide the canopy into voxels and to identify the material occupying those voxels. The scattering properties of occupying vegetation elements were used to estimate the mean, or effective permittivity in each voxel. Then during the SAR calculation, for each scattering event a list was built indicating through which voxels the broadside ray passed for that event, and the attenuation due to each voxel was calculated and accumulated into the scattering amplitude.

6.3.2 This process is somewhat inefficient and therefore unsuitable to the present task. The intention is to adopt a new, more efficient method. The first stage is to estimate an effective permittivity for both the short vegetation layer, and the tree crowns. Each crown will have the same mean permittivity, which will be assumed isotropic. The permittivity estimate will be accomplished by integrating the forward scattering amplitudes of plant elements and interpreting them as effective polarisabilities. Thus the

approximation will correspond to the Foldy-Lax model, which is widely used and appropriate for the wavelengths under consideration.

6.3.3 Having determined the mean dielectric properties of crowns and short vegetation a 3D grid of points will be generated to sample the attenuation function. One grid is required for the incident and scattering directions, and another for the ground reflection direction. For each grid point the broadside ray paths are determined and the attenuation associated with the path determined by finding ray intersections with the boundaries of the understorey layer and the crowns. Transmission coefficients are generally close to unity and may not be required, however if calculated it will be assumed that the transmission of the ray through the interface between air and crown is at normal incidence. In reality of course our crown boundary is completely artificial, as trees do not have well-defined crown volumes, which in itself is good reason for omitting transmission boundary effects.



6.3.4 Figure 6.3.1. Mapping of attenuation into the 3D space occupied by the forest canopy. Direct and reflected ray paths are determined for each grid point. The passage of each ray through crowns and short vegetation is used to determine the attenuation associated with each path.

6.3.5 During the SAR image calculation the grid will act as a look-up table for attenuating scattering amplitudes. A scattering amplitude will be assigned the same attenuation as that associated with the nearest grid point to the scatterer centre. This will provide efficiency and preserve the effects of inhomogeneity.

6.4 Direct-Ground (DG) SAR Image

6.4.1 The ground surface will be divided into flat, rough facets, derived from the large-scale DEM. In principle each facet has a unique realisation of surface roughness drawn from an underlying statistical distribution. The polarimetric scattering amplitude of each facet is therefore unique: the variation leading to speckle in the DG term under the right conditions. A

fully deterministic approach to the calculation of DG terms is computationally prohibitive. Thus facet scattering amplitudes will be calculated using the covariance matrix associated with the underlying surface scattering model calculated in the local frame of the facet.

6.4.2 In past work [ref] each facet was assigned a complex, polarimetric scattering amplitude drawn from the multivariate Gaussian distribution:

$$p(\mathbf{S}) = \exp(-\mathbf{S}^{*t}\underline{\mathbf{C}}^{-1}\mathbf{S})/(\pi^3 \det \underline{\mathbf{C}}) \quad (6.4.1)$$

6.4.3 sampled using Monte Carlo (MC) techniques.

6.4.4 The model employed is two-scale, with the small perturbation model (SPM) used to model the small-scale response. The copolar channels are perfectly correlated in the SPM, and even with large-scale roughness effects this can lead to unrealistically low polarimetric entropy in the DG term. Thus in practice the covariance matrix approach is adopted using a high (0.95) HH-VV correlation, although this requires careful sampling.

6.4.5 Sampling eqn. (6.41) at high HH-VV coherence is problematic, and an alternative approach will be tested for this study. Since in the SPM HH and VV are perfectly correlated, so we may choose a single, local scattering amplitude factor for each facet, using a single complex gaussian distribution. To afford a more realistic polarimetric entropy, the local scattering amplitude matrix recovered this way will be rotated about the radar line-of-sight by a random angle chosen from a uniform distribution the extent of which is designed to achieve the desired HH-VV coherence in the local frame of the facet [ref]. The final local scattering amplitude will then be converted into the radar frame.

6.4.6 The hybrid deterministic-stochastic technique may be made to preserve interferometric coherence by preserving the random number sequence employed in the MC sampling. In practice any change in incidence angle, however slight, may alter the course of the MC sampling on some facets as the local facet incidence alters the local mean RCS and covariance. This can have a small effect upon coherence for the occasional image pixel. This effect may be minimised by preserving the scaled facet scattering amplitude factors between tracks of the interferometric baseline. Even without this precaution the technique yields a high mean ground coherence as desired.

6.5 Direct-Vegetation (DV) SAR Image

6.5.1 Deterministic Model

6.5.1.1 Volume terms for large scatterers, i.e. stems, primary and secondary branches are calculated in deterministic fashion. Branches are divided into short segments, and scattering will be modelled using an approximate but efficient form for the truncated infinite cylinder approximation.

6.5.2 Hybrid Stochastic Model

6.5.2.1 The numerous small plant elements in each crown and in the short vegetation layer are treated using a hybrid approach. When calculating scattering terms the understorey is divided into areas smaller than a resolution cell, and each is populated with a number of random realisations of occupying scattering elements. Crowns are treated in a similar fashion: distributing realizations throughout the crown evenly.

6.5.2.2 Scattering amplitudes are scaled if the number of plant elements that would actually be found in each are or volume is greater than the number of realizations. Realisations will be preserved between calculations for each track on the interferometric baseline: in this fashion computational efficiency is maintained whilst preserving both backscattering coefficient, and coherence behaviour with baseline.

6.5.2.3 Scattering by small elements will be calculated using the Rayleigh-Gans approach [ref], whilst the scattering due to very small objects, such as Pine needles, at low frequency will, if not significant, be ignored. These considerations will serve to make the calculation computationally efficient.

6.6 Ground-Vegetation (GV) SAR Image

6.6.1 These terms will be calculated as each scattering object is realised, reducing the computational load as many of the terms required in the DV calculation are used in the GV calculation. The calculation of “effective scattering centre” is a heavy computational burden if we require account to be taken of local slope. Thus the approximation will be adopted that the reflection is from the mean terrain surface, and is modulated by the mean roughness. Thus the “effective” scattering centre will be found easily for all scatterers as the projection onto the mean terrain surface of the scattering centre [ref].

6.6.2 Both the deterministic and hybrid stochastic approaches will be adopted for the GV calculation in the manners previously described in the DV section. The ground-volume-ground (GVG) interaction will not be considered as it is generally found not to be significant. Note that whilst DV terms display layover, the same is not true for GV terms. If included GVG terms would display layover away from the radar, and appear to have phase centres below the soil surface. In the shadow region of the forest stand the GVG term may thus appear dominant and lead to confusing tree height inversions. To avoid this, and the associated high costs of its computation, the GVG term will be omitted.

6.7 Summary of Input Parameters

6.7.1 The following parameters are required for SAR image generation:

6.7.2 Frequency: the SAR sensor center frequency in Gigahertz (between L-band and P-band)

6.7.3 Tracks: the number of tracks (minimum of 2) required for the interferometric SAR simulation.

6.7.4 Slant ranges: the broadside slant range to scene center (in metres) for each of the specified tracks.

6.7.5 Incidence angles: the global incidence angle at scene center (in degrees) for each of the specified tracks.

6.7.6 Azimuth_resolution: the desired final SAR cross range resolution (in metres) specified as the width at half height power of the system point spread function in azimuth.

6.7.7 Azimuth sampling frequency: specified as a fraction of the azimuth resolution, nominally two-thirds. Together with azimuth resolution and the length in azimuth of the area to be imaged this will be used to determine the image dimension in azimuth.

6.7.8 Slant range resolution: the desired final SAR slant range resolution (in metres) specified as the width at half height power of the system point spread function in slant range. Note that this is assumed constant for each simulated track, as is the system center frequency. Note also that the ground range resolution thus varies from track to track, as calculated in the simulation.

6.7.9 Ground range sampling frequency: specified as a fraction of ground range resolution. Together with ground range resolution and the length in ground range of the area to be imaged this will be used to determine the image dimension in ground range.

6.7.10 In addition there are a number of bio-physical parameters required in order to determine the scattering properties of the scene, such as branch and leaf moisture contents and soil moisture. Of these only soil moisture and surface roughness models will be requested from the user.

7. File Formats and Calling Convention

7.1 Introduction

7.1.1 There are a number of files required for the correct operation of the program and their formats are described in this document.

7.2 Input Text File

7.2.1 The input parameter file will be an ASCII text file with the three letter extension “sar”, e.g. “run001.sar”. Input text files will have the following format:

```
2             /* The number of requested tracks                              */
4242.640687   /* Slant range in metres                                       */
45.0          /* Incidence angle in degrees                                  */
4242.640687   /* Slant range in metres                                       */
46.0          /* Incidence angle in degrees                                  */
1.24          /* Centre frequency in GHz                                     */
1.5           /* Azimuth resolution (width at half-height power) in metres   */
1.5           /* Slant range resolution (width at half-height power) in metres */
0             /* DEM model: 0 = perfectly smooth … 10 = very rough           */
0.02          /* Ground slope in azimuth direction (dimensionless)           */
0.01          /* Ground slope in ground range direction (dimensionless)      */
222           /* Random number generator seed                                */
0             /* Tree species: 0 = HEDGE, 1,2,3 = PINE, 4 = DECIDUOUS        */
18.0          /* Mean tree height in metres                                  */
2827.433388   /* Area of the forest stand in square metres                   */
80            /* Desired stand density in stems per hectare                  */
1             /* Ground moisture content model: 0 = dry ... 10  = wet        */
```

Figure 7.2.1 An example of the ASCII input file format

7.2.2 Each line of the input file consists of an initial value (one per line) followed by optional text describing the parameter associated with the value.

7.2.3 Each line of text is terminated by a line feed (LF, ASCII 10, 0x0A) character, or a carriage return character (CR, ASCII 13, 0x0D) or both.

7.2.4 The text input file contains sixteen (16) lines, one for each parameter described in the following:

7.2.5 The number of requested tracks. This is an int and always has the value two (2) for calls to **PolSARProSim** from within **PolSARPro v2.0**.

7.2.6 The next four lines describe the tracks to be simulated for the interferometric pair of SAR images. Each track is parameterised using a slant range (in metres) followed by an incidence angle (in degrees). Slant range values and incidence angle values are input as type double.

7.2.7 The centre frequency follows as type double and is given in Gigahertz.

7.2.8 The azimuth resolution follows as type double and is given in metres. NOTE the definition of resolution for the purposes of **PolSARProSim** is width of the point spread function at half height in power.

7.2.9 The slant-range resolution follows as type double and is given in metres. As with azimuth resolution this is defined as the width of the point spread function at half height in power. The slant-range resolution is held fixed for each track, which corresponds to constant system bandwidth.

7.2.10 There then follows an int parameter used to define the ground surface properties. This parameter has the minimum value zero (0), corresponding to a perfectly smooth large-scale surface, and the maximum value ten (10), for a very rough large-scale surface. This value is translated into actual values of surface height standard deviation and correlation length within PolSARProSim.

7.2.11 Ground slope in azimuth (type double) is recorded on the next line of the input file: this is a dimensionless quantity describing the slope of the mean underlying surface in the azimuth direction.

7.2.12 Ground slope in range (type double) is recorded on the next line of the input file: again this is a dimensionless quantity describing the slope of the mean underlying surface in the azimuth direction.

7.2.13 The next line is an int value used to seed the random number generator.

7.2.14 The next line is an int value used to define the species of tree occupying the forest. Possible values are listed in Figure 7.1. Not all of these will be operational as only two operational values are to be implemented in this version of PolSARProSim. These will include the value zero (0), representing the **HEDGE**, and one of the values between one (1) and (4). The final choice will depend upon time contraints of the implementation stage, and performance of the final model.

7.2.15 The next three lines are used to describe the forest stand. The first is a double representing the mean height of trees in metres.

7.2.16 The next double is the area of the forest stand in square metres.

7.2.17 The next int is the forest stand density given as the number of trees per hectare.

7.2.18 Finally an int value is used to describe the requested soil moisture level.

7.2.19 Note that the SAR image area and final image dimensions will all be determined using these parameters and other “hard-wired” values such as sampling rate etc.

7.2.20 Note also that these values are “requested” values. The actual values used within the simulation may not correspond exactly to these values. However the internal parameter values will be close to those requested, and the actual values used will be output into a text file with the extension “.out”.

7.2.21 The output text file will be stored in the requested directory. This brings us to the format for calling the simulation from within **PolSARPro**. The format of the call to the simulation will be

7.2.22 **`PolSARProSim prefix directory`**

7.2.23 That is to say the executable will be called with name **PolSARProSim** and will be passed two string variables on the command line. The first variable will be the input/output filename prefix, and the second the pathname of the directory where **PolSARProSim** is to seek input and to write output. For example the call

7.2.24 **`PolSARProSim psps C:\Workspace`**

7.2.25 will cause the simulation to read input from the file with name “psps.sar” in the directory “C:\Workspace”, and to write text output to the file “psps.out” in the same directory. Other forms of output will include a graphic file with a simplified representation of the simulated forest stand as viewed from the platform at broadside, and the SAR images themselves. The format of these files is discussed in the following sections.

7.3 Graphic File Format

7.3.1 Following the input of parameters and construction of the forest scene a small graphic image will be calculated and output as a guide to the visual interpretation of the resulting SAR images.

7.3.2 Graphic image files will have the extension “gri”. A single image will be output to the working directory with the name “forest_image.gri”.

7.3.3 Procedures for reading and writing the graphic image files will be provided as part of **PolSARProSim** and may be called from within **PolSARPro**.

7.3.4 Graphics files with the extension “.gri” are binary files with the following format:

long	: np, the total number of pixels in the image
int	: nx, the number of pixels in the “x”-direction
int	: ny, the number of pixels in the “y”-direction
int	: n, the number of char bytes in the filename
n×char	: the filename
int	: Ninfo, the number of bytes of ancillary text information
Ninfo×char	: the ancillary text information
n×graphic_pixel	: the image data

Figure 7.3.1. The graphic image file format

7.3.5 The type graphic_pixel is defined as follows:

```
typedef struct graphic_pixel_tab  {
        unsigned char     red;
        unsigned char     green;
        unsigned char     blue;
} graphic_pixel;
```

Figure 7.3.2. The graphic_pixel record definition

7.3.6 The image data is written as a single block with equivalent statement:

7.3.7 fwrite (pGI, sizeof (graphic_pixel), np, pGF);

7.3.8 where in the above pGI is of type graphic_pixel* and pGF is of type FILE*.

7.4 SAR Image File Format

7.4.1 Single-Look-Complex (SLC) SAR images will be output from **PolSARProSim** to files with the extension ".sim" in the following format:

```
double          : dx, the pixel dimension in the "x"-direction
double          : dy, the pixel dimension in the "y"-direction
int             : n, the number of char bytes in the filename
n×char          : the filename
double          : Lx, the SAR image dimension in the "x"-direction
double          : Ly, the SAR image dimension in the "y"-direction
int             : Ninfo, the number of bytes of ancillary text information
long            : np, the total number of pixels in the image
int             : nx, the number of pixels in the "x"-direction
int             : ny, the number of pixels in the "y"-direction
Ninfo×char      : the ancillary text information
int             : pixel type (see figure 7.4.2)
n×sim_pixel     : the image data
```

Figure 7.4.1. The SAR image file format

7.4.2 The type sim_pixel is defined as follows:

```
typedef struct sim_float_complex_tag {
        float x;
        float y;
} SIM_Complex_Float;

typedef struct sim_double_complex_tag {
        double x;
        double y;
} SIM_Complex_Double;

typedef unsigned char           sim_byte;
```

```
typedef unsigned short int          sim_word;
typedef int                         sim_dword;
typedef float                       sim_float;
typedef double                      sim_double;
typedef SIM_Complex_Float           sim_complex_float;
typedef SIM_Complex_Double          sim_complex_double;

typedef union  sim_type_tag {
        sim_byte                    b;
        sim_word                    w;
        sim_dword                   dw;
        sim_float                   f;
        sim_double                  d;
        sim_complex_float           cf;
        sim_complex_double          cd;
} sim_type;

typedef struct simpixel_tag {
        int                         simpixeltype;
        sim_type                    data;
} sim_pixel;
```

Figure 7.4.2. The SAR image sar_pixel type definition.

7.4.3 Procedures for reading and writing the SAR image files will be provided as part of **PolSARProSim** and may be called from within **PolSARPro**.

Appendix A: Finding Ray Intersections

A.1 General

A.1.1 In all cases we define a ray with:

1) start position $\mathbf{p}$,
2) direction $\hat{\mathbf{a}}$, and
3) point of intersection $\mathbf{s} = \mathbf{p} + \alpha\,\hat{\mathbf{a}}$.

A.2 Intersection with a Plane

A.2.1 Let the plane have normal unit vector $\hat{\mathbf{n}}_p$ and pass through the point $\mathbf{p}_0$. Let the unit vectors $\hat{\mathbf{x}}_p$ and $\hat{\mathbf{y}}_p$, lying in the plane, form an orthonormal set with $\hat{\mathbf{n}}_p$. The point of intersection is a solution of:

$$\mathbf{s} = \mathbf{p} + \alpha\,\hat{\mathbf{a}} = \mathbf{p}_0 + A\hat{\mathbf{x}}_p + B\hat{\mathbf{y}}_p$$

A.2.2 From which it follows that if $|\,\hat{\mathbf{a}}.\hat{\mathbf{n}}_p\,| > 0$ then

$$\alpha = (\mathbf{p}_0 - \mathbf{p}).\hat{\mathbf{n}}_p\,/(\hat{\mathbf{a}}.\hat{\mathbf{n}}_p)$$

A.2.3 otherwise there is no intersection. Note that it is possible to find $\alpha < 0$ and the success of the algorithm will depend upon the sign of alpha when looking for branch lengths. Let's define a function that performs this operation. Our procedure with prototype:

```
int   RayPlaneIntersection (Ray *pR, Plane *pP, d3Vector *pS, double *alpha);
```

A.2.4 should return the value NO_RAYPLANE_ERRORS (= 1) if successful, and the value !NO_RAYPLANE_ERRORS if no intersection is to be found. The code will look something like this (in pseudo-code):

```
#define       NO_RAYPLANE_ERRORS  1

int   RayPlaneIntersection      (Ray *pR, Plane *pP, d3Vector *pS, double *alpha)
{
 int          rtn_value         = NO_RAYPLANE_ERRORS;
 double       adotnp            = d3Vector_scalar_product (pR->a, pP->np);
 double       p0mpdotnp;

 if (fabs(adotnp) > FLT_EPSILON) {
  p0mpdotnp  = d3Vector_scalar_product (d3Vector_difference (pP->p0, pR->s0), pP->np);
  *alpha     = p0mpdotnp/adotnp;
  *pS        = d3Vector_sum (pR->s0, d3Vector_scalar_multiply (pR->a, *alpha));
 } else {
  rtn_value  = !NO_RAYPLANE_ERRORS;
```

```
 *alpha        = 0.0;
 *pS           = Zero_d3Vector ();
}
 return (rtn_value);
}
```

A.3 Intersection with a Cylinder

A.3.1 Let the cylinder have axial unit vector $\hat{\mathbf{z}}_c$ passing through the base point $\mathbf{c}_0$. Let the unit vectors $\hat{\mathbf{x}}_c$ and $\hat{\mathbf{y}}_c$, form an orthonormal set with $\hat{\mathbf{z}}_c$. The point of intersection is a solution of:

$$\begin{aligned}\mathbf{s} &= \mathbf{p} + \alpha\,\hat{\mathbf{a}} \\ &= \mathbf{c}_0 + z_s\,\hat{\mathbf{z}}_c + R(\cos\theta_s\hat{\mathbf{x}}_c + \sin\theta_s\hat{\mathbf{y}}_c)\end{aligned} \tag{A.3.1}$$

A.3.2 It follows that

$$\hat{\mathbf{z}}_c.\mathbf{s} = (\hat{\mathbf{z}}_c.\mathbf{p}) + \alpha\,(\hat{\mathbf{z}}_c.\hat{\mathbf{a}}) = (\hat{\mathbf{z}}_c.\mathbf{c}_0) + z_s \tag{A.3.2a}$$

$$\hat{\mathbf{x}}_c.\mathbf{s} = (\hat{\mathbf{x}}_c.\mathbf{p}) + \alpha\,(\hat{\mathbf{x}}_c.\hat{\mathbf{a}}) = (\hat{\mathbf{x}}_c.\mathbf{c}_0) + R\cos\theta_s \tag{A.3.2b}$$

$$\hat{\mathbf{y}}_c.\mathbf{s} = (\hat{\mathbf{y}}_c.\mathbf{p}) + \alpha\,(\hat{\mathbf{y}}_c.\hat{\mathbf{a}}) = (\hat{\mathbf{y}}_c.\mathbf{c}_0) + R\sin\theta_s \tag{A.3.2c}$$

A.3.3 We need only use the last two equations to recover an expression for α as the solution to a quadratic equation:

$$[(\hat{\mathbf{x}}_c.(\mathbf{p}-\mathbf{c}_0)) + \alpha\,(\hat{\mathbf{x}}_c.\hat{\mathbf{a}})]^2 + [(\hat{\mathbf{y}}_c.(\mathbf{p}-\mathbf{c}_0)) + \alpha\,(\hat{\mathbf{y}}_c.\hat{\mathbf{a}})]^2 = R^2 \tag{A.3.3}$$

A.3.4 writing

$$A = \hat{\mathbf{x}}_c.(\mathbf{p}-\mathbf{c}_0) \qquad B = (\hat{\mathbf{x}}_c.\hat{\mathbf{a}}) \qquad C = \hat{\mathbf{y}}_c.(\mathbf{p}-\mathbf{c}_0) \qquad D = (\hat{\mathbf{y}}_c.\hat{\mathbf{a}}) \tag{A.3.4}$$

A.3.5 then

$$\alpha^2(B^2+D^2) + 2(AB+CD)\alpha + (A^2+C^2-R^2) = 0 \tag{A.3.5}$$

A.3.6 so that

$$\alpha = \frac{-(AB+CD) \pm \sqrt{(AB+CD)^2 - (B^2+D^2)(A^2+C^2-R^2)}}{(B^2+D^2)} \tag{A.3.6}$$

A.3.7 Note that if the argument to the square root is negative the ray does not intersect the cylinder. Also if both *B* and *D* are zero then the ray direction is parallel with the cylinder axis and we have either no intersection or an infinity of intersections if the ray lies in the cylinder surface. Otherwise there are two solutions, and the choice of solution will depend upon the application. Let's define a function that performs this operation. Our procedure with prototype:

```
int     RayCylinderIntersection (Ray *pR, Cylinder *pP, d3Vector *pS1, double *alpha1,
                                 d3Vector *pS2, double *alpha2);
```

A.3.8 should return the value NO_RAYCYLINDER_ERRORS (= 2) if successful, and the value !NO_RAYCYLINDER_ERRORS if no intersection is to be found. The code will look something like this:

```
#define  NO_RAYCYLINDER_ERRORS      2

int      RayCylinderIntersection    (Ray *pR, Cylinder *pC, d3Vector *pS1, double *alpha1,
                                     d3Vector *pS2, double *alpha2)
{
 int      rtn_value= NO_RAYCYLINDER_ERRORS;
 double   A        = d3Vector_scalar_product (pC->x, d3Vector_difference (pR->s0, pC->base));
 double   B        = d3Vector_scalar_product (pC->x, pR->a);
 double   C        = d3Vector_scalar_product (pC->y, d3Vector_difference (pR->s0, pC->base));
 double   D        = d3Vector_scalar_product (pC->y, pR->a);
 double   B2D2     = B*B+D*D;
 double   ABCD     = A*B+C*D;
 double   sqrt_arg = ABCD*ABCD-B2D2*(A*A+C*C-pC->radius*pC->radius);

 if (fabs(B2D2) > FLT_EPSILON) {
  if (sqrt_arg > 0.0) {
   *alpha1         =  (sqrt(sqrt_arg)-ABCD)/B2D2;
   *alpha2         = -(sqrt(sqrt_arg)+ABCD)/B2D2;
   *pS1            = d3Vector_sum (pR->s0, d3Vector_scalar_multiply (pR->a, *alpha1));
   *pS2            = d3Vector_sum (pR->s0, d3Vector_scalar_multiply (pR->a, *alpha2));
  } else {
   rtn_value       = !NO_RAYCYLINDER_ERRORS;
  }
 } else {
  rtn_value        = !NO_RAYCYLINDER_ERRORS;
 }
 return (rtn_value);
}
```

A.4 Intersection with a Cone

A.4.1 Let the cone have axial unit vector $\hat{\mathbf{z}}_c$ passing through the base point $\mathbf{c}_0$ to the tip of the cone at $\mathbf{t}_c = \mathbf{c}_0 + h\,\hat{\mathbf{z}}_c$. Let the unit vectors $\hat{\mathbf{x}}_c$ and $\hat{\mathbf{y}}_c$, form an orthonormal set with $\hat{\mathbf{z}}_c$. The point of intersection is a solution of:

$$\begin{aligned}\mathbf{s} &= \mathbf{p} + \alpha\,\hat{\mathbf{a}} \\ &= \mathbf{c}_0 + z_s\,\hat{\mathbf{z}}_c + R(z_s)(\cos\theta_s\hat{\mathbf{x}}_c + \sin\theta_s\hat{\mathbf{y}}_c)\end{aligned}, \qquad 0 \le z_s \le h \qquad \text{(A.4.1)}$$

A.4.2 Note that

$$R(z_s) = (h - z_s)\tan(\beta) \qquad \text{(A.4.2)}$$

A.4.3 It follows that

$$\hat{\mathbf{z}}_c.\mathbf{s} = (\hat{\mathbf{z}}_c.\mathbf{p}) + \alpha\,(\hat{\mathbf{z}}_c.\hat{\mathbf{a}}) = (\hat{\mathbf{z}}_c.\mathbf{c}_0) + z_s \qquad \text{(A.4.3a)}$$

$$\hat{\mathbf{x}}_c.\mathbf{s} = (\hat{\mathbf{x}}_c.\mathbf{p}) + \alpha\,(\hat{\mathbf{x}}_c.\hat{\mathbf{a}}) = (\hat{\mathbf{x}}_c.\mathbf{c}_0) + R(z_s)\cos\theta_s \qquad \text{(A.4.3b)}$$

$$\hat{\mathbf{y}}_c.\mathbf{s} = (\hat{\mathbf{y}}_c.\mathbf{p}) + \alpha\,(\hat{\mathbf{y}}_c.\hat{\mathbf{a}}) = (\hat{\mathbf{y}}_c.\mathbf{c}_0) + R(z_s)\sin\theta_s \tag{A.4.3c}$$

A.4.4 We need to recover an expression for α as the solution to a quadratic equation:

$$[(\hat{\mathbf{x}}_c.(\mathbf{p}-\mathbf{c}_0)) + \alpha\,(\hat{\mathbf{x}}_c.\hat{\mathbf{a}})]^2 + [(\hat{\mathbf{y}}_c.(\mathbf{p}-\mathbf{c}_0)) + \alpha\,(\hat{\mathbf{y}}_c.\hat{\mathbf{a}})]^2 = R(z_s)^2 \tag{A.4.4}$$

$$R^2(z_s) = (h - z_s)^2\tan^2(\beta) \tag{A.4.5}$$

$$z_s = (\hat{\mathbf{z}}_c.(\mathbf{p}-\mathbf{c}_0)) + \alpha\,(\hat{\mathbf{z}}_c.\hat{\mathbf{a}}) \tag{A.4.6}$$

A.4.5 writing

$$A = \hat{\mathbf{x}}_c.(\mathbf{p}-\mathbf{c}_0) \qquad B = (\hat{\mathbf{x}}_c.\hat{\mathbf{a}}) \qquad C = \hat{\mathbf{y}}_c.(\mathbf{p}-\mathbf{c}_0) \tag{A.4.7}$$

$$D = (\hat{\mathbf{y}}_c.\hat{\mathbf{a}}) \qquad E = \hat{\mathbf{z}}_c.(\mathbf{p}-\mathbf{c}_0) \qquad F = (\hat{\mathbf{z}}_c.\hat{\mathbf{a}}) \tag{A.4.8}$$

A.4.6 We have

$$[A + \alpha\,B]^2 + [C + \alpha\,D]^2 = (h - E - \alpha\,F)^2\tan^2(\beta) \tag{A.4.9}$$

A.4.7 which is quadratic in α. Expanding:

$$[A + \alpha\,B]^2 + [C + \alpha\,D]^2 = (h - E - \alpha\,F)^2\tan^2(\beta) \tag{A.4.10}$$

$$\begin{aligned}&\alpha^2(B^2 + D^2) + 2(AB + CD)\alpha + (A^2 + C^2)\\ &= [(h-E) - \alpha\,F]^2\tan^2(\beta)\\ &= [(h-E)^2 + \alpha^2F^2 - \alpha\,2F(h-E)]\tan^2(\beta)\end{aligned} \tag{A.4.11}$$

$$\begin{aligned}\alpha^2(B^2 + D^2 - F^2\tan^2(\beta)) + 2(AB + CD + F(h-E)\tan^2(\beta))\alpha\\ + (A^2 + C^2 - (h-E)^2\tan^2(\beta)) = 0\end{aligned} \tag{A.4.12}$$

$$a\alpha^2 + b\alpha + c = 0 \tag{A.4.13}$$

$$a = B^2 + D^2 - F^2\tan^2(\beta) \tag{A.4.14}$$

$$b = 2(AB + CD + F(h-E)\tan^2(\beta)) \tag{A.4.15}$$

$$c = A^2 + C^2 - (h-E)^2\tan^2(\beta) \tag{A.4.16}$$

A.4.8 So that α has solution

$$\alpha = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{A.4.17}$$

A.4.9 The same caveats apply: if the denominator is zero, or if the argument to the square root is negative then there is no solution, otherwise two solutions will exist and these should not lie beyond the tip of the cone. With our cone object definition we may find the intersections using the following code:

```
#define  NO_RAYCONE_ERRORS   2

int      RayConeIntersection      (Ray *pR, Cone *pC, d3Vector *pS1, double *alpha1,
                                   d3Vector *pS2, double *alpha2)
{
 int      rtn_value= NO_RAYCONE_ERRORS;
 double   A        = d3Vector_scalar_product (pC->x, d3Vector_difference (pR->s0, pC->base));
 double   B        = d3Vector_scalar_product (pC->x, pR->a);
 double   C        = d3Vector_scalar_product (pC->y, d3Vector_difference (pR->s0, pC->base));
 double   D        = d3Vector_scalar_product (pC->y, pR->a);
 double   E        = d3Vector_scalar_product (pC->axis, d3Vector_difference (pR->s0, pC->base));
 double   F        = d3Vector_scalar_product (pC->axis, pR->a);
 double   tan_beta = tan(pC->beta);
 double   tan_beta_2= tan_beta*tan_beta;
 double   a        = B*B+D*D-F*F*tan_beta_2;
 double   b        = 2.0*(A*B+C*D+F*(pC->height-E)*tan_beta_2);
 double   c        = A*A+C*C-(pC->height-E)*(pC->height-E)*tan_beta_2;
 double   sqrt_arg = b*b-4.0*a*c;

 if (fabs(a) > FLT_EPSILON) {
  if (sqrt_arg >= 0.0) {
   *alpha1        =  (sqrt(sqrt_arg)-b)/(2.0*a);
   *alpha2        = -(sqrt(sqrt_arg)+b)/(2.0*a);
   *pS1           = d3Vector_sum (pR->s0, d3Vector_scalar_multiply (pR->a, *alpha1));
   *pS2           = d3Vector_sum (pR->s0, d3Vector_scalar_multiply (pR->a, *alpha2));
  } else {
   rtn_value      = !NO_RAYCONE_ERRORS;
  }
 } else {
  rtn_value       = !NO_RAYCONE_ERRORS;
 }
 return (rtn_value);
}
```

A.5 Intersection with a Spheroid

A.5.1 Let the spheroid have axial unit vector $\hat{\mathbf{z}}_s$ passing through the base point $\mathbf{c}_0$ to the tip of the spheroid at $\mathbf{t}_s = \mathbf{c}_0 + h\,\hat{\mathbf{z}}_s$. Note that the spheroid has semi-axes a_1 (major, $\hat{\mathbf{z}}_s$) direction and a_2 (minor) and that $0 \le h \le 2a_1$. Let the unit vectors $\hat{\mathbf{x}}_s$ and $\hat{\mathbf{y}}_s$, form an orthonormal set with $\hat{\mathbf{z}}_s$. The point of intersection is a solution of:

$$\begin{aligned}\mathbf{s} &= \mathbf{p} + \alpha\,\hat{\mathbf{a}} \\ &= \mathbf{c}_0 + z_s\,\hat{\mathbf{z}}_s + R(z_s)(\cos\theta_s\hat{\mathbf{x}}_s + \sin\theta_s\hat{\mathbf{y}}_s)\end{aligned}, \qquad 0 \le z_s \le h \qquad \text{(A.5.1)}$$

A.5.2 Note that now we have for the spheroid radius

$$R^2(z_s) = \tan^2(\beta)\left[a_1^2 - (a_1 - h + z_s)^2\right], \qquad \tan\beta = a_2 / a_1$$

A.5.3 Following our previous analysis:

$$\hat{\mathbf{z}}_s.\mathbf{s} = (\hat{\mathbf{z}}_c.\mathbf{p}) + \alpha\,(\hat{\mathbf{z}}_s.\hat{\mathbf{a}}) = (\hat{\mathbf{z}}_s.\mathbf{c}_0) + z_s$$

$$\hat{\mathbf{x}}_s.\mathbf{s} = (\hat{\mathbf{x}}_s.\mathbf{p}) + \alpha\,(\hat{\mathbf{x}}_s.\hat{\mathbf{a}}) = (\hat{\mathbf{x}}_s.\mathbf{c}_0) + R(z_s)\cos\theta_s$$
$$\hat{\mathbf{y}}_s.\mathbf{s} = (\hat{\mathbf{y}}_s.\mathbf{p}) + \alpha\,(\hat{\mathbf{y}}_s.\hat{\mathbf{a}}) = (\hat{\mathbf{y}}_s.\mathbf{c}_0) + R(z_s)\sin\theta_s$$

A.5.4 We need to recover an expression for α as the solution to a quadratic equation:

$$[(\hat{\mathbf{x}}_s.(\mathbf{p}-\mathbf{c}_0)) + \alpha\,(\hat{\mathbf{x}}_s.\hat{\mathbf{a}})]^2 + [(\hat{\mathbf{y}}_s.(\mathbf{p}-\mathbf{c}_0)) + \alpha\,(\hat{\mathbf{y}}_s.\hat{\mathbf{a}})]^2 = R(z_s)^2$$
$$z_s = (\hat{\mathbf{z}}_s.(\mathbf{p}-\mathbf{c}_0)) + \alpha\,(\hat{\mathbf{z}}_s.\hat{\mathbf{a}})$$

A.5.5 writing

$$A = \hat{\mathbf{x}}_s.(\mathbf{p}-\mathbf{c}_0) \qquad B = (\hat{\mathbf{x}}_s.\hat{\mathbf{a}}) \qquad C = \hat{\mathbf{y}}_s.(\mathbf{p}-\mathbf{c}_0)$$
$$D = (\hat{\mathbf{y}}_s.\hat{\mathbf{a}}) \qquad E = \hat{\mathbf{z}}_s.(\mathbf{p}-\mathbf{c}_0) \qquad F = (\hat{\mathbf{z}}_s.\hat{\mathbf{a}})$$

A.5.6 We have

$$\alpha^2(B^2 + D^2 + F^2\tan^2(\beta))$$
$$+ 2(AB + CD + F(a_1 - h + E)\tan^2(\beta))\alpha$$
$$+ (A^2 + C^2) - [a_1^2 - (a_1 - h + E)^2]\tan^2(\beta) = 0$$

A.5.7 Which we write as

$$a\alpha^2 + b\alpha + c = 0$$

A.5.8 with

$$a = B^2 + D^2 + F^2\tan^2(\beta)$$
$$b = 2(AB + CD + F(a_1 - h + E)\tan^2(\beta))$$
$$c = (A^2 + C^2) - [a_1^2 - (a_1 - h + E)^2]\tan^2(\beta)$$

A.5.9 So that α again has solution

$$\alpha = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

A.5.10 The same caveats apply: if the denominator is zero, or if the argument to the square root is negative then there is no solution, otherwise two solutions will exist and these should not lie beyond the tip of the cone. With our cone object definition we may find the intersections using the following code:

```
#define  NO_RAYSPHEROID_ERRORS        2

int      RaySpheroidIntersection    (Ray *pR, Spheroid *pS, d3Vector *pS1, double *alpha1,
                                     d3Vector *pS2, double *alpha2)
{
```

```
int     rtn_value           = NO_RAYSPHEROID_ERRORS;
double  A                   = d3Vector_scalar_product (pS->x, d3Vector_difference (pR->s0, pS->base));
double  B                   = d3Vector_scalar_product (pS->x, pR->a);
double  C           = d3Vector_scalar_product (pS->y, d3Vector_difference (pR->s0, pS->base));
double  D           = d3Vector_scalar_product (pS->y, pR->a);
double  E           = d3Vector_scalar_product (pS->axis, d3Vector_difference (pR->s0, pS->base));
double  F           = d3Vector_scalar_product (pS->axis, pR->a);
double  tan_beta = tan(pS->beta);
double  tan_beta_2= tan_beta*tan_beta;
double  h           = pS->h;
double  a1          = pS->a1;
double  a           = B*B+D*D+F*F*tan_beta_2;
double  b           = 2.0*(A*B+C*D+F*(a1-h+E)*tan_beta_2);
double  c           = A*A+C*C-tan_beta_2*(a1*a1-(a1-h+E)*(a1-h+E));
double  sqrt_arg = b*b-4.0*a*c;

if (fabs(a) > FLT_EPSILON) {
 if (sqrt_arg >= 0.0) {
  *alpha1           =  (sqrt(sqrt_arg)-b)/(2.0*a);
  *alpha2           = -(sqrt(sqrt_arg)+b)/(2.0*a);
  *pS1              = d3Vector_sum (pR->s0, d3Vector_scalar_multiply (pR->a, *alpha1));
  *pS2              = d3Vector_sum (pR->s0, d3Vector_scalar_multiply (pR->a, *alpha2));
 } else {
  rtn_value         = !NO_RAYSPHEROID_ERRORS;
 }
} else {
 rtn_value          = !NO_RAYSPHEROID_ERRORS;
}
return (rtn_value);
}
```