Department of Applied Computing

PANGU

PLANET AND ASTEROID NATURAL SCENE GENERATION UTILITY USER MANUAL

DOCUMENT NUMBER	UoD-PANGU-MANUAL
ISSUE	ISSUE 2.70
DATE	21ST December 2006
ESA CONTRACT NO.	17338/03/NL/LvH/bj
ESA TECHNICAL MANAGER	P Harwood
	Dr S. Mancuso

AUTHORS	Dr I. Martin, University of Dundee
	Dr M. Dunstan, University of Dundee
	Dr SM. Parkes, University of Dundee
	Mr D. Matthews, University of Dundee
APPROVED	Dr SM Parkes, University of Dundee

This page has been left blank intentionally.

Document Revision History			
Revision	**Date**	**Responsible**	**Modifications/ Reasons for Change**
Issue 1	May 2002	I Martin	Initial draft release.
Issue 1.01	June 2002	M Dunstan	Modifications following release.
Issue 1.02	August 2002	M Dunstan	Modifications after ESA comments.
Issue 1.50	November 2002	M Dunstan	Updated for the 1.50 release.
Issue 2.00	February 2003	M Dunstan	Updated for the 2.00 release.
Issue 2.50	April 2005	M Dunstan	Updated for the 2.50 release.
Issue 2.60	October 2005	M Dunstan	Updated for the 2.60 release.
Issue 2.65	December 2005	M Dunstan	Updated for the 2.65 release.
Issue 2.70	December 2006	M Dunstan	Updated for the 2.70 release

This page has been left blank intentionally.

TABLE OF CONTENTS

This page has been left blank intentionally.

ACRONYMS AND ABBREVIATIONS

ANSI	American National Standards Institute
DEM	Digital Elevation Model
edtexture	PANGU executable texture manipulator for the viewer
ESA	European Space Agency
FAQ	Frequently Asked Question
FeatureListGen	PANGU executable which creates feature lists (craters or boulders)
GPF	General protection fault (fatal application error under Windows)
GUI	Graphical User Interface
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronic Engineers
imgview	PANGU executable for viewing PPM files generated by viewer
LIDAR	LIght Detection And Ranging
mergeshadows	PANGU executable to support area light sources
mkshadows	PANGU executable offline shadow map generator for the viewer
mktexture	PANGU executable texture generator for the viewer
pan2pov	PANGU executable utility to convert .pan files into POV-Ray scripts
pandump	PANGU executable utility to display brief details about a .pan file
PANGU	Planetary and Asteroid Natural Scene Generation Utility
pangu_client	Linux client for retrieving images from a PANGU server
PC	Personal Computer
PovGen	PANGU executable which initiates POV-Ray to render a scene.
POV-Ray	Persistence of Vision Ray Tracer. A freeware ray-tracing program
RADAR	RAdio Detection And Ranging
ROAM	Real–time Optimally Adapting Meshes (rendering algorithm)
SurfaceGen	PANGU executable which creates and adds features to surface models
UML	Unified Modelling Language
UoD	University of Dundee
viewer	PANGU executable which uses OpenGL to visualise surface models

This page has been left blank intentionally.

1. QUICK START GUIDE

1.1 A NOTE ON FILENAMES

PANGU is designed to be used on both Windows and UNIX systems and this introduces a few problems with file names. For example, file names under Windows are not case-sensitive but they are under UNIX. Also the Windows directory separator character is normally \ while UNIX uses /. Like most Windows applications, PANGU applications will always accept / as a directory name separator and we have tried to use this separator throughout this manual so that the PANGU input files can be used on either platform. The exceptions to this rule are where the text is specific to Windows and where command line examples are given. UNIX users are expected to type / instead of \ in these cases.

Some parts of the text instruct the user to run a `.bat` file: UNIX users should look for a `.sh` file with the same name or type the commands from the .bat file themselves translating \ characters into /.

1.2 A NOTE ON EXAMPLE COMMANDS

Specific command line examples assume that you are working from the `testmodel1` directory of your PANGU installation or a copy of that directory in the same folder.

1.3 INSTALLATION

1.3.1 Windows NT/2000/XP

The Windows installation is distributed as a self-extracting executable called `Pangu_2.70_setup.exe` (or a similar name based on the version number). To install PANGU, run this file from the CD or anywhere on your system. Note: the installer will overwrite files for older versions of PANGU if the older version is in the directory selected for installation. Three dialogs are displayed during installation:

1) A licence agreement displayed which must be accepted for the installation to complete.
2) An information dialog is displayed which describes how to use the additional pre-computed example models which are included in separate directories on the PANGU installation CD. These example models take up approximately 100 MB and so are not included in the basic PANGU installation.
3) The location where PANGU will be installed can be selected. This will be `C:\Program Files\Pangu` unless an alternative location is specified.

PANGU does not currently have a Graphical User Interface and so no icons are added to the systems Start Menu during installation. The PANGU tools can be used either by double clicking the batch files from Windows Explorer or by executing commands in a DOS box.

Two additional pre-computed models, not included in the PANGU installation, have been provided in the Models directory on the PANGU installation CD. To use the extra pre-computed models (`4k_128x128` and `4k_512x512`) copy them from the CD to the PANGU installation directory (by default this will be `C:\Program Files\Pangu`). To allow the batch files to access the PANGU executables, the directories `4k_128x128` and `4k_512x512` must be on the same directory level as the `testmodel1` directory. These directories contain batch files which will view the pre-computed models with different illumination settings. A `README.txt` file is included in each directory which details all the models and batch files included in these additional example directories.

Two MPEG movies of descents onto PANGU surfaces have also been provided on the PANGU installation CD in the `mpegs` directory. One was created using PANGU 1.02 and the other using PANGU 1.50 These can be viewed by double-clicking on the file under Windows.

1.3.2 Windows NT/2000/XP Uninstall

PANGU can be uninstalled by either deleting the PANGU directory or via Add/Remove programs in the system control panel.

1.3.3 Linux

Before starting the installation, decide whether you want a simple user installation (similar to the Windows installation where each user gets their own copy of the test model and all the binaries) or a root installation (the binaries and dynamic libraries are copied into a common directory such as `/usr/local/bin` and `/usr/lib`). For a root installation you must be root when running the installer:

```
su -
```

If your CD-ROM is not mounted then:

```
mount /mnt/cdrom
```

Change the current working directory to the CD ROM and run `lin_install`, for example by:

```
cd /mnt/cdrom
./lin_install
```

Adjust the commands for your particular setup. On some systems you may not be able to run the installer directly from the CD-ROM: instead you will need to create a temporary working directory and copy the relevant files from the CD:

```
mkdir working-tmp
cd working-tmp
cp /mnt/cdrom/*install.sh .
cp /mnt/cdrom/linpan-2_70.tar .
```

Then run `./lin_install` as above. Once the installation has been completed you can remove the `working-tmp` directory and all the files that it contains.

1.4 CONTACT DETAILS

For help with PANGU installation and use please contact:

Iain Martin

imartin@computing.dundee.ac.uk

Tel: +44 (0) 1382 388833

Martin Dunstan

mdunstan@computing.dundee.ac.uk

Tel: +44 (0) 1382 388836

1.5 CONFIDENCE TEST

A confidence test batch file has been provided in the `models/testmodel1` directory. This will test the PANGU installation by generating a hierarchical surface, with craters and boulders and then initiating the viewer to render the surface. To run the confidence test, navigate to the `models/testmodel1` directory and run the `confidence.bat` batch file (Windows users) or the `CONFIDENCE.sh` shell script (Linux users). The tools will generate a large amount of text describing each action during the creation of the model. After the surface model has been created the viewer will be launched to visualise the surface in automatic fly-by mode. If the viewer displays a surface with craters and boulders then the confidence test has been successful.

1.6 NAVIGATING AROUND A SURFACE

Close the viewer if it is still running. Run `viewer.bat` in the `models/testmodel1` directory. This allows navigation around the surface using the keyboard, mouse and position controls.

- Horizontal drags with the left mouse button rotate the camera in the horizontal plane (change the azimuth or yaw angle) or use left and right arrow keys.

- Vertical drags with the left mouse button rotate the camera in the forward vertical plane (change the elevation or pitch) or use up and down arrow keys.
- Horizontal drags with the right mouse button move the camera from side to side without changing the attitude.
- Drags with the right mouse button move the object under the mouse around the screen.
- Use 'z' to zoom in and 'Z' to zoom out. Note: to change the zoom step, close the viewer, adjust the **move_step** parameter in `pangu.ini` in the `testmodel1` directory and re-run the viewer.
- Camera distance, attitude and look-at target can be entered directly using the side controls. Note that if the side controls have been used it is necessary to click in the image window to use the mouse/keyboard controls again.

To quit the **viewer** press the 'q' or Escape keys.

1.7 CREATING A NEW MULTI-LAYER SURFACE MODEL

This section gives an example of how to create and view a new, smaller surface model by modifying an existing surface model. Computing shadows is extremely time consuming and is performed offline. The new small surface model is used to demonstrate how to use the **mkshadows** tool to create a shadow map.

1. Create a copy of the `testmodel1` directory:
   1.1. Create a copy of the `testmodel1` directory in the PANGU `models` directory.
   1.2. Change the name of the directory to `Model_1`.
   1.3. Delete the file `Model_1/4_layers.pan` if it exists.
2. Create a small surface model with three layers:
   2.1. Edit `newsurface.txt`. This contains parameters which define the creation of a new fractal surface.
   2.2. Change the Magnitude parameter from 9 to 7. This will produce a base DEM of 129 x 129 pixels.
   2.3. Save changes.
   2.4. Edit `layers.txt`. This file contains parameters which define the hierarchical surface model as a series of layers.
   2.5. Change the "Number of Layers..." parameter from 4 to 3.
   2.6. Delete the section describing layer 3. These are the last seven lines in the file.
   2.7. Change the "Polygon file name" parameter to "3_layers.pan".
   2.8. Save changes.
3. Change the crater list generation to match the smaller surface model:
   3.1. Edit the file `newcraters.txt`.
   3.2. Change the "Number of craters to generate…" parameter from 20 to 5. This will create a sparser crater population by changing the average number of craters per square kilometre, of size between 50 m and 100 m, from 20 craters/km^2 to 5 craters/km^2.
   3.3. Change parameters "Area X size" and "Area Y size" from 4096 to 1024. This defines the area into which the randomly generated craters are distributed. This new, smaller surface model is 1024 square metres so this modification changes the area into which craters are defined from 4096 square metres to 1024 square metres.
   3.4. Save changes.
4. Change the boulder list generation to match the smaller surface model:
   4.1. Edit the file `newboulders.txt`.
   4.2. Change the "Number of Boulders to add" parameter from 10000 to 250. Reducing the number of boulders to add to the surface will speed up the creation of the shadow map in step 7.

4.3. Change parameters "Area X size" and "Area Y size" from 4100 to 1024. Similar to step 3.3, this changes the area into which boulders are distributed to match the area of the surface model.

4.4. Save changes.

5. Create the new surface model:

5.1. Run `newsurface.bat`: this creates the underlying surface which is held in the file called `surface.tga`.

5.2. Run `NpCraterlist.bat`: this produces a list of craters for each surface layer defined in `layers.txt`. It uses the newcraters.txt file to define how craters are to be generated. The lists of craters produced are held in text files called layer_*N*_craterlist.txt where *N* is the layer number 0, 1, 2, …

5.3. Run `NpBoulderlist.bat`: this produces a list of boulders based on the information in the newboulders.txt file. The list of boulders is held in `boulderlist.txt`.

5.4. Run `expandsurface.bat`: this produces a hierarchy of surface layers, expanding the surface contained in the `surface.tga` file, and adding craters and boulders to the surface layers, as defined by the crater lists and boulder list.

6. View the newly created surface model:

6.1. Edit `pangu.ini`.

6.2. Change the **model** parameter to `3_layers.pan`.

6.3. Change the **shadowmap** parameter to `3_layers.smap`.

6.4. Ensure that the **view_mode** is set to **model**: this mode is often easier to use than craft mode.

6.5. Save changes.

6.6. Run `viewer.bat`. After a delay the viewer ought to appear with your new surface displayed.

7. Adding shadows to the model:

7.1. Quit the viewer.

7.2. Edit the [mkshadows] section of `pangu.ini`.

7.3. Change the **model** parameter to `3_layers.pan`: this is the model the shadow map will be generated for.

7.4. Change the **save_shadowmap** parameter to `3_layers.smap`: this is where the shadow map will be saved.

7.5. Set **shadow_time_limit** to 60 (or higher if you are prepared to wait longer). If the time limit expires before all shadows have been computed then the "Total vertices tested" will be less than the "Total vertices in model" in the output of **mkshadows** (check `PANGU.log`). Partial shadow maps will affect the realism of the images produced by the viewer, particularly for low Sun angles. The time is measured in seconds.

7.6. Set **overwrite** parameter to true: any existing `3_layers.smap` file will be overwritten.

7.7. Save changes.

7.8. Run `mkshadows.bat`

7.9. Run `viewer.bat`. After a delay the viewer will appear with your new surface displayed with a shadow map.

1.8 CREATING A NEW SINGLE LAYER MODEL (OLD METHOD)

This section gives an example of how to create and view a single-layer surface model created using the old PANGU methodology by modifying the settings for an existing surface model. Computing shadows has been shown before in Section 1.7 and so won't be shown here. We emphasise that this method of creating models is obsolete and should not be used. It is described here only for completeness.

1. Create a copy of the `testmodel2` directory:

1.1. Create a copy of the `testmodel2` directory in the PANGU `models` directory.

1.2. Change the name of the directory to `Model_2`.

1.3. Delete the `Model_2/1_layer.pan` file if it exists

2. Create a small, single-layered surface model:

2.1. Edit `newsurface.txt`. This file contains parameters which define the creation of a new fractal surface.

2.2. Set the Magnitude parameter to 8. This will result in a DEM 257 x 257 pixels.

2.3. Save changes.

2.4. Edit `layers.txt`. This file contains parameters which define the hierarchical surface model in a series of layers.

2.5. Change the "Polygon file name" parameter to `Model2.pan`.

2.6. Save changes.

3. Change the crater list generation to match the smaller surface model:

3.1. Edit the file `newcraters.txt`.

3.2. Set the "Number of craters to generate…" parameter to 1000. Ensure that the parameters "low diam" and "high diam" are set to zero. This will generate 1000 craters. If "low diam" and "high diam" are set to positive values then the 1000 craters with diameters between "low diam" and "high diam" will be generated per square km.

3.3. Set parameters "Area X size" and "Area Y size" to 257. This defines the area into which the randomly generated craters are distributed, i.e. the size of the DEM.

3.4. Save changes.

4. Change the boulder list generation to match the new surface model:

4.1. Edit the file `newboulders.txt`.

4.2. Set the "Number of Boulders to add" parameter to 500. Reducing the number of boulders to add to the surface will speed up the creation of the shadow map in step 7.

4.3. Set parameters "Area X size" and "Area Y size to 257. Similar to step 3.3, this changes the area into which boulders are distributed to match the area of the surface model.

4.4. Save changes.

5. Create the new surface model:

5.1. Run `newsurface.bat`: this creates the underlying surface which is held in the file called `surface.tga`.

5.2. Run `Craterlist.bat`: this produces a list of craters. It uses the `newcraters.txt` file to define how craters are to be generated. The list of craters to be produced is held in a text file called `craterlist.txt`.

5.3. Run `Boulderlist.bat`: this produces a list of boulders based on the information in the `newboulders.txt` file. The list of boulders is held in `boulderlist.txt`.

5.4. Run `makemodel.bat`: this adds the craters and boulders defined in `craterlist.txt` and `boulderlist.txt` to the surface and generates a polygon model that can be rendered using the **Viewer**.

6. View the newly created surface model:

6.1. Edit `pangu.ini`.

6.2. Change the **model** parameter to `1_layer.pan`.

6.3. Change the **shadowmap** parameter to `1_layer.smap`.

6.4. Ensure that the **view_mode** is set to **model**: this mode is often easier to use than craft mode.

6.5. Save changes.

6.6. Run `viewer.bat`. After a delay the **viewer** ought to appear with your new surface displayed.

We must emphasize that this method should not be used to create models: use the hierarchical approach.

1.9 CREATING ROAM MODELS

This section gives an example of how to create a ROAM model. These are similar to the existing mesh models but use a different rendering algorithm in the viewer. The models are also much smaller than the normal mesh models.

1. Create a copy of the `testmodel3` directory:
   1.1. Create a copy of the `testmodel3` directory in the PANGU `models` directory.
   1.2. Change the name of the directory to `Model_3`.
2. Run `confidence.bat` to generate and display the ROAM model:
   2.1. The result ought to look just the same as the testmodel1 model
   2.2. Press SHIFT-F several times to decrease the ROAM quality and notice the effect
   2.3. Press F to increase the ROAM quality back to its original value
   2.4. Press W to enter wire-frame mode and try the SHIFT-F/F key presses again. Try zooming in and out of the model while in wire-frame mode: the triangle sizes ought to remain approximately the same at all times
   2.5. Press SHIFT-E several times to increase the smallest triangle size: press E to undo the effect.
3. Quit the viewer and examine the PANGU.log file (or the console output under UNIX):
   3.1. The “ROAM limit” value can be used to set `viewer.roam_limit` in the PANGU.ini file.
   3.2. The “ROAM size factor” can be used to set `viewer.roam_size_factor` in the PANGU.ini file.
   3.3. These values control the default ROAM quality settings whenever the **viewer** is used.

The ROAM system attempts to split the triangles in the model into smaller and smaller triangles until the quality error is less than viewer.roam_limit. The quality error is the difference between a triangle and the two smaller triangles that it can be split into. The viewer attempts to ensure that the triangle size scaled by viewer.roam_size_factor) is not less than viewer.roam_limit to prevent the ROAM algorithm splitting the mesh too finely.

The general technique for choosing the ROAM parameters is to set the ROAM size factor to a very small value and adjust the ROAM limit until the model appears with the desired sharpness. Then increase the size factor to reduce any over-rendering where the mesh size is much smaller than the size of a pixel.

1.10 CREATING WHOLE-PLANET MODELS

This section gives an example of how to create a whole-planet model. These are spherical models designed to be viewed from a distance. Surface features are provided by a single texture map.

4. Create a copy of the `models/whole_planet` directory:
   4.1. Create a copy of the `whole_planet` directory in the PANGU `models` directory.
   4.2. Change the name of the directory to `Planet_1`.
5. Define a higher resolution planet (or moon):
   5.1. Edit the file `WholePlanet.txt`. This contains all the parameters which define the planet.
   5.2. Change the “Output file name” parameter to `io.pan`: this will be the Jovian moon Io.
   5.3. Change the “Number of latitudes” parameter to 10: this is the number of horizontal rings in the model.
   5.4. Change the “Number of longitudes” parameter to 20: this is the number of quadrilaterals per ring of the model.

5.5. Change the “Planet radius” parameter to 500: this is the radius of the planet in metres. In this example is it simpler to use a very small value to make viewing simpler.

5.6. Change the “Texture filename” parameter to `io_sphere.pppm`: this is a texture map for the Jovian moon Io. Any PPM image may be used provided it is square with side of length 2^N for some integer N.

6. Generate and view the model:

6.1. Run `make_planet.bat` to generate the model

6.2. Run `viewer.bat` to view the model

6.3. Note how faceted the model is particularly on the limb of Io. Press W to get the wireframe mode to see just how few triangles are used and to allow the number of lines of longitude and latitude to be counted.

1.11 ASTEROID MODEL

This section shows how to generate an asteroid model using the asteroid example provided in the directory `models/asteroid_model`. This model is then edited to create a higher resolution asteroid with fewer craters.

1. Create a copy of the `asteroid_model` directory: call it `Asteroid_1`.
2. Run `MakeAsteroid.bat`. This should create an asteroid model and display it using the viewer.

2.1. Quit the viewer when finished viewing the model.

3. Edit the file `asteroidparams.txt`. This file contains parameters which define the creation of an asteroid model.

3.1. Change the “Asteroid Magnitude” parameter from 200 to 300. This will result in an asteroid model with 601 defined latitudes and will contain more than twice the number of vertices.

3.2. Change the “Fixed stretch on x-axis parameter” from 1 to 1.5. This will result in a more oblong shaped asteroid.

3.3. Change the “Load model” parameter from 1 to 0. This specifies that a base asteroid model will not be loaded and will instead be generated.

3.4. Change the “Save model” parameter from 0 to 1. This specifies that the base asteroid model will be saved.

3.5. Change the “Save model filename” parameter to `asteroid_300.mod`. This specifies that the newly generated base asteroid model will be saved to this filename. Generating high resolution base asteroid models can take a long time so it is convenient to be able to load and save these intermediate model files.

3.6. Save changes.

4. Change the craterlist generation to increase the number of craters.

4.1. Edit the file `newcraters.txt`.

4.2. Change the “Number of craters to generate…” parameter from 3000 to 1000.

4.3. Save changes.

5. Run `MakeAsteroid.bat`. This should generate the new asteroid model and display it using the viewer.

5.1. Quit the viewer when finished viewing the model.

6. Adding shadows to the asteroid model. Shadows can be generated for asteroid models in the same way that they are generated for other surface models.

6.1. Quit the viewer.

6.2. Edit the "[mkshadows]" section of pangu.ini.

6.3. Change the “model” parameter to `asteroid.pan`: this is the model the shadowmap will be generated for.

6.4. Change the "save_shadowmap" parameter to `asteroid.smap`: this is where the shadowmap will be saved.

6.5. Set "shadow_time_limit" to 600 seconds (or higher if you are prepared to wait longer). If the time limit expires before all shadows have been computed then the “Total vertices tested” will be less than the “Total vertices in model” in the output of mkshadows (check `PANGU.log`). Partial shadowmaps will affect the realism of the images produced by the viewer, particularly for low Sun angles.

6.6. Set “clobber” parameter to true: any existing `asteroid.smap` file will be overwritten.

6.7. Save changes.

6.8. Run `mkshadows.bat`.

6.9. Run `viewer.bat`. After a delay the viewer ought to appear with your new surface displayed with a shadowmap. Note that it is unlikely that much of the asteroid would have been shadow mapped during this time so the shadowed parts of the model may not be obvious.

1.12 FLAT BOTTOMED MARTIAN CRATERS

This section describes how to modify the crater model properties to create flat bottomed craters as found on surfaces such as Mars where there are significant weathering processes.

1. Modify `testmodel1` to generate flat bottomed craters:
   1.1. Go to the `models/testmodel1` directory.
2. Modify the crater model parameter file to generate flat bottomed craters:
   2.1. Edit the file `cratermodel.txt`.
   2.2. Change “Apply flat bottom fill-ins” to 1
   2.3. Change “Flat bottom curve width” to 0.2
   2.4. Set Flat age start (normalised) to 0
   2.5. Set Flat age factor to 1
   2.6. Save changes and recreate the model by running `confidence.bat`.

1.13 DUNE MODELS

This section gives an example of creating a model with sand dunes using `testmodel4`. This model is identical to `testmodel1` except for the inclusion of sand dunes.

1. Create a copy of the `testmodel4` directory and call it `Model_4`.
   1.1. Run `confidence.bat`: a model should be generated showing dunes, craters and boulders.
   1.2. This corresponds to the centre of Figure 2-8.
2. Change the form of the dunes by changing the wind speed and sand supply parameters:
   2.1. Edit `dunemodel.txt`: this contains parameters characterising the dune model.
   2.2. Change the sand supply parameter to 2
   2.3. Change the wind speed parameter to 9
3. Create and view the new model:
   3.1. Run `confidence.bat`.
   3.2. The dune population is now sparser and composed of crescentic barchan dunes.
   3.3. This corresponds to the left of Figure 2-8.

1.14 DEFINING CRATER AND BOULDER POSITIONS MANUALLY

This section gives an example of how to define crater and boulder positions manually in a test model.

1. Create a copy of the `testmodel1` directory:
   1.1. Create a copy of the `testmodel1` directory in the PANGU `models` directory.
   1.2. Change the name of the directory to `Model_5`.
   1.3. Delete the `Model_5/4_layers.pan` file if it exists.
2. Create a small surface model with two layers:
   2.1. Edit `newsurface.txt`. This file contains parameters which define the creation of a new fractal surface.
   2.2. Change the Magnitude parameter from 9 to 7. This will result in a base DEM of 129 x 129 pixels.
   2.3. Save changes.
   2.4. Edit `layers.txt`. This file contains parameters which define the hierarchical surface model in a series of layers.
   2.5. Change the "Number of Layers..." parameter from 4 to 2.
   2.6. Delete the sections describing layers 2 and 3.
   2.7. Change the "Polygon file name" parameter to `2_layers.pan`.
   2.8. Save changes.
3. Manually define two craters:
   3.1. Open the file `layer_0_craterlist.txt` in a text editor.
   3.2. Delete all lines under the line "// x_pos y_pos diameter age". This will delete all crater definitions in this file.
   3.3. Add the following lines to the end of the file which will define two craters. The first crater is positioned in the centre of the surface, is 100 metres in diameter and is aged 25 units. The second crater is defined at position (20 m, -30 m), is 50 m in diameter and is aged 1 unit.

```
0  0     100   25
20 -30   50    1
```

   3.4. Save changes.
4. Manually define two boulders:
   4.1. Open the file `boulderlist.txt` in a text editor.
   4.2. Delete all lines under the line "// x_pos y_pos size depth libraryNumber". This will delete all boulder definitions in this file.
   4.3. Add the following lines to the end of the file which will define two boulders. The first boulder is positioned in the centre of the surface, is 5 metres in size, burial depth is 2 m and is boulder 3 in the boulder library. The second boulder is defined at position (-20 m, 10 m), is 8 m in size, is buried 4 m into the surface and boulder 3 in the boulder library. Note that there are three boulders in the current boulder library, numbered 0, 1 and 2.

```
0  0     5     2     2
20 10    8     4     2
```

   4.4. Save changes.
5. Create the new surface model:
   5.1. Run `newsurface.bat`: this creates the underlying surface which is held in the file called `surface.tga`.

5.2. Run `expandsurface.bat`: this produces a hierarchy of surface layers, expanding the surface contained in the `surface.tga` file, and adding craters and boulders to the surface layers, as defined by the crater lists and boulder list.

6. View the newly created surface model (see Section 1.7 part 6 for details):

6.1. Edit `pangu.ini`.

6.2. Change the **model** parameter to `2_layers.pan`.

6.3. Change the **shadowmap** parameter to `2_layers.smap`.

6.4. Ensure that **view_mode** is set to **model**.

6.5. Save changes.

6.6. Run `viewer.bat`. After a delay the viewer ought to appear with your new surface displayed.

7. Adding shadows to the model (see Section 1.7 part 6 for details):

7.1. Quit the viewer.

7.2. Edit the [mkshadows] section of `pangu.ini`.

7.3. Change the **model** parameter to `2_layers.pan`: this is the model the shadow map will be generated for.

7.4. Change the **save_shadowmap** parameter to `2_layers.smap`: this is where the shadow map will be saved.

7.5. Set **shadow_time_limit** to 60 (seconds) or the longest time that you are prepared to wait.

7.6. Set **overwrite** parameter to true: any existing `2_layers.smap` file will be overwritten.

7.7. Save changes.

7.8. Run `mkshadows.bat`

8. Run `viewer.bat`. After a delay the viewer ought to appear with your new surface displayed with a shadow map.

1.15 FOG/DUST MODELLING

This section gives an example of viewing a model with fog/dust.

1. Create a copy of the `testmodel1` directory:

1.1. Create a copy of the `dustmodel` directory in the PANGU `models` directory.

1.2. Change the name of the directory to `Model_Dust`.

1.3. Run `confidence.bat` to create `4_layers.pan` if it doesn't already exist.

2. Update the `PANGU.ini` file in the `Model_Dust` directory with global fog/dust by changing the following lines in the [`_defaults`] section around line 192:

```
viewer.fog.sky_distance            1e18
viewer.plane_fog.enable            false
viewer.global_fog.enable           true
viewer.global_fog.mode             linear
viewer.global_fog.linear_start     10.0
viewer.global_fog.linear_end       800.0
viewer.global_fog.density          0.002
viewer.global_fog.colour           1.0 0.0 0.0
```

3. Run `viewer.bat` and notice that the scene is filled with thick red fog/dust.

3.1. When close to the terrain the fog/dust will be thin while anything further then 800 m away will be completely obscured. Note that with the fog/dust mode set to linear the fog density is ignored.

3.2. Change the fog mode to “exp” or “exp2” to get fog with exponential extinction profiles. With these modes the `density` parameter controls the thickness of the fog/dust and the `linear_start` and `linear_end` settings are ignored.

3.3. When in the **viewer**, use F11 to cycle through fog modes and k/K to change fog density

4. Update the `PANGU.ini` file in the `Model_Dust` directory with plane fog/dust by adding to (or changing) the following lines in the [`_defaults`] section around line 192:

```
viewer.global_fog.enable            false
viewer.plane_fog.enable             true
viewer.plane_fog.density            0.004
viewer.plane_fog.colour             1.0 0.0 0.0
 viewer.plane_fog.height            100.0
```

5. Run `viewer.bat` and notice that any point in the terrain that is below 100 m in altitude will be covered with fairly thick red fog/dust cloud. Use the down arrow key to reduce the camera elevation angle and notice that fog extends to the horizon and ends with a sharp edge initially. However, as the camera elevation is reduced further so that the camera drops below the 100 m altitude then the sky will become partly obscured as well. This is most obvious with a starry background.

6. Update the `PANGU.ini` file in the `Model_Dust` directory with spherical fog/dust by adding to (or changing) the following lines in the [`_defaults`] section around line 192:

```
viewer.global_fog.enable            false
viewer.plane_fog.enable             false
viewer.sphere_fog.enable            true
viewer.sphere_fog.density           0.02
viewer.sphere_fog.colour            0.0 0.0 1.0
viewer.sphere_fog.radius            40
viewer.sphere_fog.origin            0 0 100
```

7. Run `confidence.bat` and notice that there is a semi-transparent blue sphere floating above the surface. The surface terrain ought to be visible through the edges of the sphere. With care it is possible to navigate to a point inside the sphere: the use of the craft view will make this easier (see Section 3.3.4).

8. Update the `PANGU.ini` file in the `Model_Dust` directory with spherical fog/dust by adding to or changing the following lines in the [`_defaults`] section around line 192:

```
viewer.global_fog.enable            false
viewer.plane_fog.enable             false
viewer.sphere_fog.enable            false
viewer.general_fog.enable           true
viewer.general_fog.model            ../../samples/sphere_50m.pan
viewer.general_fog.density          0.04
viewer.general_fog.colour           0.0 1.0 0.0
viewer.general_fog.origin           10 20 0
```

9. Run `confidence.bat` and notice that there is a semi-transparent green hemisphere on the surface. The surface terrain ought to be visible through the edges of the sphere. With care it is possible to navigate to a point inside the sphere although the use of the craft view (see Section 3.3.4) will make this easier.

1.16 USING THE MEMORY MANAGEMENT SYSTEM

This section gives an example of how to use the new memory management system.

1. Run the confidence test in `testmodel1` if you haven’t done so already (see Section 1.7):

1.1. Quit the viewer after a few seconds of animation

1.2. Look at the end of the `PANGU.log` file (or terminal output under UNIX) for the cache report:

1.2.1. Cache objects **X**/**Y**: this shows the number of cachable objects **X** out of a total of **Y** that are currently in the memory manager disk cache.

1.2.2. Cache blocks **X**/**Y**: this shows the number of cache blocks **X** out of a total of **Y** that are in use on disk

1.2.3. Cache wastage: cache blocks are all the same size while PANGU objects are varied in size. This means that a certain amount of the cache will be wasted and this is shown here.

1.2.4. Cache block size: this shows the size of each block in the cache. The size is computed automatically based on the current model and attempts to minimise cache wastage while maximising access speed.

1.2.5. Cachable RAM: this shows the amount of data in the model which can be cached

1.2.6. Cachable in RAM: this shows the amount of cachable data in the model which is currently in RAM

1.2.7. Cachable in cache: this shows the amount of cachable data in the model which is in the disk cache

1.2.8. Cachable on disk: this shows the amount of cachable data left on disk in the .pan file.

2. Update the memory management parameters in `PANGU.ini`:

   2.1. Load `PANGU.ini` into a text editor and locate the [_default] section

   2.2. Set `cache.size` to 30000000 (approximately 30 Megabytes): this is the amount of disk space which will be used to store model objects that have been evicted from RAM because they are not visible

   2.3. Set `cache.ceiling` to 5000000 (approximately 5 Megabytes): this sets the amount of RAM which can be used by cachable objects. Non-cachable objects will always be held in RAM but these are usually a small fraction of the amount of cachable objects in a model. If more than `cache.ceiling` cachable RAM is being used then unused objects will be unloaded into the disk cache. If the disk cache is full the oldest objects will be dropped and must be reloaded from the .pan file when they are needed.

   2.4. For slow systems or systems with slow disks the `cache.burst_limit` can be lowered. After rendering each frame the viewer will ask the memory management system to reduce memory usage below the cache ceiling set by the user. If a large model is being used there might be millions of objects which are not visible and which can be written to the cache. This would take a significant amount of time which would be better spent rendering the view. As a result the memory management system will never write more than `cache.burst_limit` objects to disk when asked to satisfy the cache ceiling. This may mean that more than the cache ceiling of RAM is being used by ensures that the viewer doesn't spend too much time in memory management.

3. Run `viewer.bat` to redisplay the model:

   3.1. Press "=" to show memory management statistics (check `PANGU.log` or the terminal for these statistics)

   3.2. Spin the model around and change the camera view from viewing directly down at the model to a side view where the horizon is visible.

Press "=" again and check the effect on the cache in `PANGU.log` or the terminal output.

Note that the memory management system attempts to keep memory usage within the `cache.ceiling` value defined by the user. However, if the current view requires more than `cache.ceiling` to be used then this will happen: objects will not be put into the cache to satisfy memory limits if they are needed for the current view.

1.17 USING THE SOCKET INTERFACE

The **viewer** can be run in server mode: remote clients can specify a camera position and attitude and the **viewer** will send back an image in return. To change the viewer to run in server mode:

- Load the `pangu.ini` file and locate the [viewer] section.
- Set the **server_mode** parameter to true.
- Set the **server_port** to 10363 (TCP/IP Port).

- Ensure that the **model** and **shadowmap** settings refer to the model and shadow map you wish to use, save and close the `pangu.ini` file.
- Start the viewer as normal (*e.g.* run `viewer.bat`).

Now use the sample PANGU client:

- Load the `pangu.ini` file and locate the [pangu_client] section. If one does not exist then create an empty one.
- Set the camera position and attitude with the **free_camera** option. For example, to position the camera at coordinates (x, y, z) with attitude (yaw, pitch, roll) you will need:

```
free_camera   x y z   yaw pitch roll
```

- Specify the server name (or IP address) and port number. If the PANGU viewer (server) is running on the current machine with the default port number then the following ought to work:

```
server_name   127.0.0.1
server_port   10363
```

- Specify the major number of the PANGU network protocol to use (0 or 1). For example:

```
protocol      1
```

- Ensure that the PANGU viewer is visible.
- Run the client program: `pangu/bin/pangu_client.exe` (`pangu_client` under Linux).
- Use the up/down arrow keys to change the pitch of the camera and the left/right arrow keys to change the yaw.

1.18 USING THE RADAR_DEMO PROGRAM

This tool is a small demonstration application which shows how a C program can be used to retrieve range and speed information so that a RADAR response can be constructed. The main purpose is to show how to use the PANGU network protocol RADAR messages rather than act as a RADAR simulation tool.

Start the **viewer** in server mode as described in Section 1.17. Then from a command line (*e.g.* a DOS window) type the following command (assuming you are in the PANGU models directory):

```
..\..\bin\radar_demo –position 0 0 600 –samples 200
```

This will run the **radar_demo** program specifying the position of the RADAR emitter to be (0,0,600) *i.e.* 600 units directly above the model origin. The RADAR beam footprint will be sampled in 200 places. The resulting histogram of range results will be written to the command window. For more details see Section 3.4.5.

1.18.1 Command-line options

The command line argument syntax is:

```
radar_demo [options]
```

The [options] are:

1.18.1.1 Server options

-server <host>	connect to the PANGU server on machine <host>
-port <port>	connect to the PANGU server using TCP/IP port <port>

1.18.1.2 Radar emitter options

-position <x> <y> <z>	position of the RADAR emitter relative to the target landing site
-velocity <x> <y> <z>	velocity of the RADAR emitter relative to the target landing site
-attitude <q_0> <q_1> <q_2> <q_3>	attitude of the RADAR emitter relative to the target landing site
-beam_width <w>	full width of the RADAR beam is <w> degrees

1.18.1.3 Simulation options

-samples <n>	sample the beam foot print <n> times
-[no]normalise	[don't] normalise the histogram samples so that their sum is 1
-[no]slope	[don't] consider the effects of surface slope on sample strength
-rbins <n>	use <n> bins for the range axis of the histogram (default 51)
-sbins <n>	use <n> bins for the speed axis of the histogram (default 1)
-rround	round up the range histogram width to be an integer power of 10
-sround	round up the speed histogram width to be an integer power of 10
-rleft	fix the left edge of the range histogram to have range 0
-sleft	fix the left edge of the speed histogram to have speed 0
-rcentre <r>	align the range histograms so that range <r> is in the centre
-scentre <s>	align the speed histograms so that speed <s> is in the centre
-rbinsize <r>	fix the range histogram bins to be <r> units in size
-sbinsize <s>	fix the speed histogram bins to be <s> units in size

1.18.1.4 Position and attitude

The RADAR emitter beam axis is along the positive z-axis by default which means that the axis points up from the model surface (to match with the conventions used elsewhere in PANGU for imaging and LIDAR). The attitude is defined by the rotation quaternion which defaults to [0, 1, 0, 0] in radar_demo. This corresponds to a rotation of 180° about the (horizontal) x-axis [1, 0, 0] which means the beam points directly down at the surface. The position of the emitter [x,y,z] defines the position relative to the PANGU origin as defined in Section 2.2. The x-y coordinates are coordinates on the surface of the model with the z-coordinate defining the height of the emitter above the surface. The units for position, range and speed are arbitrary: if the position and velocity are specified in metres and metres/second then the response will be in metres and metres/second.

1.18.1.5 Output format

The output of this program is a 2D or 3D data set defining the contents of the range/speed histogram for the RADAR response obtained from the specified position and velocity. The beam direction in this program always points down at the nadir. If the number of speed bins is one then a 2D table of signal strength *versus* range is written to the standard output stream. If the number of range bins is one then a 2D table of signal strength *versus* speed is produced. If the number of range and speed bins are both greater than one then a 3D table of signal strength *versus* range and speed is produced. The columns of the results tables are TAB separated and are prefixed by comments describing the RADAR response (lines which start with a # comment marker).

These results can be loaded into a spreadsheet or graph plotting program for analysis and display.

1.19 USING THE LIDAR_DEMO PROGRAM

This tool is a small demonstration application which shows how a C program can be used to retrieve range and slope information so that a LIDAR response can be constructed. The main purpose is to show how to use the PANGU network protocol LIDAR messages rather than act as a LIDAR simulation tool.

Start the **viewer** in server mode as described in Section 1.17. Then from a command line (*e.g.* a DOS window) type the following command (assuming you are in the PANGU models directory):

```
..\..\bin\lidar_demo -range -position 0 0 100 -attitude 0 1 0 0 -o img.ppm
```

This will run the **lidar_demo** program to obtain a range image specifying the position of the LIDAR emitter to be (0,0,100) *i.e.* 100 units directly above the model origin and the attitude quaternion to be [0,1,0,0] corresponding to looking directly down at the model origin. The image is saved in `img.ppm`. For more details see Section 3.4.1.

1.19.1 Command-line options

The command line argument syntax is:

```
lidar_demo [options]
```

The [options] are:

1.19.1.1 Server options

-server <host>	connect to the PANGU server on machine <host>
-port <port>	connect to the PANGU server using TCP/IP port <port>

1.19.1.2 Lidar emitter options

-position <x> <y> <z>	position of the LIDAR emitter relative to the target landing site
-velocity <x> <y> <z>	linear velocity of the LIDAR emitter relative to the target landing site
-acceleration <x> <y> <z>	linear acceleration of the LIDAR emitter relative to the target landing site
-jerk <x> <y> <z>	linear jerk of the LIDAR emitter relative to the target landing site
-attitude <q_0> <q_1> <q_2> <q_3>	attitude of the LIDAR emitter relative to the PANGU frame
-rotation <x> <y> <z>	angular velocity of the LIDAR emitter
-racceleration <x> <y> <z>	angular acceleration of the LIDAR emitter
-rjerk <x> <y> <z>	angular jerk of the LIDAR emitter

1.19.1.3 Corner cube options

-cattitude <q_0> <q_1> <q_2> <q_3>	attitude of the corner cube lattice (positioned at the origin)
-crotation <x> <y> <z>	angular velocity of the corner cube lattice
-cacceleration <x> <y> <z>	angular acceleration of the corner cube lattice
-cjerk <x> <y> <z>	angular jerk of the corner cube lattice

1.19.1.4 Scan options

-type <n>	use scan type <n> (default 1)
-width <n>	number of pulses emitted per line
-height <n>	number of lines per scan
-xsamples <n>	number of horizontal samples per pulse
-ysamples <n>	number of vertical samples per pulse
-fwidth <f>	angular field of view width in degrees
-fheight <f>	angular field of view height in degrees
-wx <f>	angular horizontal size of the detector pixel
-wy <f>	angular vertical size of the detector pixel
-tpulse <f>	time between successive pulses on a line
-tgap <f>	time between the last pulse of one line and the first pulse of the next
-offset <a> <e>	set the azimuth <a> and elevation <e> of the centre of the scan (degrees)

1.19.1.5 Result options

-o <file>	write data to <file> (a PPM image if `-text` or `-dem` not specified)
-grey	save images in grey-scale for printing (default is colour)
-dem	save data as a PANGU DEM (but in PPM format not TGA)
-text	save data as text: sample (*e.g.* range), x-coordinate, y-coordinate
-corner_cubes	ask server for corner cube data even if it won't be used
-elevation	save pulse elevation angle data
-crange	save corner cube range data
-cslope	save corner cube slope data
-range	save range data (default)
-slope	save slope data
-time	save pulse emission time data
-maprange <a> <b>	ignore min/max limits from server and use min=<a>, max=<b>

1.19.1.6 Image labelling options

-label	label saved images with axes
-invert	reverse the colours of labelled images
-tick <f>	specify the angular separation of ticks on labelled images
-grid <f>	specify the angular separation between grid lines on labelled images

1.19.1.7 Overview

The default behaviour of **lidar_demo** is to use the PANGU TCP/IP **SetLidarParameters** message using the values passed to the program from the command line and from internal defaults. Then **GetLidarMeasurement** message is used to obtain the necessary range and slope information requested by the user. This information is encoded as a PPM image which can be written to a file. Images are rainbow colour-coded with red representing small ranges/slopes and blue/violet representing large ranges/slopes. If the `-text` option is used then the data is saved as a text file that could be loaded into a spreadsheet for analysis. The result options desfined in Section 1.19.1.6 can be used to add labelled axes, grid lines and colour map keys to the image to support further visual analysis.

The `-wx`/`-wy` command line options correspond to w_x and w_y of Section 3.4.1. Similarly the `-xsamples`/`-ysamples` options correspond to n/m; `-width`/`-height` correspond to W/H, `-tpulse`/`-tgap` to T_{pulse}/ $T_{interrow}$, `-offset` to Az_0/ El_0, `-fwidth`/`-fheight` to FOV_x / FOV_y.

1.19.1.8 Craft Kinematics

The PANGU/LIDAR model was designed to support the simulation of a system in which a full scan took up to one second to obtain. During this time the LIDAR emitter/detector may have moved a large distance and undergone large changes in attitude. To capture this behaviour the client must specify the linear and angular motion of the craft when it is in the middle of the scan. For linear motion the position, velocity, acceleration and jerk can be specified via the `-position`/`-velocity`/`-acceleration`/`-jerk` options; similarly for angular motion the attitude, angular velocity (rotation), acceleration and jerk can be specified via the `-attitude`/`-rotation`/`-racceleration`/`-rjerk`. If these values are not specified to **lidar_demo** then velocity, acceleration and jerk will be assumed to be zero.

1.19.1.9 Corner Cube Kinematics

The PANGU/LIDAR model was designed to support the simulation of corner cubes attached in a rigid lattice around the target at the origin (*e.g.* a sample-return canister). Although the corner cube lattice is fixed in position it may be rotating relative to the LIDAR emitter/detector. Thus the attitude and angular velocity, acceleration and jerk may be specified via the `-cattitude`/`-crotation`/`-cacceleration`/`-cjerk` options. To obtain range and slope images for the corner cube lattice use the `-crange` or `-cslope` options.

1.19.1.10 Output Images

For a W×H scan with nxm samples per pulse the result image will have (W×n)×(H×m) pixels. These will be arranged in a rectangular grid in the same way that a visual image would be arranged with one aspect of the pulse values being used to colour individual pixels. The default aspect is range (distance) to the target.

Range images are colour coded as follows: the smallest range obtained from the set of all pulse results is assigned the colour red and the largest range the colour violet. All intermediate ranges are assigned intermediate colours of the rainbow. If the pulse did not intersect with the target then the pixel is coloured black. The -maprange option can be used to override the largest and smallest range values used to determine the mapping to allow the same range/colour mapping to be used for different images for comparison. Otherwise images obtained from different distances may be assigned identical colours with specific colours representing different distances.

Slope images are colour coded in the same way except that the the slope angle is used instead of range. For a surface whose normal is parallel to the LIDAR pulse direction the angle is zero while a surface parallel to the LIDAR pulse direction has angle 90°. As before the smallest slope angle will be coloured red and the largest coloured violet.

The azimuth and elevation images are identical to the slope images except that the azimuth or elevation angle of the LIDAR beam inside the emitter is used. Similarly the time image is identical to the range image except that the time at which the pulse was emitted relative to the centre of the scan is used.

1.20 USING THE ALIAS_MEASURE PROGRAM

This tool can be used to provide a quantative measure of the amount of aliasing present in PANGU images. It is based on the assumption that a pair of images obtained from similar view points ought to be identical except for differences due to zoom effects. Any difference between the two images is assumed to be due to rendering artifacts. Significant differences between pairs of pixels are assumed to be due to aliasing artifacts.

The tool accepts two images of identical size taken at distances D_1 and D_2 respectively from the planet surface at the centre of the field of view and a zoom ratio $Z=D_1/D_2$. If $Z>1$ the first image will be zoomed by a factor of Z to compensate for the difference in scale relative to the second (reference) image. If $Z<1$ then the second image will be zoomed by a factor of $1/Z$ instead to compensate for the difference in scale relative to the first (reference) image. The sum of the absolute difference in pixel values between the new image pair is then computed and scaled by the number of pixels in the image. The result is a number which may be used (with care) to compare aliasing in PANGU images.

A measure value of 0 indicates that the images are identical. The threshold value at which significant aliasing occurs must be determined manually for a particular view point: the measure value provides a way to objectively compare the effects of different **viewer** rendering settings.

Note that if $Z\neq1$ then the zoom compensation filter will produce a non-zero alias measure even if the two view points were rendered without any artifacts. Thus care must be used when interpreting the results from this program.

1.20.1 Command-line options

The command line argument syntax is:

```
alias_measure [options] file1 file2 [ratio]
```

The [options] are:

-blur	apply a Gaussian blur filter to the reference image
-zoom <file>	save the zoom-compensated image to <file>
-other <file>	save the reference image to <file>
-difference <file>	save the difference image to <file>

The `file1` and `file2` options specify the name of the images taken at distances of D_1 and D_2 respectively from the planet surface visible in the centre of the images. The optional ratio argument specifies the value of the distance ratio D_1/D_2 and is assumed to be 1 if not specified. The two images must be identical in width and height.

The program outputs five floating point values, one *per* line, labelled GREY, RED, GREEN, BLUE and MAX. These are the alias measure for grey-scale images, the red, green and blue colour planes and the maximum of all four.

1.21 USING THE TXTTOPAN PROGRAM

Version 1.50 and later of **surfacegen** provided the ability to save a copy of the digital elevation models (DEMs) used to create the layers of the `.pan` polygon file as a single text DEM file using the `-Y` command line option. The text DEM file format was designed so that it could be read easily into a program such as MatLab for analysis. The 3D coordinates of each unique vertex point in the surface model (excluding boulders but including craters and sand dunes) is stored on a separate line of the text file separated by semicolons (;).

The **txttopan** tool can be used to convert one of these text DEM files into a PANGU `.pan` polygon file. Only files created using the `-Y` option to **surfacegen** may be reliably converted by this program. This is because **txttopan** must analyse the pattern of vertex points in the input file to determine the size and number of layers in the model.

1.21.1 Command-line options

The command line argument syntax is:

```
txttopan [options]
```

The [options] are:

-R	generate a ROAM model
-Z	generate the output `.pan` file lazily

-L <dem >	load data from the text DEM file <dem>
-O <output>	write the output `.pan` file to <output>

Note that the `-R`, `-Z` and `-L` options must be specified before the `-O` option otherwise they will be ignored.

1.22 USING THE VIEWER SELF-TEST FACILITIES

The PANGU **viewer** application has been extended with self-test facilies. Three tests are provided with release 2.70: they can be used to validate the rendering and back-projection system.

1.22.1 Command-line options

The command line argument syntax is:

```
viewer [options]
```

The self-test [`options`] are:

`-test lambert_00`	Lambertian reflection model test 0
`-test back_project_00`	back-projection test 0
`-test back_project_01`	back-projection test 1

It is recommended that the `-quit` option of the viewer is used to force the **viewer** to quit immediately after all the self tests have been performed. The `-list_tests` option can be used to see the list of available tests.

A detailed description of the available self-tests can be found in Section 3.4.7.

2. INTRODUCTION

This document contains the user manual for PANGU. It describes how to use PANGU to create, and then visualize, synthetic planetary surface images. There are three main executables:

1. SurfaceGen (generates surface models).
2. FeatureListGen (generates lists of craters and boulders which are used by SurfaceGen).
3. Viewer (renders images from surface models created by SurfaceGen).

Other PANGU executables:

1. mkshadows and mergeshadows (generate shadow maps for point and area light sources).
2. mktexture and edtexture (generate and manipulate surface and boulder textures).
3. imgview (view images saved by the viewer including blink comparison).
4. pandump (display low-level information about a PANGU .pan file).
5. pangu_client (example remote TCP/IP client).
6. PovGen (generate a POV-Ray version of a PANGU model).
7. pan2pov (convert a PANGU `.pan` file into a POV-Ray file).
8. lidar_demo (remote TCP/IP client to access LIDAR results).
9. radar_demo (remove TCP/IP client to access RADAR results).

PANGU has been tested by the developers on Win98, WinNT, Win2000, WinXP as well as various versions of RedHat/Fedora Linux.

2.1 THE PANGU INSTALLATION STRUCTURE

After installation the following structure will be installed on Windows platforms. The Linux installation has a similar structure, mostly omitting Windows-specific files.

The PANGU directory ought to contain the following folders and files:

- bin: a folder of PANGU executables.
- client: a folder containing a client socket interface source code.
- contrib: a folder containing non-standard library contributions.
- docs: a folder containing user documentation.
- lib: a folder containing Windows dynamic libraries which may be needed.
- models: a folder containing example surface models
- asteroid_model: example PANGU parameter files and batch files for an asteroid model.
- testmodel1: example PANGU parameter files and batch files for a hierarchical model.
- testmodel2: example PANGU parameter files and batch files for a single-layer model.
- testmodel3: example PANGU parameter files and batch files for a multi-layer ROAM model.
- testmodel4: example PANGU parameter files and batch files for a sand dune model.
- upgrade_model: PANGU parameter files used to validate the 2.10 upgrade.
- wholeplanet_model: example PANGU parameter files and batch files for a whole-planet model.
- PovrayInclude: the PANGU/POV-Ray boulder0.inc, boulder1.inc and boulder2.inc files.
- samples: a folder containing example surface models and shadow maps.
- textures: a folder containing surface and boulder textures for use with the viewer.

Surface models can be created by calling **surfacegen** and **featurelistgen** directly from a command line or by running the `models/testmodel*` batch files which have been provided as a convenience. Users are advised to generate hierarchical surface models to enable the simulation of large surface models.

The batch files in the `models/testmodel*` directories used to generate hierarchical surfaces are:

- confidence.bat: runs each PANGU subcommand in turn and the runs the viewer.
- newsurface.bat: creates a new fractal surface DEM.
- npcraterlist.bat: creates crater lists for each level in the hierarchy.
- npboulderlist.bat: creates a boulder list.

- expandsurface.bat: creates a hierarchical surface with craters and boulders from a base DEM and crater and boulder lists.

The batch files in the `models/testmodel*` directories used to generate single resolution surfaces are:

- newsurface.bat: creates the initial landscape DEM
- craterlist.bat: creates crater lists for each level in the hierarchy
- addcraters.bat: adds craters to initial landscape to produced cratered DEM
- boulders.bat: creates boulder distribution, possibly based on crater list

The generated surfaces stored in .pan format may be rendered using the viewer whose settings are found in the `[_defaults]` and `[viewer]` sections of `pangu.ini` in the model directories. Simply set the model entry to the name of the generated surface and run the viewer executable. The viewer and its support tools are found in the bin directory:

- edtexture: brighten or darken an existing texture map
- imgview: display PANGU texture maps and screen shots (PPM files)
- mergeshadows: combine shadow maps to simulate area light sources
- mkshadows: generate a shadow map for a specific model and Sun position
- mktexture: generate a suitable texture map for the viewer
- pan2pov: convert a `.pan` file into a POV-Ray script
- pandump: display brief details about a .pan file
- pangu_client: control the PANGU viewer remotely using the keyboard arrow keys
- viewer: the main OpenGL file viewer

Sample surfaces and shadow maps are stored in the `samples` directory:

- surf_513.pan: single layer surface generated from a 513x513 DEM
- surf_513.smap: shadow map for surf_513.pan for Sun 6.97e10, 55°, 15°
- texture_surface.ppm: sample DEM for **mktexture**

Dynamic libraries in the lib directory:

- glui.dll: Windows DLL for the **viewer** control panel
- glut32.dll: Windows DLL for the **viewer**, **imgview** and **mktexture** programs
- libglui.so: Linux shared object for the **viewer**

2.2 THE PANGU COORDINATE SYSTEM

PANGU uses a right-handed coordinate system in which the x-y plane is parallel to the section of the surface of the planet being modelled and where the z-axis points upwards towards the zenith (see Figure 2-1). Although the orientation of the x- and y-axes can be whatever the user wishes, we consider the y-axis to point north and the x-axis to point east. The position of the Sun (the primary light source) is always specified using spherical-polar coordinates: we specify its distance and its azimuth and elevation angles.

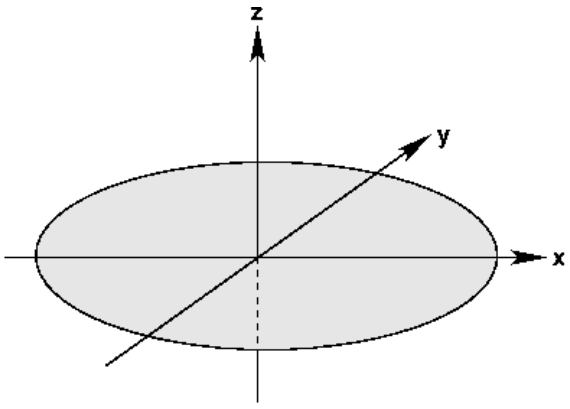

Figure 2-1: the PANGU coordinate system

2.2.1 Azimith/elevation and yaw/pitch angles

Azimuth angles are measured clockwise from the y-axis thus the y-axis is at azimuth 0 and the x-axis at azimuth +90°. Elevation angles are the angle of the object above the x-y plane with the z-axis/zenith at elevation +90°.

In contrast, yaw angles are measured anti-clockwise from the y-axis; pitch angles are positive above the x-y plane just like elevation angles. Figure 2-2 shows a plan view of a free camera placed at the origin and a fixed camera looking at the origin. This shows the meaning of the azimuth and yaw angles as viewed from the origin. Figure 2-3 shows a side view of the same camera configuration and shows the meaning of the pitch and elevation angles (they are the same). The terms free and fixed camera are explained in Sections 3.3.3 and 3.3.4.

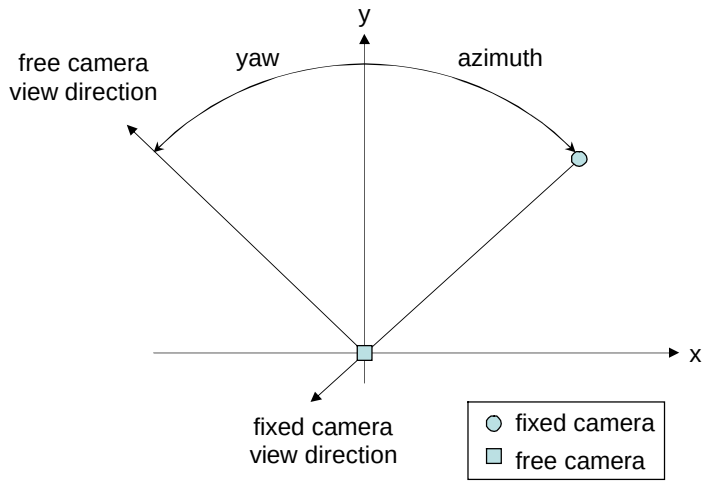

Figure 2-2: camera angles and view directions for the same yaw/azimuth angle (plan view)

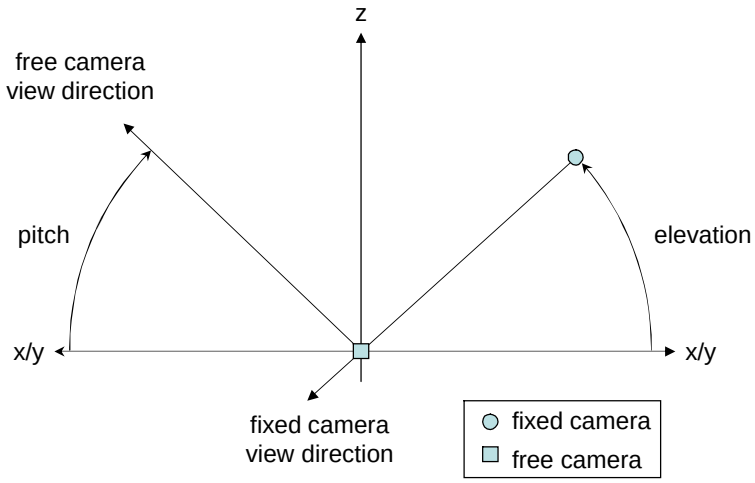

Figure 2-3: camera angles and view directions for the same pitch/elevation angle (side view)

2.2.2 DEMs and the PANGU coordinate frame

DEMs used as the basis of PANGU models are usually represented as bitmap images in which the value of a given pixel corresponds to the elevation of the corresponding point in the model. When the DEM is viewed on-screen as a bitmap the PANGU y-axis points up and the PANGU x-axis pointing right; the PANGU origin is in the centre.

2.2.3 Quaternions

When attitudes are specified using quaternions they define the rotation of the source reference frame relative to the PANGU coordinate frame shown in Figure 2-1. For example, the RADAR emitter is defined such that the beam axis lies along the z-axis of the RADAR emitter frame (which might be the local reference frame of the spacecraft). The unit quaternion represents no rotation so the axes of the RADAR emitter frame are aligned precisely with those of the PANGU frame. This means that the beam axis points upwards along the positive z-axis of the PANGU frame. To

point the RADAR beam axis downwards along the negative z-axis, a rotation of 180° is required about any axis in the x-y plane *e.g.* the x-axis.

A rotation of angle θ about axis $[D_x, D_y, D_z]$ can be described by the quaternion $[q_0, q_1, q_2, q_3]$:

$$q_0 = \cos\left(\frac{\theta}{2}\right)$$

$$q_1 = \sin\left(\frac{\theta}{2}\right)D_x$$

$$q_2 = \sin\left(\frac{\theta}{2}\right)D_y$$

$$q_3 = \sin\left(\frac{\theta}{2}\right)D_z$$

Thus a rotation of 180° about the x-axis [1, 0, 0] is encapsulated by the quaternion [0, 1, 0, 0].

2.3 INTRODUCTION TO PANGU MODELS

PANGU models are usually composed of a single open polygon mesh representing the surface of a terrain with zero or more closed meshes representing boulders resting on or embedded into the terrain. The simplest open terrain meshes are created from a single square digital elevation model (DEM) in which each pixel of the DEM defines the elevation of a vertex of the terrain mesh. However, PANGU terrain meshes are usually created in the form of hierarchy of multiple layers, each layer twice the resolution of the previous one. The centre of each layer in a hierarchy lie along the z-axis of the model. An example of a layered mesh is given in Figure 2-4 while an example of the way that DEMs stack up is shown in Figure 2-5. The DEMs for each layer contain the same number of pixels but represent different physical resolutions. The central area of each DEM in a layered model is usually ignored because those pixels are over-ridden by the contents of the DEMs for the higher resolution layers.

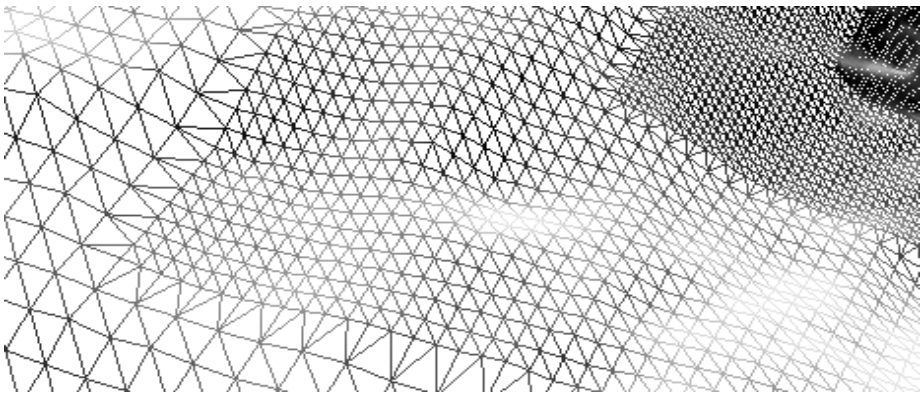

Figure 2-4: multi-layer mesh example

The input DEMs for PANGU terrain meshes can be obtained from external sources such as the MOLA data set (see Section 3.2.3), or generated by PANGU using fractal techniques (see Section 3.2.2). For multi-layered hierarchical models the user normally provides the DEM for the lowest resolution layer of the model and uses PANGU to create the higher resolution layers using fractal expansion and interpolation.

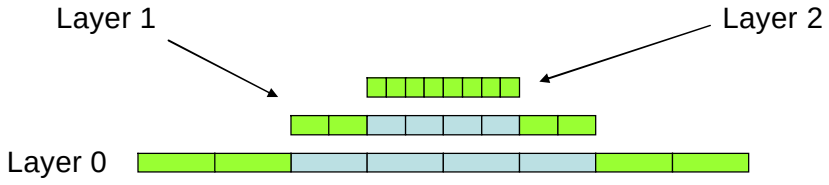

Figure 2-5: DEMs for a layered model (exploded side view)

After creating the terrain DEMs, PANGU can be used to add realistic craters to the model. PANGU can be used to create craters automatically with sizes and ages following user-defined statistical distributions. Alternatively users can add or remove craters by hand or create crater lists using their own external applications. PANGU applies craters to models taking care to match the crater edges to the terrain and ensuring that crater age restrictions are obeyed. For example, old craters are added to the terrain before new ones.

In addition to the normal open or "flat" terrain models, PANGU can create closed models of complete asteroids. These models can be cratered just like the normal terrain models. Asteroid models are created in two phases: the first phase is used to create the basic lumpy, potato-shaped solid while the second phase generates the surface roughness. The creation of asteroid models is described in Section 3.2.9.

2.4 INTRODUCTION TO CRATER MODELLING

PANGU currently models so-called "simple" craters although the modelling itself is not simple. The 3D shape of the crater is computed based on the size and age of the crater and on the shape of the terrain in which the crater lies. This profile is shown in Figure 2-6. The ejecta blanket surrounding the crater is generated fractally to match the roughness of the surrounding terrain while the crater profile and crater plan are also modified using fractal techniques to achieve their realistic finish.

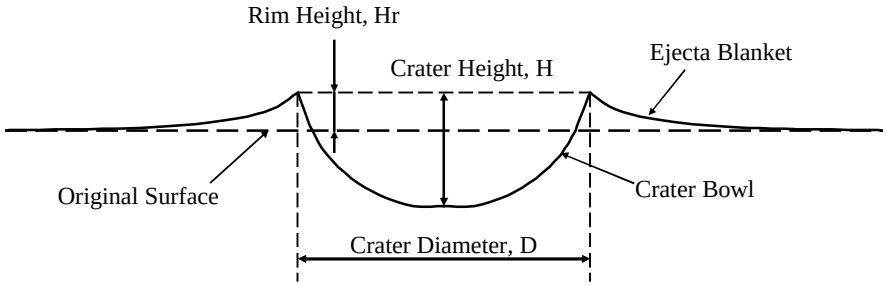

Figure 2-6: PANGU "simple" crater profile

PANGU 2.40 extends the simple crater model shown in Figure 2-6 to support Martian crater profiles. These are the same as craters on rocky bodies such as the Moon and Mercury except that the crater bowl is usually flat due to sand filling the bottom of the crater. An example of this is shown in Figure 2-7. The extent of the filling in can be controlled by a parameter in the relating fill in to crater decay.

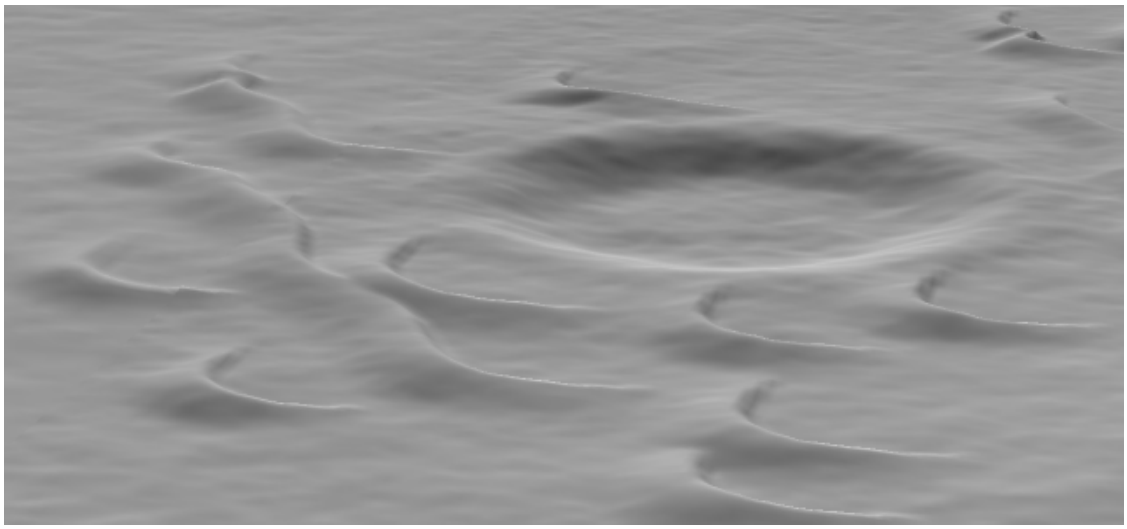

Figure 2-7: Martian craters and dunes generated by PANGU

2.5 INTRODUCTION TO DUNE MODELLING

In addition to modelling Martian craters, PANGU can generate dune fields to model those formed in uni-directional wind regimes. Within these constraints there is a continuum of dune forms from barchan dunes through barchanoid ridges to transverse ridge dunes.

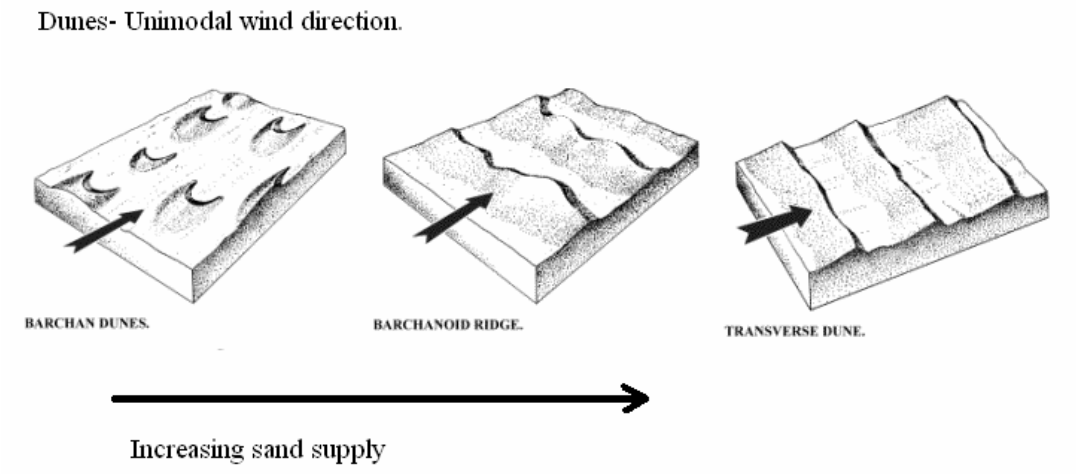

Figure 2-8: examples of sand dunes

A barchan dune field consists of isolated crescent shaped barchan dunes. Within Barchanoid ridge dunes systems it is still possible to distinguish features of Barchan dunes. With the gradual straightening of the barchanoid dunes a transition to transverse dunes occurs. PANGU creates dune fields based on the barchan dune form. The dune field is created by "seeding" the area with randomly placed barchan dunes. In low sand supply regimes a sparse population of barchans is used and a barchan dune field results. With increasing supplies of sand an increasing population of barchans is used. As they are randomly placed there is an increasing chance that individual barchans will overlap. Where this occurs the barchans are merged to produce a barchanoid ridge. This process leads, with increasing densities of initial barchans to the formation of transverse ridge dune systems.

2.6 BASIC LIDAR PRINCIPLES

PANGU 2.40 provides support for modelling LIDAR systems. These are devices for measuring range using narrow beams of light. The LIDAR system modelled by PANGU is quite flexible but it has been tailored towards the needs of the LiGNC project.

The LIDAR detector modelled by PANGU uses an oscillating mirror to control the direction of a laser beam in azimuth and elevation. The reflected light is collected by a detector representing a single pixel of the LIDAR image, or by a small array of detectors representing a super-sampled pixel of the LIDAR image as shown in Figure 2-9. A complete range image of the target is obtained by scanning the beam in a zig-zag pattern across the field of view. The output of the detector(s) is mapped to the corresponding pixel in the result image during the scan. Typically the range images were 100×100 pixels for landing simulations generated at 1 Hz.

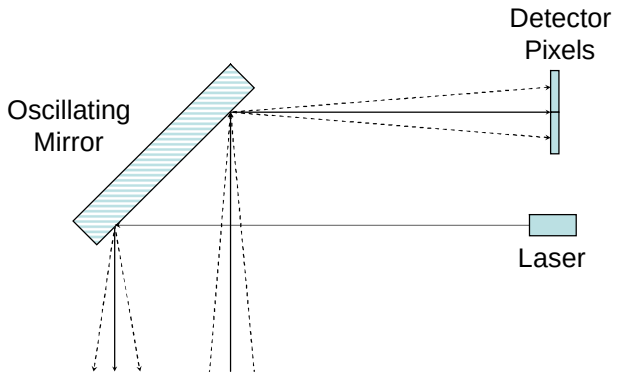

Figure 2-9: LIDAR emitter/detector configuration

Due to the long duration of a scan relative to the likely speed of the craft, PANGU computes the position and attitude of the LIDAR system for every pixel of the result image using linear and angular velocity, acceleration and jerk.

2.7 BASIC RADAR PRINCIPLES

PANGU 2.60 provides support for modelling RADAR altimeters via the TCP/IP interface (see Section 7). A RADAR altimeter is treated as a device which emits a beam of radio waves of one or more frequencies along a single direction (the beam axis) which propagates within a 3D cone with the apex coincident with the RADAR transmitter. This configuration is illustrated in Figure 3-4.

RADAR altimeters are available in two main forms: pulsed and frequency-modulated continuous-wave (FMCW).

2.7.1 Pulsed RADAR

As its name suggests, a pulsed RADAR altimeter operates by emitting a pulse of radio waves towards the target (the surface of the terrain). The pulse is reflected from different parts of the target and the RADAR receiver is used to detect the reflected signal. By measuring the time difference between the emission of the pulse and the reception of the reflection the range to the target can be determined. Even if an infinitely narrow pulse is emitted, the reflection from different parts of the terrain will arrive back at the receiver at different times. This causes the reflection of the pulse to be spread out in time. Since the generation of narrow pulses usually requires lots of energy alternative techniques such as pulse compression can be used to generate wider pulses which can be transmitted using less power and detected with the same benefits as if a narrow pulse had been used.

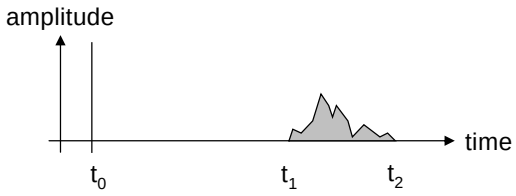

Figure 2-10: a narrow pulse and the response from extended target

Figure 2-10 shows an example of an infinitely narrow pulse emitted at time t_0 with the start of the reflected response from an extended target arriving at time t_1 and finishing at time t_2. The emitted and received pulses are not shown to scale since the amplitude of the reflection is normally much smaller than the amplitude of the transmitted pulse.

2.7.2 FMCW RADAR

The main alternative to pulsed RADAR is FMCW or frequency-modulated, continuous-wave RADAR. Instead of emitting a narrow pulse, FMCW devices emit a signal whose frequency varies (modulated) continuously between two limits. The modulation frequency is normally much lower than the base frequency, perhaps as low as a few Hz compared to the base frequency which would normally be in the GHz region. For a point target, the signal received by the FMCW device will be the same as the transmitted signal but with a phase shift in the modulation as shown in Figure 2-11. The phase shift can be determined by looking at the zero-crossing points in the difference between the transmitted and received signals and can be used to determine the range to the target.

In general, the shift between transmitted and receive signals will be due to both the range to the target (horizontal phase shift) and the relative speeds along the line of sight (vertical frequency shift due to the Doppler effect). The Doppler effect for a point target is shown in Figure 2-12. However, the average frequency difference measured over a complete modulation cycle will eliminate the Doppler effect to give the range; the difference in frequency between successive peaks and troughs in the beat signal can be used to measure the Doppler effect and thus speed.

The situation is complicated further when the target isn't a point. Figure 2-11 and Figure 2-12 show the response from a single point target: multiple point targets will produce multiple return signals each shifted by different amounts while extended targets will return a continuous set of return signals with the shifts all merging together. An example of this is shown in Figure 2-13.

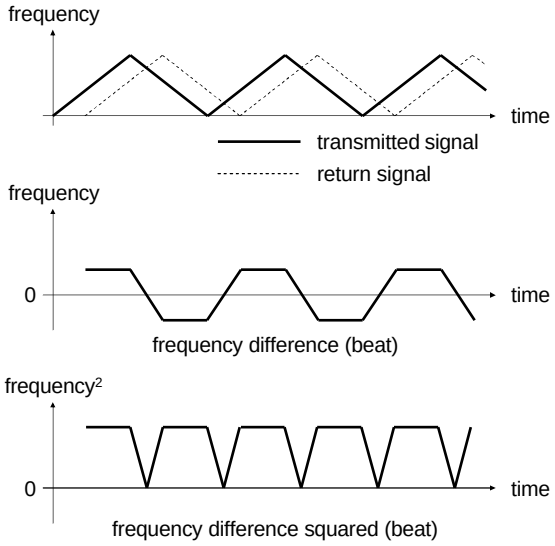

Figure 2-11: continuous wave signal/response, difference and difference squared

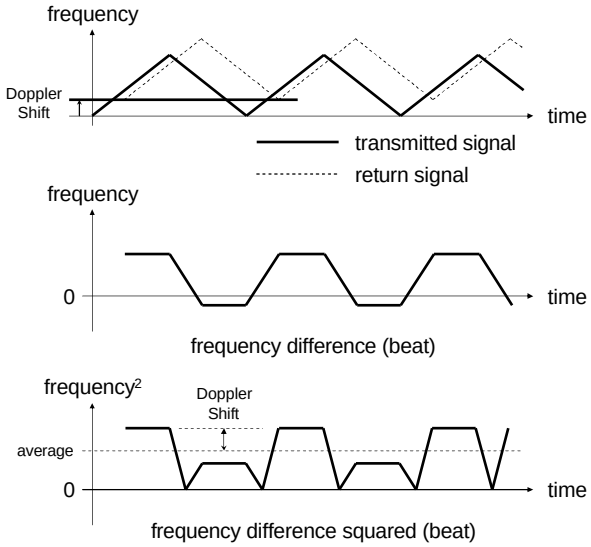

Figure 2-12: effect of Doppler shift on FMCW signals

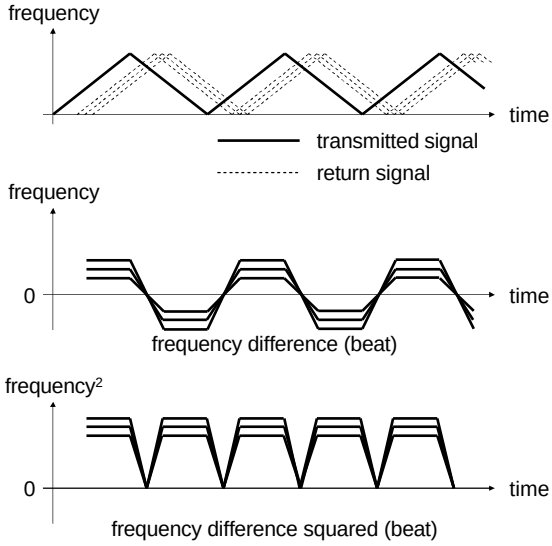

Figure 2-13: effect of extended targets on FMCW signals

2.8 INTRODUCTION TO FOG AND DUST MODELLING

PANGU 2.65 provides support for modelling fog and dust clouds. Two types of fog/dust are provided: global and local-planar. Global fog/dust allows the scene rendered by PANGU to be obscured by fog/dust of a given density using linear or exponential extinction profiles. The global nature of this type of fog/dust cloud means that it is independent of view position and direction. It is intended to be used to improve the realism of views for surface and low-altitude vehicles. Local-planar fog/dust uses an infinite horizontal plane below which lies fog/dust of uniform density with an exponential extinction profile. Above the plane there is no fog/dust. Unlike global fog/dust, the local-planar model only obscures the parts of the terrain which lie below the plane. It is intended to model large dust clouds which extend upwards to a fixed altitude or an atmosphere above a flat surface model.

Both types of fog/dust model need to know the limits of the atmosphere so that the amount of fog projected against the sky can be determined. The atmosphere limit is modelled using a sphere of user-defined radius centred on the view point and is referred to as the sky dome.

Examples of dense global fog/dust and local-planar fog/dust in PANGU are shown in Figure 2-14.

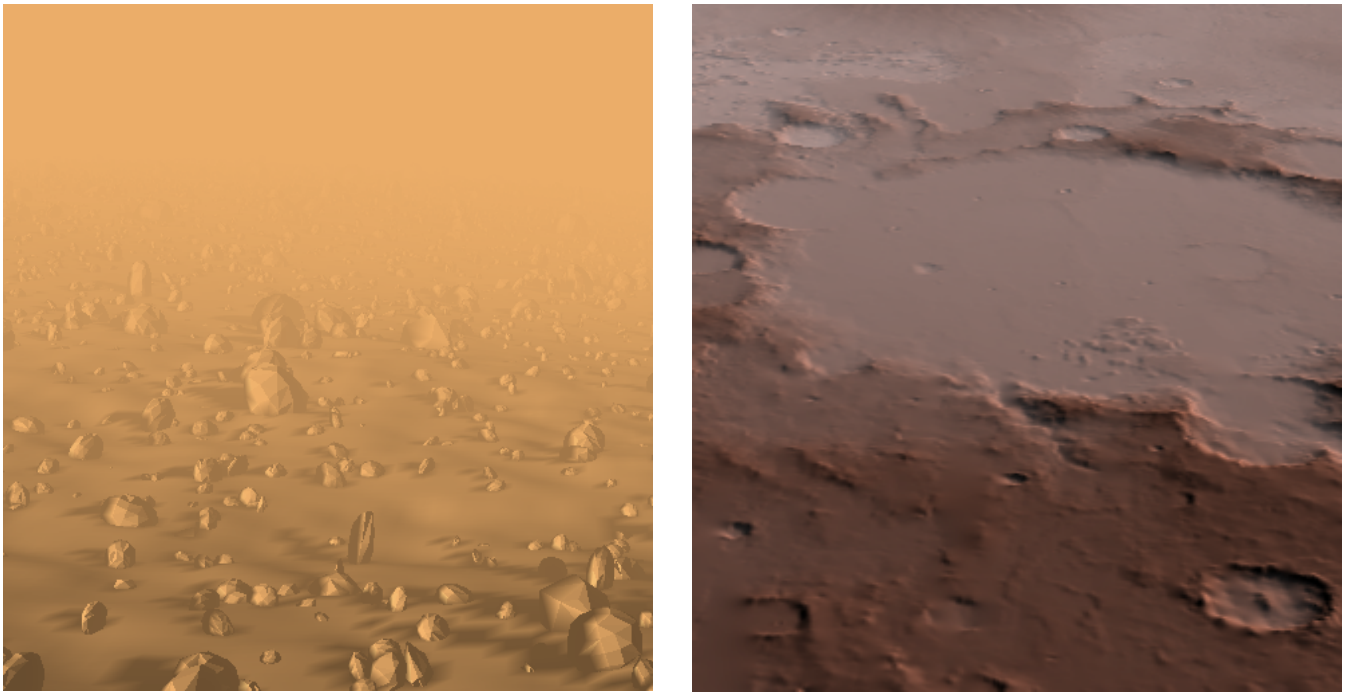

Figure 2-14: dense global fog modelling Titan (left) and planar dust modelling Mars (right)

Global fog/dust clouds are controlled via the `global_fog` INI options. The left image of Figure 2-14 was generated using the following settings:

```
surface.diffuse.colour              0.76 0.56 0.34
boulders.diffuse.colour             0.76 0.56 0.34
viewer.global_fog.enable            true
viewer.global_fog.colour            0.92 0.68 0.41
viewer.global_fog.density           0.00368423
viewer.global_fog.mode              exp
viewer.global_fog.sky_distance      1000000
```

The right image of Figure 2-14 was generated using the following settings:

```
surface.diffuse.colour              0.8 0.5 0.4
sun.colour                          1.5 1.5 1.5
viewer.plane_fog.enable             true
viewer.plane_fog.colour             0.863 0.784 0.784
viewer.plane_fog.density            0.0001
viewer.plane_fog.sky_distance       1000000
```

PANGU 2.70 extended the dust cloud facilities to provide dust-inside-sphere and general dust volumes. The dust inside a sphere facility is intended to provide a model of an atmosphere above a curved planet surface where the sphere matches the radius of curvature of the atmosphere around the planet. The general dust volume facility is intended to provide an approximate model of an arbitrary shaped dust cloud or group of dust clouds.

Due to the increased number of fog/dust cloud types all of which share the same sky distance, PANGU 2.70 uses a new INI option `viewer.fog.sky_distance` to define the sky distance.

3. STANDARD USE

This section describes how to use PANGU to create a hierarchical surface model comprising of a fractal surface with added craters and boulders. This is expected to be the most common use of PANGU. A description of a PANGU hierarchical surface model will be given followed by detailed instructions on how to create one.

The examples in this section assume that the executables **surfacegen** and **featurelistgen** are in a "bin" directory with a path `../../bin` relative to the current `models/testmodel1` directory. This is the structure set up during installation. The commands in this section can either be executed from a command line or by double-clicking on the batch files provided. If a command line is being used, navigate to the `testmodel1` directory prior to issuing PANGU commands.

3.1 HIERARCHICAL MODEL DESCRIPTION

In PANGU, a hierarchical model is generated from a base DEM by creating new layers of increasing resolution by expanding the previous DEM and adding fractal detail. Craters are added to each new surface layer as it is created. Restrictions with PANGU hierarchical surfaces models are:

- An initial DEM must be provided in TGA or pgu (PANGU floating point) format. These may be generated by **surfacegen**.
- Each layer within the hierarchy is represented by one DEM.
- All layers must be defined in TGA or pgu files with the same size dimensions, square with sides of length 2^n+1, n=1,2,3,…
- Each higher resolution layer is twice the resolution of the previous layer.
- Each higher resolution layer is centred within the previous layer.
- To add craters to a hierarchy, a crater list must be specified for each layer.
- To add boulders to a hierarchy, a single boulder list must be specified.
- To view the hierarchy, the individual layers are combined into a polygon file which can then be visualized using the viewer.

3.2 CREATING A HIERARCHICAL MODEL

The following steps are required to generate a new model from scratch.

1. Define a surface layers parameters in a text file. This is `layers.txt` in the `testmodel1` directory.
2. Create, or obtain, a base DEM.
3. Generate crater lists for each surface model layer.
4. Generate a boulder list.
5. Expand the base DEM into the full surface model while adding craters and boulders defined in lists; adding dunes if required.

To generate a layered surface, craters, boulders and additional fractal detail are added to a base DEM. Any base DEM in TGA format could be used as long as the image is square with sides of length 2^n +1, n=1, 2, 3, ... The imported DEM must be converted into 16-bit TGA format and an associated text file must be created. This text file contains information such as image width, height, horizontal resolution and vertical resolution. See Section 3.2.2 for details of how to generate a base DEM using **surfacegen** and Section 3.2.3 for an overview of importing a base DEM from outside PANGU.

3.2.1 Surface layers definition file

This section describes the surface layer parameters text file using the file `layers.txt` as an example. The hierarchy is defined within a text parameter file which specifies input/output files, crater and boulder distributions and the horizontal pixel resolution of each layer. This file is used as an input file to generate crater lists, a boulder list and the hierarchical surface model. The example layers parameter file, `layers.txt` is shown below. This defines a surface model with four layers. Each line is either a parameter definition or a comment. Lines starting with "//" are comments. Parameters descriptions ending with "?" are booleans with 0 denoting false and 1 denoting true.

```
Identifier = PANGU: Surface Layers Parameters File
// Base File Type: 0 = No output, 1 = PanguSurface, 2 = Bitmap, 3 = 16bitTga, 4 = 8bitTga
Base file type = 3
Base file name = surface.tga
Create polygon file ? = 1
Save DEM for each layer ? = 0
Polygon file name = 4_layers.pan
Add crater parameters file name = addCraters.txt
Generate new craters parameters file name = newcraters.txt
Complete craterlist file name = craterlist.txt
Generate new boulders parameters = newboulders.txt
Boulderlist file name = BoulderList.txt
Minimum Crater size (in pixels) = 2
Minimum Boulder size (in pixels. Note can be sub-pixel) = 1
Number of layers including the base layer = 4
Seed random generator? = 1
Random seed [0 – 2^31] = 353
//
// Layer 0 (Base Layer)
X centre position = 0
Y centre position = 0
Horizontal resolution = 8
Fractal Number = 0.8
Height Factor = 0.2
Crater List Filename = layer_0_craterlist.txt
//
// Layer 1
X centre position = 0
Y centre position = 0
Horizontal resolution = 4
Fractal Number = 0.8
Height Factor = 0.2
Crater List Filename = layer_1_craterlist.txt
//
// Layer 2
X centre position = 0
Y centre position = 0
Horizontal resolution = 2
Fractal Number = 0.8
Height Factor = 0.2
Crater List Filename = layer_2_craterlist.txt
//
// Layer 3
X centre position = 0
Y centre position = 0
Horizontal resolution = 1
Fractal Number = 0.8
Height Factor = 0.2
Crater List Filename = layer_3_craterlist.txt
```

The first group of parameters relate to the entire hierarchical surface and specify input/output files, crater distributions and boulder distribution.

The first parameter denotes that this is a PANGU surface layer file. The initial DEM is defined by “Base file type” and “Base file name”. To view the hierarchy as a single object then a polygon file must be created so Create Polygon File should be 1. The next parameter specifies whether individual surface layer DEMs are to be created. This is not normally required. The polygon file parameter will be the output surface model which can be rendered by the viewer.

The “Add crater parameters” file specifies a text file which defines how craters are added to the surface. The “Generate new craters” file specifies the distribution of new craters to generate. The complete crater list file name is not used for generating the hierarchy in this use case, as each layer will be assigned a specific crater list. The boulder distribution is specified in the “Generate new boulders parameters” text file and the boulder definitions are stored in the boulder list file. The number of layers in the hierarchy is then defined.

Following the global hierarchical definitions, six parameters are specified for each layer. In current versions of PANGU the x and y positions should be 0. The horizontal resolution defines (in metres) the distance between

elevation definitions in the DEMs. In the above example, the horizontal resolution of the lowest resolution layer is 8 m. Each subsequent layer *must* have a horizontal resolution half that of the previous layer. The fractal parameters of each layer can be specified. This can be used to varying the "roughness" of the surface at different resolutions. A specific crater list file is defined for each surface layer.

3.2.2 Creating a base DEM.

This section describes how to create a base DEM. The `newsurface.bat` batch file uses the **surfacegen** command to generate an initial DEM using the parameters stored in `newsurface.txt`. The command to generate a base DEM is:

```
..\..\bin\surfacegen –f newsurface.txt
```

This command is stored in `newsurface.bat`.

The surface generated is defined using the parameters in `newsurface.txt`. Edit these values to alter the base DEM that will be created.

- Select an output file type: choose type 3 (Targa/TGA).
- Choose a name for the output file.
- The magnitude parameter defines the surface resolution: for fractal types 0, 1 and 2 the DEM will be a square with sides subdivided into $2^{magnitude}+1$ elements. For fractal type 3 the sides have $2^{magnitude}$ elements. Note that the number of subdivisions in the DEM will be used in other parameter files for area sizes *etc*. The three fractal techniques produce similar results. Type 0 is Random Mid-Point Displacement, Type 2 is Random Mid-Point Displacement with Successive Random Additions and Type 3 is Spectral Synthesis.
- If "Fix Height Resolution?" is 0 then the heights in the DEM will be adjusted so that they span the full 16bit range of a pseudo-greyscale Targa file. If it is set to 1 then each height unit of the DEM will correspond to "Height Resolution" units of the surface.
- The horizontal resolution parameter defines the resolution of each layer within a hierarchical model. This should be set to the same value as for layer 0 in the `layers.txt` file.
- Choose the fractal type to be used to generate the surface. Types 1 and 2 generate surfaces with various heights while type 3 generates surfaces that are up to "Height Range" units high. "Scale to Height Range?" is generally 1 for fractal types 0-2 and 0 for type 3. If the generated surface is too rough with height-range scaling enabled, try decreasing the height range or increasing the magnitude.
- The random number generator can either be seeded automatically by the program or by setting "Seed Random Numbers?" to 1 and setting "Seed" to the chosen seed. This can used to generate the same base DEM each time the program is run. The random seed should be between 0 and 32767.
- Information about the DEM will be written to a file if "Create associated surface info file?" is set to 1. If the DEM is saved as surface9.tga then the surface information file will be called surface9.tga.txt. This should be set to 1 for all PANGU surface models.
- To create a single resolution surface model the "Number of Layers" should be 1. If this parameter is greater than 1 then multiple DEMs of varying resolution will be generated. Each higher resolution layer will be centred in the previous layer and twice its resolution. This value should be set to 1 as further layers will be created when adding craters to the surface.

3.2.3 Importing a base DEM from outside PANGU

To import a DEM from outside PANGU the user must convert the DEM into 24-bit Targa bitmap (TGA) format and create an appropriate surface information file. If the TGA-format DEM is stored in the file *surface*.tga then the surface information file must be called *surface*.tga.txt. The surface information file describes the DEM width, height and horizontal resolution. Note that the TGA image must be square with sides of length 2^n+1, n=1, 2, 3, ...

To produce a TGA-format DEM, each signed floating-point DEM height value must be encoded as a 16-bit integer and stored in the red and green components of a pixel in a 24-bit colour TGA file. If the floating point height sample is *H* and the DEM height resolution is *Z* then the 16-bit encoded integer *N* is defined by:

$$N = \frac{H}{Z} + 32768$$

The top 8 bits of this value (*N* quo 256) are stored in the red component of the TGA pixel while the bottom 8 bits (*N* mod 256) are stored in the green component. The blue component is ignored.

Note that although **surfacegen** only accepts TGA files it may be easier to convert the raw DEM data into a 24-bit colour bitmap format such as a PPM or BMP using the encoding scheme described above and then use a graphics program such as PaintShop to convert the bitmap into a 24-bit TGA file.

3.2.3.1 Surface (DEM) information files

As described above, each TGA file *surface*.tga must be accompanied by a valid surface information file *surface*.tga.txt describing the original DEM and its encoding. An example of this file created by **surfacegen** is given below:

```
Identifier = PANGU: Surface Info File
// Surface File Type: 0 = No output, 1 = PanguSurface, 2 = Bitmap, 3 = 16bitTga, 4 = 8bitTga
Surface file type = 3
Surface file name = surface.tga
Magnitude = 9
Length (xsize) = 513
Breadth (ysize) = 513
Fix Height Resolution? = 0
Height Resolution = 0.000906817
Horizontal Resolution = 8
Fractal Number = 0.8
Height Factor = 0.2
Height Range = 59.4292
```

The surface file type must be 3 for this version of PANGU and the file name is the name of the TGA file containing the encoded DEM. The magnitude parameter defines the dimensions of the DEM ($2^{magnitude} + 1$) and the length and breadth parameters must match the image dimensions. If appropriate, the height resolution can be fixed by setting "Fix Height Resolution?" to 1 and defining the floating point height resolution on the following line (the *Z* parameter of Section 3.2.3).

The horizontal resolution defines the physical distance between DEM samples in the horizontal direction while the fractal number and height factor parameters should match the properties of the surface being imported. The fractal number is used to control the "roughness" of craters added to the surface and significant differences may produce unrealistic models. The height range defines the difference between the largest and smallest floating-point DEM value.

3.2.4 Generate crater lists for each surface model layer

This section describes how to generate crater lists for a hierarchical surface using `layers.txt` as the example surface layers definition file. The command to generate crater lists for a hierarchical model is:

```
../../bin/featurelistgen –l layers.txt
```

This command is stored in `NpCraterList.bat`.

This command will generate a crater list for each layer specified in `layers.txt` and update `layers.txt` to include the names of the new crater lists generated.

- The "generate new craters" file specifies the distribution of new craters to generate. In this example this is "newcraters.txt". Edit this file to alter the crater lists generated.
- The random generator can be seeded to reproduce identical crater lists if required. If it is not seeded then a seed based on the PC's clock will be used to give an unknown seed.

- The "Number of craters to generate in list" parameter is used in two ways. If either of the "low diam" or "high diam" parameters are zero then the "numbers of craters to generate" parameter will specify the number of craters generated. If "high diam" is greater than "low diam" and both are greater than zero then a total number of craters generated will be calculated so that each square kilometre will have on average "Number of craters generated" between low and high diameter.
- The "Horizontal scale" parameter scales the crater (x, y) position and diameter values. It is recommended that this value is left at 1. The "X Offset" and "Y Offset" values specify the origin offset around which the crater positions are referenced. The "Area X" and "Area Y" size specify the area into which randomly generated craters are distributed.
- The output crater list is not used in this instance. It defines the crater produced for a single resolution model.
- The crater diameter distribution filename specifies a pre-defined crater diameter distribution which defines the diameters of randomly generated craters.
- The Crater age distribution filename specifies a file containing distribution of crater ages.
- The Crater decay filename specifies a file relating crater age to crater depth and rim height degradation.

3.2.5 Generate a boulder list.

This section describes how to generate a boulder list for a hierarchical surface using `layers.txt` as the example surface layers definition file. The command to generate a boulder list for a hierarchical model is:

```
../../bin/featurelistgen –a layers.txt
```

This command is stored in `NpBoulderlist.bat`.

This command will generate a single boulder list which can be generated from a random distribution and/or by relating boulder positions to craters. The output boulder list file is defined in `layers.txt`.

- The "generate new boulders" parameter specifies the distribution of new boulders to generate. In this example this is "newboulders.txt". Edit this file to alter the boulder list generated.
- The output boulder list parameter is not used in this use case. It specifies the boulder list output file when creating a boulder list for a single resolution surface model.
- The random generator can be seeded to reproduce identical boulder lists if required. If it is not seeded then a seed based on the PC's clock will be used to give an unknown seed.
- The "Horizontal scale" parameter scales the boulder (x, y) position and diameter values. It is recommended that this value is left at 1. The "X Offset" and "Y Offset" values specify the origin offset around which the boulder positions are referenced. The "Area X" and "Area Y" size specify the area into which randomly generated boulders are distributed.
- Boulders can be distributed generally and/or relative to craters as specified by the two booleans, "Use general distribution" and "Use crater list".
- If a general distribution of boulders is specified then they are generated using the number of boulders to add, and their size and depth distribution.
- The rest of the parameters define how boulders are distributed in relation to craters. "Boulder maximum size graph filename" describes a graph of boulder size relative to the crater size. The "Boulder size distribution" describes the probability of a particular boulder size (relative to crater size) occurring. "Boulder minimum depth graph filename" describes a graph of the minimum depth at which a boulder might be buried (relative to the surface) relative to the age of the crater. The "Boulder depth distribution" describes the initial depth distribution for craters of age 0. "Boulder radial distribution filename" describes the probability of finding a boulder at a normalised radial distance from the crater centre. The rim of the crater has normalised radial distance of 1. "Boulder frequency" is the number of boulders relative to crater diameter while the "Boulder frequency distribution" describes the number of boulders to generate for a particular crater.

3.2.6 Expand the base DEM into the full surface model

The command to generate the hierarchical model is:

```
../../bin/surfacegen –l layers.txt
```

This command is stored in `expandsurface.bat`.

This will create polygon mesh file (`.pan`) which can then be visualised using the dedicated viewer. To curve the surface to match the radius of curvature of a particular planet a `PlanetProperties.txt` file must be created (see Section 5.2.7). Then the hierachical model may be created using the command

```
../../bin/surfacegen –P PlanetProperties.txt –l layers.txt
```

To create a .pan file which can be rendered using the ROAM algorithm the –R option must be used:

```
../../bin/surfacegen –R –P PlanetProperties.txt –l layers.txt
```

3.2.7 Expand the base DEM into the full surface model adding dunes

The use of the dune option requires the inclusion of a dune parameters file in the model directory. The example given is `testmodel4/dunemodel.txt`.

The command to generate the hierarchical model with dunes is:

```
../../bin/surfacegen –u dunemodel.txt –l layers.txt
```

This command is stored in `testmodel4/expandsurface.bat`.

This will create polygon mesh file (`.pan`) which can then be visualised using the dedicated viewer.

Planetary curvature and ROAM model options are as per section 3.2.6.

3.2.8 Creating larger models

The simplest method to create new surface models of different size and resolution is to create a copy of the `testmodel1` directory for every new surface model before altering the parameter files. To create a larger, more realistic, surface model increase the base surface size (i.e. to magnitude 10 which gives a 1025 x 1025 DEM) and increase the number of layers in the surface layers parameter definition file (i.e. to 8). This will create a larger more realistic model but will take longer to create and render. It is important to remember to set the horizontal resolution values correctly in the surface layers parameter definition file. For an eight layer model, a recommendation for the horizontal resolution values would be 32, 16, 8, 4, 2, 1, 0.5 and 0.25 in order of layer definition. Using a 1025 x 1025 base model this would create a hierarchical model of a square area of 32 * 1025 m (32 km) in width with the highest resolution layer modelled to 25 cm resolution.

3.2.9 Creating complete asteroid models

A complete asteroid model is created in a series of stages that can be controlled using parameters. First a spherical model is generated. Then surface roughness is added using two stages of Poisson Faulting, a standard technique for adding fractal detail to spherical objects. The first stage of Poisson Faulting produces an excessively rough object using large faults. This is then smoothed into a smooth but irregularly shaped object. A second stage of Poisson faulting adds surface roughness using small faults. Poisson faulting is a slow algorithm and generating a high resolution asteroid can take many hours so a facility is provided to save the model at this stage.

The asteroid is then distorted along each of the three axes. This allows ellipsoidal asteroid models to be created. A crater list can then be added to the asteroid model if desired. This should be a `CraterList` where the x- and y-positions of craters define their latitude and longitude respectively in degrees. Latitude values should be in range –90 º to 90 º and longitude between 0 º and 360 º. Running **featurelistgen** with option `–s` will randomly generate a `CraterList` for an asteroid.

Finally the asteroid model is saved as a PANGU object file (`.pan`) which can be displayed in the **viewer**.

3.2.9.1 Creating asteroid craterlists using featurelistgen

To create a `CraterList` for an asteroid, run **featurelistgen** with option `-s`. This randomly generates craters in the usual way using crater diameter and age distributions. The differences are that the area and offset parameters are ignored (the craters are generated over the whole asteroid) and that only an absolute number of craters can be specified. This means that you cannot specify a number of craters within a specified diameter range per km^2.

```
../../bin/featurelistgen –s newcraters.txt
```

3.2.9.2 Creating the asteroid model using surfacegen

To create an asteroid model run **surfacegen** with option `-s`

```
../../bin/surfacegen –s asteroidparams.txt
```

The asteroid parameters text file defines the creation of the asteroid model. There are two possible outputs: a `.pan` file which can be displayed using the **viewer** and a `.mod` file which is an intermediate asteroid model file that defines height values for each latitude and longitude. Support for this intermediate file was created because the Poisson faulting on a high-resolution model can take many hours but the resultant `.mod` file can be quickly loaded from file. This allows different `CraterList` files and stretching parameters to be applied quickly to a high-resolution model that was created previously.

The asteroid parameters file is described in detail in Section 5.3.

3.2.10 Creating whole-planet models

A whole planet model consists of a perfect sphere represented as a user-defined number of triangular patches. The radius of the sphere is also provided by the user along with the name of a texture file which can be used to improve the visual appearance of the model when rendered. The complete planet model will be written to a PANGU object file (`.pan`) which can be displayed in the usual way using the **viewer**.

Whole-planet objects are constructed by dividing a sphere into a number of rings or disks along equally-spaced lines of latitude. Two special “rings” are used for the north and south pile while all other rings are like a strip of paper joined at the ends. Each ring is subdivided into a number of quadrilaterals (quads) based on lines of longitude and each quad is constructed from two triangles. This system means that whole-planet objects can be rendered by the **viewer** just like the other surface meshes produced by **surfacegen**. Additionally all the existing shadow-casting and false-colour imaging facilities automatically work with whole-planet models.

3.2.10.1 Creating the whole-planet model using surfacegen

To create a whole-planet model run **surfacegen** with option `-w`

```
../../bin/surfacgen –w WholePlanet.txt
```

The whole-planet parameters text file defines the creation of the whole-planet model and is described in Section 5.4.

3.3 VIEWING A MODEL

The viewer is the OpenGL visualisation tool of the PANGU suite and is supported by several additional tools for producing shadow maps and texture maps. Basic features of the viewer will be introduced first before moving onto advanced topics such as the creation of shadow maps, flight paths and movie stills. The introduction assumes that you will be using the program with Windows Explorer or the Linux equivalent such as GNOME or KDE (i.e. not from a DOS prompt or UNIX shell). The viewer and tools can be run from the command line and work in a similar manner.

3.3.1 Preparation

Before attempting to use the viewer you must ensure that it will be able to find an INI file and that any filenames in the INI file match those on your system. The default INI file for the viewer and tools is `pangu.ini`: for this introduction it must be stored in the current model directory.

Almost every error and information message from the viewer will be appended to the file specified in the **log_errors** setting of `pangu.ini`: you must ensure that the viewer can write to this file and that it doesn't contain important information! The default is `PANGU.log` in the current directory. If you encounter a problem when using the viewer or any of the associated tools you *must* examine messages at the end of the log file for explanations.

Next you must ensure that the **model** setting in `pangu.ini` corresponds to the name of the PANGU object file that you want to view. If the filename contains spaces then the name *must* be enclosed in quote marks e.g. "An Example.pan". The suffix (e.g. .pan) must always be specified if it is present on the original file. Also note that Linux is case-sensitive so care must be taken with filenames and section names.

3.3.2 Basic interaction

Launch the viewer by double-clicking on the `viewer.bat` file. There may be a long delay with lots of hard disk and processor activity before the main viewer window appears. Under Windows a DOS window ought to appear quickly and will remain on the screen until the application terminates: if it vanishes before the viewer window appears then check the error log (e.g. `PANGU.log`) for an explanation.

Click on the standard window close button to terminate the viewer. When the control panel is visible (see below) the Quit button will also terminate the viewer. When the main rendering area is active the ESCAPE and Q keys can be used to terminate the application.

3.3.3 Model view

By default the viewer uses a model-based view rather than a craft-based view. In the model-based view the camera is fixed to one end of a rigid rod with the other end (the target) fixed in space (ideally in the middle of the model being viewed). Changes in the azimuth and elevation angles rotate the rod on which the camera is attached leaving the target fixed in position. Changes in distance affect the length of the rod while changes in position move the target end of the rod dragging the camera with it. The camera always looks at the target. Mathematically the model view mode uses the spherical polar coordinate system with the target at the origin.

3.3.3.1 Using the control panel

If the control panel is visible at the right of the viewer then the controls inside it may be used to change the camera alt-azimuth angle, its distance from the target and the position of the target. Simply click in one of the text boxes and change the value. Alternatively clicking with the left mouse button on the small up and down arrows beside the text boxes changes the value stored in the text box. For continuous changes, click on the small up and down arrows and then move the mouse up or down while keeping the mouse button down. The SHIFT and CONTROL keys can be used to adjust the rate of change.

3.3.3.2 Using the keyboard

Ensure that the main viewing window is active by clicking somewhere on the current scene. Use the left and right arrow keys on the keyboard to change the azimuth angle of the camera; use the up and down arrows to change the elevation angle. Press the Z key to move the camera closer to the target and SHIFT-Z to move away from it. To move the target around the scene, press the TAB key once. Now the left and right arrow keys move the target horizontally around the surface while the up and down arrow keys move it towards and away from the viewer. Pressing U or SHIFT-U will move the target up or down. Press the TAB key again to use the arrow keys for changing the camera angles. To roll the camera, press R or SHIFT-R.

3.3.3.3 Using the mouse

While the pointer is over the main viewing window, click and hold down the left mouse button. Moving the mouse horizontally while keeping the left mouse button down (dragging) will change the azimuth angle; dragging vertically will change the elevation angle. Dragging diagonally will change the azimuth and elevation angles at the same time so be careful! Dragging with the right mouse button will move the target around a horizontal plane; dragging vertically with the middle mouse button will change the zoom.

3.3.3.4 Using the pangu.ini file or the command line

The initial camera position and orientation can also be set in the `pangu.ini` file or specified as command line options. The **fixed_camera** setting in `pangu.ini` defines the position of the target, the distance from the target and the azimuth and elevation angles of the camera relative to the target. To start with the target at (100, -200, 40) and with the camera at a distance of 300 units, azimuth 180 degrees and elevation 45 degrees the `pangu.ini` setting is:

```
fixed_camera     100 -200 40    300 180 45
```

Alternatively the following command line options could be used:

```
-target 100 -200 40     -camera 300 180 45
```

3.3.4 Craft view

The craft view (see the **view_mode** setting in `pangu.ini`) is trickier to use but provides first-person camera interaction using yaw, pitch and roll. Changes in yaw rotate the camera in the horizontal plane towards the left or right; changes in pitch rotate the camera in a vertical plane up or down. We refer to this as a free camera.

3.3.4.1 Using the control panel

The same technique as the model-based view works for the craft-based view. This time the camera attitude (yaw, pitch and roll) and camera position (rather than the camera target) can be changed.

3.3.4.2 Using the keyboard

As before, ensure that the main viewing window is active by clicking somewhere on the current scene. The left and right arrow keys rotate the camera towards the left or right respectively. The up and down arrow keys rotate the camera up or down. Pressing TAB allows the camera to be moved: the left and right arrow keys move the camera to the left and right, the up and down keys move the camera forwards for backwards along the current view direction (which may cause the camera to fly away from the surface or crash into it)! Press the TAB key again to rotate the camera. As before the R and SHIFT-R keys change the roll.

3.3.4.3 Using the mouse

While the pointer is over the main viewing window, click and hold down the left mouse button. Horizontal drags will rotate the camera to the left or right while vertical drags change the pitch. Horizontal drags with the right mouse button will move the camera towards the left or right. Vertical drags with the right mouse button will move the camera forwards or backwards along the current view direction. Vertical drags with the middle mouse button will move the camera up or down. Note that when the camera pitch is -90 degrees (looking directly down on the surface) then the up direction is parallel to the surface. This means that vertical drags with the middle mouse button will move the camera across the surface!

3.3.4.4 Using the pangu.ini file or the command line

Just like the model view, the initial camera position and orientation can also be set in the `pangu.ini` file or specified as command line options. The **free_camera** setting in `pangu.ini` defines the position of the camera and its yaw, pitch and roll angles. To start with the camera at (100, -200, 40), yaw of 10 degrees, pitch of -90 degrees and zero roll the `pangu.ini` setting is:

```
free_camera      100 -200 40    10 -90 0
```

Alternatively the following command line options could be used:

```
-position 100 -200 40     -attitude 10 -90 0
```

3.3.5 Unexpected or unrealistic views

Since the PANGU viewer does not place any restrictions on the position or orientation of the camera, it is possible to generate views in which the camera is placed “underneath” or “inside” the model which appears to have pieces cut

from it (see Figure 3-1). This often occurs in the model-view mode when pitch angles are close to zero. The cut-away effect is due to OpenGL ignoring any part of the model that is closer to the viewpoint than the value of **viewer.near_distance** in `pangu.ini` (this parameter must be greater than zero). Note that OpenGL will also ignore any part of the model which is further away from the viewpoint than the value of **viewer.far_distance** in `pangu.ini`.

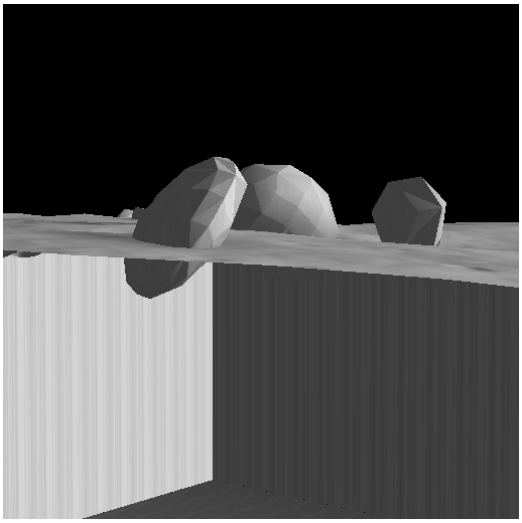

Figure 3-1: example of an invalid camera position

Occasionally views at low pitch angles appear to show "floating" boulders: this is due to the fact that boulders are placed at random orientations and may appear balance in ways which defy nature! However, boulders are always embedded into the surface of the model and do not actually float. Boulders are intended to be viewed from higher elevations where the "floating boulder" effect is not noticeable.

3.3.6 Basic INI options for the viewer

Previously we introduced the **model** and **view_mode** settings for `pangu.ini` but there are a few other settings which users may need and these are described below. In the examples, the notation X | Y means that the option value can be either X or Y; parameters in italics are explained in the text. These settings should be placed in the [_defaults] or [viewer] sections of `pangu.ini`. An example `pangu.ini` file is given in Section 8.5 and more advanced options are described in Section 6.1.2.

3.3.6.1 Model and shadow map options

model *fil*

viewer.model *fil*

This option instructs the viewer to load the model from the file *fil* unless the user specifies an alternative model on the command line.

shadowmap *fil*

viewer.shadowmap *fil*

Computing cast shadows for surfaces with more than a few thousand vertices is extremely time consuming and we are forced to use an offline shadow map generator to compute shadows for a particular model and specific Sun position and attitude. The **mkshadows** and **mergeshadows** tools are provided for this purpose (see below). The shadow map file *fil* must correspond to a shadow map generated from the model that the viewer will display. Otherwise the viewer application may terminate abnormally shortly after starting.

viewer.ignore_hashcode true|false

Each shadow map contains a unique hash code number which identifies the model from which the map was created. If the hash code stored in a shadow map file does not match the corresponding code in the model then

the viewer will refuse to load the shadow map. The user may force the viewer to load the shadow map by setting this parameter to true. However, doing so may cause the viewer to terminate abnormally.

3.3.6.2 Camera properties

fixed_camera *x y z d azi alt*

The model-view (fixed) camera will be positioned at a distance *d*, azimuth angle *azi* and elevation angle *alt* relative to the target at (*x*, *y*, *z*).

free_camera *x y z a p r*

The craft-view (free) camera will be positioned at a (*x*, *y*, *z*) with yaw angle *a*, pitch angle *p* and roll angle *r*.

viewer.field_of_view *fov*

The horizontal field of view width will be *fov* degrees.

3.3.6.3 User interface properties

angle_step *f*

Changes in yaw, pitch, azimuth and elevation angles using the keyboard will be made in steps of *f* degrees.

full_screen true | false

viewer.full_screen true | false

If true, the viewer will occupy the whole screen rather than use a window.

move_step *f*

Changes in position using the keyboard will be made in steps of *f* units.

mouse_rotate_scale *f*

Changes in yaw, pitch, azimuth and elevation angles using the mouse will be made in steps of *f****angle_step** degrees for each pixel of mouse movement. This is allows finer movement control compared to the keyboard.

view_mode model | craft

Tells the viewer to start in model-view or craft-view mode.

viewer.report_keys true | false

If this setting is true then the viewer will display a message describing the effect of each key when pressed. For example, pressing S will display the message "s: save image to file". This allows users of slow machines to know what is happening and can be used to educate new users. The message will disappear when the next view is rendered and so some messages may not be visible for very long.

window_size *width height*

When not in full-screen mode the rendering area will be *width* pixels by *height* pixels in size.

3.3.6.4 Illumination and surface properties

ambient *r g b*

viewer.ambient *r g b*

This defines the red, green and blue (RGB) intensity of ambient light as *r*, *g* and *b*. Every part of the scene receives this amount of light even if it is totally shadowed. It is used to model general light scattering.

boulders.diffuse.colour *r g b*

The fraction of RGB intensities reflected by boulder surfaces under the Lambert model is *r*, *g* and *b*.

sun.colour *r g b*

Define the RGB intensity of the Sun as *r*, *g* and *b*. If any part of the surface directly faces the Sun then the colour in the rendered image under the Lambert model will be the product of **sun.colour** with **surface.diffuse.colour** plus the product of **ambient** with **surface.diffuse.colour** clamped to the range [0, 1].

sun.position *r t p*

This defines the Sun position in spherical polar coordinates: distance *r*, azimuth angle *t* and elevation angle *p*.

surface.diffuse.colour *r g b*

The fraction of RGB intensities reflected by the surface under the Lambert model is *r*, *g* and *b*.

3.3.6.5 Rendering properties

do_boulders true | false.

If true, boulders will be displayed. Use the B key to enable/disable boulders at runtime.

smooth_shadows true | false.

viewer.smooth_shadows true | false.

If true the viewer will attempt to anti-alias shadow edges. ***This option should not be used with models generated by PANGU 1.50 or later***.

3.3.6.6 Other options

log_errors *f*

All error and information messages will be written to the file *f* (e.g. `PANGU.log`).

3.3.7 Generating shadow maps

Shadow map generation is very time consuming and may take hours or even days for larger multi-layer models. As a result the user must manually create shadow maps for each model they wish to view and for each Sun position. The **mkshadows** program is used to generate these shadow maps and for this example we will use the command line. To generate a shadow map "output.smap" for the model stored in "output.pan" for the Sun distance of Mercury aphelion and angular position 55 degrees azimuth and 15 degrees elevation:

```
../../bin/mkshadows -sun 6.97e10 55.0 20.8 -o output.smap output.pan
```

Since this may take a while a time limit of 600 seconds (for example) can be specified:

```
../../bin/mkshadows -shadowtime 600 -sun 6.97e10 55.0 20.8 -o output.smap output.pan
```

To view the model with this shadow map:

```
../../bin/viewer -shadowmap output.smap output.pan
```

or update the **shadowmap** `pangu.ini` setting as described above. Provided all or most of the shadow map has been computed the difference ought to be striking.

3.3.8 Area light sources

Extended or area light sources produce shadows with two components: the dark inner umbra where none of the light source is visible, and the lighter penumbra where part of the light source is visible.

3.3.8.1 Generating Perfect Penumbra

One technique for generating a true penumbra is to compute the percentage of the light source which is visible at a given point on the surface being shadowed. This percentage can be used to determine how dark the shadow is. However, computing the visible area of a disc occluded by an arbitrary polygonal model is not easy and is computationally expensive.

3.3.8.2 Approximating Penumbra

An alternative representation of an area light source is a group of many point light sources: the shadow percentage for any point is the average of the contributions of every point source. Luckily it turns out that only a small number of point sources across the area source need to be sampled to obtain a realistic shadow percentage.

Clearly the larger the number of samples the better the more accurate the penumbra produced but this must be balanced against the fact that our shadow maps are sampled at each vertex. If the angular separation of vertices when viewed from the light source is larger than the angular size of the light source viewed from those vertices then the penumbra cannot be displayed.

3.3.8.3 Using mergeshadows

To simulate area light sources in PANGU the user must generate a set of shadow maps with the Sun in slightly different positions. The maps can be joined into a single map using the **mergeshadows** tool. This gives the user complete freedom over the number of times the area light source is sampled and the distribution of the samples.

For example, if the Sun is at 55.0 degrees azimuth and 20.0 degrees elevation then the following five shadow maps ought to be sufficient to simulate a Sun with angular size 1.6 degrees:

	55.0, 20.8	
54.2, 20.0	55.0, 20.0	55.8, 20.0
	55.0, 19.2	

To generate a shadow map “output.smap” for the model “output.pan” for the Sun positions above using the Sun distance of Mercury aphelion one could use the following commands:

```
../../bin/mkshadows -sun 6.97e10 55.0 20.8 -o m0.smap output.pan
../../bin/mkshadows -sun 6.97e10 54.2 20.0 -o m1.smap output.pan
../../bin/mkshadows -sun 6.97e10 55.0 20.0 -o m2.smap output.pan
../../bin/mkshadows -sun 6.97e10 55.8 20.0 -o m3.smap output.pan
../../bin/mkshadows -sun 6.97e10 55.0 19.2 -o m4.smap output.pan
../../bin/mergeshadows -o output.smap m0.smap m1.smap m2.smap m3.smap m4.smap
```

3.3.8.4 Using mkshadows

The **mkshadows** tool has been updated to enable area light sources to be simulated automatically. This can be used to avoid creating and merging multiple shadow maps. To use this feature you must pass the `-area_light` and `-rings` options to **mkshadows**, specify the number of samples to generate via the `-samples` option and define the physical radius of the light source. The tool will use these options to simulate to simulate an area light source using rings of evenly distributed point light sources. Alternatively set the **sun.sample_method** INI option to “rings”.

To generate a shadow map “output_area_40.smap” for the model “output.pan” for the Sun (radius 6.96265×10^8 m) at Mercury aphelion with 40 samples one could use the following command:

```
../../bin/mkshadows -sun 6.97e10 55.8 20.0 -radius 6.96265e8 -area_light
   -rings -samples 40 -o output_area_40.smap output.pan
```

Alternatively to reduce the amount of typing the Sun position, radius, number of samples, output shadow map name and input model name can be specified in the [mkshadows] section of `pangu.ini`:

```
[mkshadows]
    sun.position        5.318769e10 234.4 30.0
    sun.radius          6.96265e8
    sun.samples         40
    sun.sample_method   rings
    model               output.pan
    shadowmap           output_area_40.smap
    overwrite           true
```

With these settings the command line becomes:

```
../../bin/mkshadows
```

Note that taking *N* samples of a light source will normally increase the time that **mkshadows** takes to produce a shadow map by a factor of about (but not more than) *N*.

3.4 NEW FEATURES

This section describes the use of new features that have been added to PANGU since the original 1.00 release. Please refer to Section 11 for a complete record of the changes made to PANGU with each release.

3.4.1 LIDAR Simulation (introduced with PANGU 2.40)

The **viewer** has been extended to support the generation of data for a LIDAR instrument used by the LiGNC project. This consists of a LIDAR emitter which performs a zig-zag scan of the laser beam over a rectangular field of view. The LIDAR detector consists of a small array of pixels (*e.g.* 2×2) which collect the response for each scan position and store the result in a separate image memory. Figure 2-9 is a simplified diagram of the LIDAR emitter/detector.

The client is responsible for defining the characteristics of the LIDAR system and passing the information to PANGU via the **SetLidarParameters** message (see Section 7.3). This includes the field of view of the scan, the number of horizontal and vertical scan positions, the number of detector pixels (for super-sampling), the time taken to scan one pixel, the time taken to scan a horizontal line and the angular size of the detector pixels. The client can also define what type of results they wish to receive for each scan: range/slope information, azimuth/elevation scan angles, corner cube range/slope information and the time at which each pulse was emitted. The client is expected to use the range and slope information in their LIDAR model to reconstruct the LIDAR detector signal.

Since the time for a single scan is expected to be of the order of one second, PANGU must integrate equations of linear and angular motion to determine the craft position and attitude for each pixel of the result image. For each scan request the client must provide the linear position, velocity, acceleration and jerk along with the angular attitude, velocity, acceleration and jerk. These parameters are defined relative to the time at the middle of the scan.

3.4.1.1 Zig-zag scan

The scanning pattern without supersampling modelling a single detector pixel is shown in Figure 3-2. The pattern with supersampling modelling an array of detector pixels is shown in Figure 3-3.

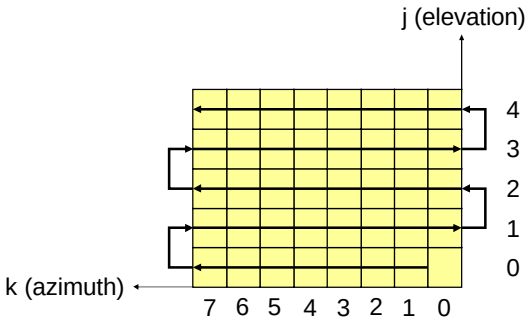

Figure 3-2: zig-zag scan without supersampling

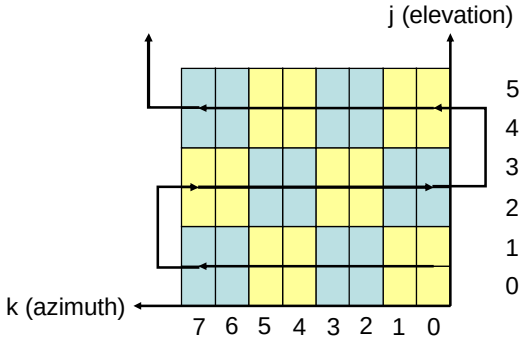

Figure 3-3: zig-zag scan with supersampling

The *j,k* parameters define the position of the pixel in the result image: they are used to define the time at which the pulse for the pixel was emitted and the direction in azimuth and elevation in the LIDAR frame. Note that when super sampling is used one pulse is used for all the pixels in the super-pixel. Thus each group of super-pixels corresponds to the same point in time but with slightly different azimuth and elevation angles.

Given the *j,k* values for a result pixel we can determine the *J,K* values for the pulse:

$$K = \left\lfloor \frac{k}{n} \right\rfloor \qquad dk = k \bmod n$$

$$J = \left\lfloor \frac{j}{m} \right\rfloor \qquad dj = j \bmod m$$

where (*dk, dj*) are the coordinates of sample (*i, j*) within the *n*×*m* sub-sample group for the pulse. The range of *K* is (0,*W*-1) and the range of *J* is (0,*H*-1) where *W* are the number of beams emitted per line and *H* the number of lines scanned per frame.

3.4.1.2 Time at which a pulse is emitted

Given the time t_c at the centre of the scan, the time T_{frame} taken to scan the whole frame, the time T_{row} taken to scan a line including the gap $T_{interrow}$ between lines, the time T_{line} to scan a line excluding the gap between lines, the time T_{pulse} between consecutive pulses on a line, we can compute the time t at which the pulse at position (*K, J*) is emitted:

$$t = t_c - \frac{T_{\text{frame}}}{2} + J \cdot T_{\text{row}} + (-1)^J \cdot K \cdot T_{\text{pulse}} + \left[\frac{(-1)^J + 1}{2}\right] T_{\text{line}}$$

The equations for T_{frame}, T_{row}, and T_{line} are:

$$T_{\text{frame}} = T_{\text{row}} \cdot H - T_{\text{interrow}}$$

$$T_{\text{row}} = T_{\text{pulse}} \cdot W + T_{\text{interrow}}$$

$$T_{\text{line}} = T_{\text{pulse}} \cdot W$$

3.4.1.3 Azimuth and elevation of the pulse

Given the azimuth offset Az_0 of the centre of the scan, the horizontal field of view FOV_x, and the angular width w_x of a sub-sample, the azimuth *Az* of the pulse at position (*K,J*) is:

$$Az = Az_0 + K \cdot \frac{FOV_{\text{x}}}{W} - \frac{FOV_{\text{x}}}{2} + dk \cdot w_{\text{x}} + \frac{w_{\text{x}}}{2}$$

$$w_{\text{x}} = \frac{FOV_{\text{x}}}{W \cdot n}$$

Given the elevation offset El_0 of the centre of the scan, the vertical field of view FOV_y, and the angular height w_y of a sub-sample, the elevation *El* of the pulse at position (*K,J*) is:

$$El = El_0 - J \cdot \frac{FOV_{\text{y}}}{H} + \frac{FOV_{\text{y}}}{2} - dj \cdot w_{\text{y}} - \frac{w_{\text{y}}}{2}$$

$$w_{\text{y}} = \frac{FOV_{\text{y}}}{H \cdot m}$$

3.4.1.4 Position and attitude of the LIDAR emitter/detector

Given the initial position vector $\mathbf{p_0}$, linear velocity $\mathbf{v_0}$, linear acceleration $\mathbf{a_0}$ and linear jerk $\mathbf{j}$ at time 0, the position $\boldsymbol{p}$ of the LIDAR instrument at time *t* is given by:

$$\mathbf{p} = \mathbf{p_0} + \mathbf{v}_0 t + \frac{\mathbf{a}_0 t^2}{2} + \frac{\mathbf{j} t^3}{6}$$

Given the initial attitude q_0, angular velocity $\mathbf{v_0}$ and angular acceleration $\mathbf{a}$ at time 0, the attitude *q* of the LIDAR instrument at time *t* is given by:

$$q = q_0 \cdot \left[\cos\frac{|\mathbf{r}|}{2} + \left((\hat{\mathbf{r}} \cdot \hat{\mathbf{i}})\sin\frac{|\mathbf{r}|}{2}\right)\hat{\mathbf{i}} + \left((\hat{\mathbf{r}} \cdot \hat{\mathbf{j}})\sin\frac{|\mathbf{r}|}{2}\right)\hat{\mathbf{j}} + \left((\hat{\mathbf{r}} \cdot \hat{\mathbf{k}})\sin\frac{|\mathbf{r}|}{2}\right)\hat{\mathbf{k}}\right]$$

where:

$$\mathbf{r} = \mathbf{v}_0 t + \frac{\mathbf{a}_0 t^2}{2}$$

and where $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$ and $\hat{\mathbf{k}}$ are the unit vectors in 3D space. The vector **r** is the axis of rotation.

This information allows PANGU to determine the direction in which the pulse is emitted and from what position.

3.4.1.5 Landing

During a landing simulation the LIDAR is used to scan the surface "without gaps". This means that the angular separation between pulses is tuned to match the angular separation of the pixels in the result image.

3.4.1.6 Rendezvous

During a rendezvous simulation when the system is attemping to locate the target the LIDAR can be used to scan the field of view "with gaps". In this situation the angular separation between pulses is larger than the angular separation of pixels in the result image.

3.4.1.7 Corner Cubes

During normal scanning PANGU fires infinitely thin rays from the LIDAR emitter, reflects them off the target and records the resulting range (echo delay) and slope information (signal strength) received by each detector pixel. This is sufficient for extended objects such as planet surfaces where an infinitely thin ray will always intersect the model at some point (unless the ray isn't directed at the surface). However, a small target such as a sample-return canister might be smaller than a detector pixel. To solve this problem and to support corner cube reflectors (which always return a strong signal irrespective of the angle of incidence of the emitted signal) PANGU models corner cube reflectors differently. The LIDAR pulse is modelled as a cone and the corner cube nearest to the LIDAR emitter that lies within the cone will be used to fill a detector pixel.

3.4.2 Memory management (introduced with PANGU 2.50)

The **viewer** and **mkshadows** have been extended with a memory management system which is designed to reduce the amount of RAM used to store the PANGU model being rendered or shadow mapped. In addition to optimisations of the runtime representation of objects to reduce wastage, mesh objects have been extended to support lazy loading and caching. These benefits can be seen by comparing the performance of PANGU 2.40 with PANGU 2.50 when loading and rendering large models.

Lazy loading allows a mesh object to be partially loaded into memory leaving the positions, normals and polygon mesh information omitted. When the client application (*e.g.* the **viewer**) needs to access the data that isn't in memory it can load it from the .pan file. This means that objects which are never displayed are never loaded into RAM saving time and memory.

Caching support allows a mesh object to write its position, normals and polygon data into a set of fixed-sized memory blocks and to read them back again.

When the memory management system determines that the RAM usage of the application has reached the ceiling set by the user (see the `viewer.cache_ceiling` INI option or the `-cache_ceiling` command line argument) it will ask every cachable (mesh) object which has not been rendered for several frames to format itself into a set of fixed-sized memory blocks. These blocks are written to a disk cache file and the mesh object then releases the memory that was associated with the cached data. If there is no room in the cache for an object then the memory manager will evict the oldest object. When the client application needs to access a cached mesh object again it ask the memory manager to reload it. If the object has been evicted from the cache then the object will be reloaded from the .pan file.

The size of the cache file on disk is controlled by the `viewer.cache_size` INI option: the value is the size in bytes of the cache. The size of the cache blocks is tailored to the model being cached.

The amount of RAM which can be used by cachable objects is controlled by the `viewer.cache_ceiling` INI option. Objects which cannot be cached will always be stored in RAM. The memory manager will allow more than the ceiling to be used if there are no objects which are available for caching (*e.g.* they are all visible).

To prevent the memory manager spending excessive amounts of time writing unused objects to the cache a limit can be placed on the maximum number of objects to be written in one attempt (one image frame). This limit is controlled by the `viewer.cache_burst_limit` INI option.

The memory management algorithm works well with large layered models where small pieces of the model can be unloaded from RAM. Unfortunately the system doesn't work for large single layer models or for ROAM models. Large single-layer models are generated as a single unit so there are no small pieces which can be unloaded from RAM unless the entire surface is outside the field of view. The **surfacegen** tool would need to be updated to split large models into smaller pieces to solve this problem. ROAM models use much less memory than the previous mesh models so lack of memory management support is not a big problem. Some benefits of the memory management system could be achieved in future for multi-layer ROAM models but not for single layer ROAM models.

3.4.3 ROAM Terrain (introduced with PANGU 2.50)

The ROAM algorithm is a technique for generating "optimal" triangle meshes which can be rendered using OpenGL or other systems. The meshes are optimal with respect a chosen quality metric for the ROAM mesh tessellation.

Previously the **viewer** was either forced to render every visible triangle in a model or to chose a particular level of detail for a hierachical layer and then render every visible triangle in that layer. These techniques work well but there is a tendancy for high resolution areas of the model to be "over rendered" where multiple triangles are rendered for each pixel of the image. This causes visual artefacts such as flashing when the camera viewpoint changes slightly. The root cause of this problem is that the **viewer** is unable to control the resolution of the model at each image pixel.

The basic ROAM algorithm can be summarised as follows: the complete terrain is treated as a square which is split across one diagonal to produce two base triangles. For each frame of the image the base triangles are split into two new triangles. Each new triangle is then split into two new triangles until the desired level of quality is achieved. In PANGU the quality metric used is the visible error in triangle shape: how accurate is a triangle compared to the two triangles that it can be split into? Additionally PANGU tries to avoid splitting triangles below the size of a pixel.

It is not possible to apply the ROAM algorithm to existing `.pan` files since they are too general. Instead the surface generator tool **surfacegen** has been extended with the –R command line option. If this is specified before the –L option then the resulting .pan file will be suitable for rendering with the ROAM algorithm. These ROAM `.pan` files are often much smaller than the general mesh .pan file.

The viewer uses the `viewer.roam_limit` INI option value to control the quality of the ROAM meshes that are rendered. A value of 0.25 pixels is usually needed to obtain a view equivalent to the previous PANGU mesh rendering code. In addition the `viewer.roam_size_factor` INI option can be used to control the smallest triangle size that is rendered. A value of 0.0025 is sufficient to prevent most triangles from being smaller than one pixel. Smaller values will allow smaller triangles to be rendered while larger values will increase the size of the smallest triangle rendered. These values can be modified by pressing the F (quality limit) and E (triangle size factor) keys.

Note that boulders are always rendered using the original mesh algorithm.

3.4.3.1 Note on ROAM quality and triangle size limits

The ROAM algorithm works well with large (multi-pixel) triangles which are textured to provide surface detail. However, with PANGU the surface detail is provided by the individual model triangles which need to be rendered as small as possible but not too small. The ideal situation is where every triangle covers just one pixel of the result image: if the triangles cover multiple pixels then the resulting images will be less detailed; if they are much smaller than a pixel then the resulting images may be "noisy", particularly when sequences of images are generated.

Users may find that if the ROAM quality limit (controlled by the F key) is made too small (high quality) then the viewer may take a long time to render the scene and triangles smaller than one pixel will be rendered and the benefits of the ROAM algorithm will be list.

The solution is to use the second parameter: the triangle size limit. This parameter is used to place a limit on the size of the smallest triangle that can be rendered. The triangle size limit can override the quality setting if splitting the triangle to improve the quality would create triangles that are too small.

In general the triangle size limit can be set to a fixed value such as 0.0025 and then the ROAM quality limit can be set to a small value (e.g. less than 0.1). These settings will encourage the ROAM algorithm to render triangles that are approximately one pixel in size whenever possible at the cost of increased rendering time.

With large models users may wish to use a larger ROAM quality limit to render scenes quickly without much detail. When a scene of interest is found then the ROAM quality limit can be changed to increase the scene quality.

3.4.4 Memory Management or ROAM?

Users are encouraged to use the memory management facilities to view existing large multi-layer models and for using PANGU on machines with limited RAM. As a general guideline the PANGU RAM usage will be twice the size of the model being viewed. If the model is more than 1/3 the size of the available memory on the PC then the memory management facilities ought to allow the model to be viewed without causing the PC to become unusable.

For new models users are encouraged to generate ROAM models since these are smaller than the original mesh models and the rendering quality can be controlled dynamically. Furthermore the rendering quality links the view point to individual triangles: with mesh files and bi-resolution layers the rendering quality links the view point to complete layers and can produce poor results for low-angle views of a model.

3.4.5 RADAR Simulation (introduced with PANGU 2.60)

The PANGU network protocol has been updated with a **GetRadarResponse** command and **RadarResponse** reply as described in Section 7.3.2. This can be used to support the simulation of RADAR altimeters in client programs.

The RADAR simulator requires the client to specify the position, attitude quaternion and velocity of the RADAR emitter and the full angular width of the beam. This is characterised by the cross-section view shown in Figure 3-4. The velocity of the emitter is needed only if the client is interested in the radial speed of the target model relative to the radar emitter *e.g.* for Doppler effects. The client must specify the number of times the RADAR beam foot print on the model is to be sampled and how the signal response is integrated for the client.

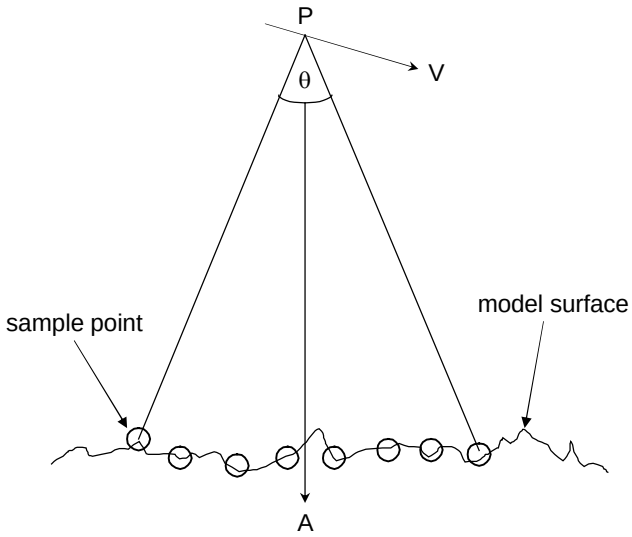

Figure 3-4: RADAR simulation beam/model cross-section

Figure 3-4 shows the basic properties of the RADAR simulation as a cross-section. The RADAR emitter is at position P moving with velocity V. The beam axis is along direction A: in this example pointing directly down at the nadir. The beam axis is specified by the attitude quaternion: the unit quaternion causes the beam axis to lie along the positive z-axis *i.e.* upwards. The angular width of the beam is θ degrees and some of the sample points on the model within the

beam foot print are highlighted with circles. Each sample has range (distance from P), radial speed (relative to P) and strength (of the reflected RADAR signal).

3.4.5.1 RADAR response and signal integration

The strength of all samples is integrated using a 2D range/speed histogram (such as the one shown in Figure 3-5) to obtain the signal response which is returned to the client. The range and radial speed of each sample point is used to select a histogram bin whose signal intensity is increased by the strength of the sample. Normally the strength of the sample is scaled by the cosine of the angle of incidence (surface slope) and the total signal intensity is normalised to 1.

3.4.5.2 Histogram attributes, size and limits

Figure 3-5 shows the general format of a RADAR histogram with four speed bins and ten range bins. The grey squares indicate histogram bins that contain sample results *i.e.* are non-zero. The smallest range of any sample is r_{min}, and the smallest speed of any sample is s_{min}. Likewise the largest range and speed of any sample is r_{max} and s_{max}. However, for "left-aligned" range histograms r_{min} is fixed at 0 and for "left-aligned" speed histograms s_{min} is fixed at 0.

The range and speed associated with the first histogram bin is r_{offset} and s_{offset} respectively. The physical width of the range histograms is r_{width} and the width of the speed histograms is s_{width}. Normally the r_{offset} and s_{offset} values are set to the values of r_{min} and s_{min} with $r_{width} = r_{max} - r_{min}$ and $s_{width} = s_{max} - s_{min}$ (after applying left alignment to r_{min} and s_{min} if necessary). If these widths are less than 0.001 then they are rounded up to 0.001 to avoid creating narrow histogram bins. For histograms whose range and/or speed have been rounded up to the nearest integral power of ten, r_{width} and/or s_{width} will be values such as 0.1, 1, 10, 100 *etc.*

For "centre-aligned" range and/or speed histograms, $r_{offset} = r_{centre} - r_{width}/2$ and/or $s_{offset} = s_{centre} - s_{width}/2$.

Unless fixed-sized histogram bins have been selected, the size of the histogram bins is obtained by dividing the width of the histograms by the number of bins (which must always be specified).

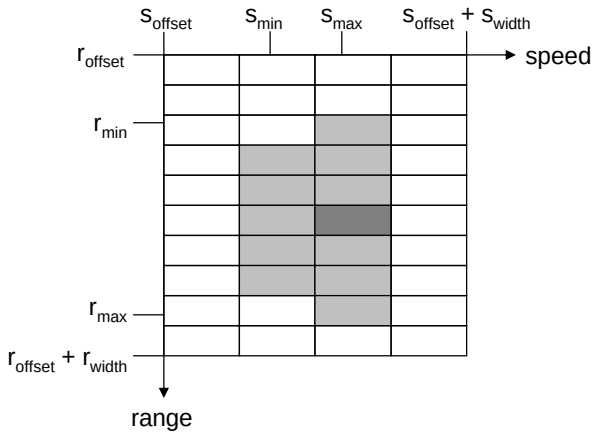

Figure 3-5: RADAR histogram format

3.4.5.3 Normal use

Normally the client will specify the number of samples of the RADAR foot print to be taken and the number of range and speed histogram bins to use for the integration of the RADAR response signal. PANGU will automatically adjust the histogram coverage and resolution to match the properties of the returned samples. However, to simplify the display of the RADAR response in simple graphical clients, clients may wish to request rounding of the histogram widths or restrict the alignment of the histograms.

3.4.5.4 Rounded histograms

The facility to round the width of the histograms up to the nearest power of ten is intended for use by simple graphical clients that wish to avoid rapid changes in the scale of the returned RADAR responses due to variations in the terrain

covered by the RADAR beam. In conjunction with left- or centre-alignment, the limits of the RADAR response are likely to vary slowly enough for human readers to interpret the response in real-time.

3.4.5.5 Aligned histograms

Using "left-aligned" histograms will normally decrease the resolution of the returned signal due to the corresponding increase in the size of the histogram bins. This form of alignment might be used for a simple RADAR display in which the target appears as a moving object relative to the RADAR emitter.

Using "centre-aligned" histograms will not decrease the resolution of the returned signal although a poor choice of centre value may result in all or part of the response being discarded due to the histogram limits. This form of alignment could be used in a simple RADAR display which highlights the variation in distances to different parts of the target, perhaps centred on the bore-sight range.

3.4.5.6 1D histograms

If the number of speed histogram bins is 1 then the 1D histogram of signal strength versus range will be obtained; likewise if the number of range histogram bins is 1 then the 1D histogram of signal strength versus speed will be obtained. This feature can be used by client programs which implement a simplified RADAR altimeter in which only range (or only speed) is considered. More advanced client programs can use the combined range/speed results along with the frequency of the transmitter to model the effects of Doppler shifts on the signal.

3.4.6 Fog/Dust Cloud Simulation (introduced with PANGU 2.65; extended by PANGU 2.70)

The PANGU viewer has been updated with support for global fog/dust cloud simulation. It also has a model of local dust clouds which are trapped below a horizontal plane of constant altitude, dust clouds or atmosphere within a sphere and local clouds within a general volume. Global fog has three types of extinction profile: linear, exponential and exponential-squared. Local clouds only have exponential extinction.For all types of fog/dust the viewer computes the fraction *f* by which the view is obscured by the fog/dust. A value of 0 means that there is no fog visible while a value of 1 means that only fog is visible.

3.4.6.1 Global fog/dust, linear profile

This type of fog/dust has the simplest extinction profile: the user specifies the range to the point where fog/dust extinction begins r_0 and the range to the point of total extinction r_1. The extinction factor at any point in the scene corresponds to a linear interpolation of the fog/dust extinction between the two extremes (total fog/dust and no fog/dust) based on the distance to the scene location. If the amount of extinction is defined by the fraction *f* where 0 is transparent and where 1 is opaque then:

$$f = \begin{cases} 0 & r < r_0 \\ \dfrac{r - r_0}{r_1 - r_0} & r_0 \le r \le r_1 \\ 1 & r > r_1 \end{cases}$$

where *r* is the range to the point in the scene being obscured.

This type of fog/dust is generally used to provide depth-cueing in which objects either become darker or closer in colour to the background the farther away they are.

3.4.6.2 Global fog/dust, exponential profile

Exponential fog/dust determines the extinction factor based only on the distance to the point in the scene and a fixed scale factor representing fog density. If the amount of extinction is defined by the fraction *f* where 0 is transparent and where 1 is opaque then:

$$f = 1 - \exp^{-r\rho}$$

where *r* is the range to the point in the scene being obscured and where ρ is the fog density.

This type of fog/dust is used to model the physical effects of always being inside a fog/dust cloud.

3.4.6.3 Global fog/dust, exponential-squared profile

This type of extinction profile is the same as exponential fog/dust except that the exponential term is squared:

$$f = 1 - \exp^{-(r\rho)^2}$$

3.4.6.4 Local-planar fog/dust

This type of fog/dust is characterised by the situation shown in Figure 3-6. The grey area represents a side view cross-section through the surface of the terrain which is partly covered by the fog/dust cloud. Two view points are shown: one inside the fog/dust looking in different directions and one outside the fog/dust also looking in different directions.

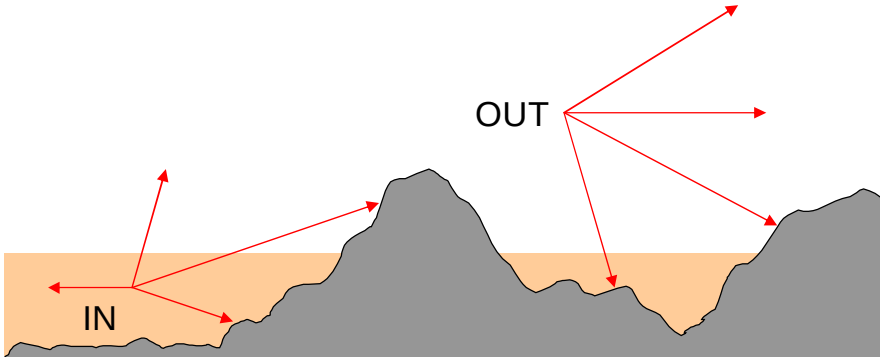

Figure 3-6: geometry of local-planar fog/dust with two view points

For all views in which fog is visible, the terrain or sky behind the fog/dust will be obscured by a factor *f* based on the exponential profile shown in Section 3.4.6.2.

3.4.6.5 Local-spherical fog/dust

This type of fog/dust cloud can be used to represent the atmosphere around a curved planet surface similar to that shown in Figure 3-7. Two view points are shown: one inside the fog/dust looking in different directions and one outside the fog/dust also looking in different directions.

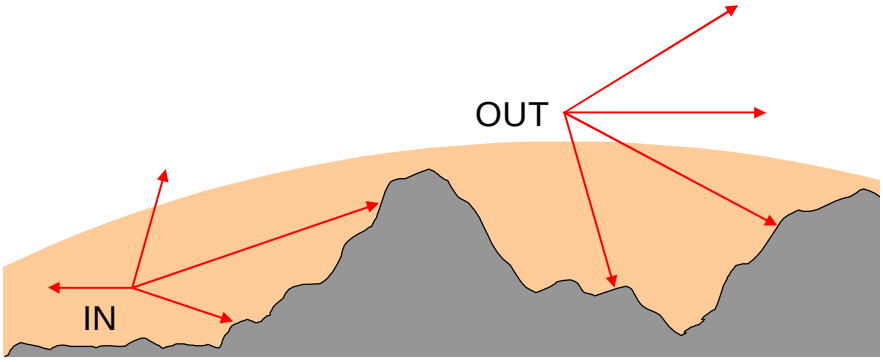

Figure 3-7: geometry of local-spherical fog/dust with two view points

3.4.6.6 Local-general fog/dust

PANGU 2.70 extends the simple fog/dust models presented above with a general fog/dust model. In this situation the user provides a model (a `.pan` file) which contains one or more solid objects which define the limits of the dust cloud or clouds. The user must also specify the location of the cloud model relative to the LDS origin. The cloud model must be closed *i.e.* it must represent a solid object (like an asteroid model) not a flat surface such as those generated by PANGU from DEMs. Otherwise the optical depth cannot be calculated and the cloud will have no effect on the scene.

The **viewer** assumes that there is one entry point and one exit point on the cloud along a given view direction (*c.f.* paths A and B in Figure 3-8). The distance between these two points defines the optical depth of the cloud and is used to determine the obscuration factor *f* of the terrain within or behind the cloud.

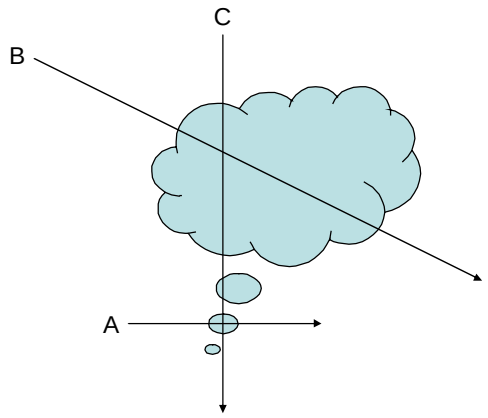

Figure 3-8: Different optical paths through a general cloud

If the view through a cloud model has multiple entry and exit points (either due to a grazing view along the uneven surface of the cloud or because multiple distinct clouds are present in the cloud model as in path C of Figure 3-8) then the computed optical depth will be larger than it ought to be. This is not expected to be a problem.

3.4.6.7 Sky dome

The PANGU **viewer** needs to know the distance at which extinction due to fog/dust ends relative to the camera. This distance is defined as the sky dome whose radius is set by the user. The amount of fog visible in the sky corresponds to the *f* factor resulting from the relevant fog extinction equation defined in previous sections with the *r* value set to the sky dome radius.

In PANGU 2.65 the two fog/dust types had separate sky distance parameters. However, due to the increased number of fog/dust cloud types, PANGU 2.70 uses a new INI option `viewer.fog.sky_distance` to define the sky distance for all fog/dust cloud types.

Note that the sky distance value must be larger than the distance to any part of the visible scene including any fog/dust clouds. For example, consider a scene with dust-inside-a-sphere which is used to enclose a spherical planet model with a dust sphere radius of 1×10^6 m. If the camera is placed at a distance of 3×10^6 m from the centre of the planet then the sky distance must be greater than 5×10^6 m (3×10^6 m + 1×10^6 m + 1×10^6 m).

3.4.7 Self-test facilities (introduced with PANGU 2. 70)

The PANGU **viewer** application has been extended with self-test facilies. Three tests are provided with release 2.70: they can be used to validate the rendering and back-projection system. The tests are accessed via the new command line argument `-test` which must be followed by the name of the test. The current self-test options are:

`-test lambert_00`	Lambertian reflection model test 0
`-test back_project_00`	back-projection test 0
`-test back_project_01`	back-projection test 1

It is recommended that the `-quit` option of the viewer is used to force the **viewer** to quit immediately after all the self tests have been performed. The `-list_tests` option can be used to see the list of available tests.

3.4.7.1 Self-test lambert_00

The `lambert_00` self-test is used to validate the Lambertian illumination equation $C_p = C_s\cos(\theta) + C_a$ used by the **viewer** where C_p is the colour of the image pixel, C_s the colour of the surface multiplied by the colour of the Sun, C_a is the ambient light colour and θ is the angle between the surface normal and the direction of the light source. Normally

C_a is black. The PANGU **viewer** C_p will be further modified by the floating-point/integer conversion scheme applied by OpenGL and the GPU.

The user must select appropriate ambient, Sun and surface diffuse colours via the command line or via a PANGU INI file such as `pangu.ini`. When the test is executed a flat planar surface is rendered at Sun elevation angles in the range [-90°, 90°] in steps of 1°. The colour of the pixel in the centre of the field of view is then captured and written to a text file `TEST_LAMBERT_00.txt`. The first column records the elevation angle, the second records the integer value of the red channel, the third records the green channel and the last records the blue channel. If the ambient, Sun and surface colours are all shades of grey then the last three columns ought to be identical. The pixel values lie in the range [0,255] with 255 corresponding to PANGU colour level 1.0. A plot of the elevation angle against one of the pixel colour channel values ought to yield a smooth cosine curve as shown in Figure 3-9. If the C_p value exceeds 1.0 then the top of the curve will be flattened at the 255 level indicating that the image is over-exposed. The minimum values at ±90° elevation will show the level of ambient light level as shown by the lower curve in Figure 3-9.

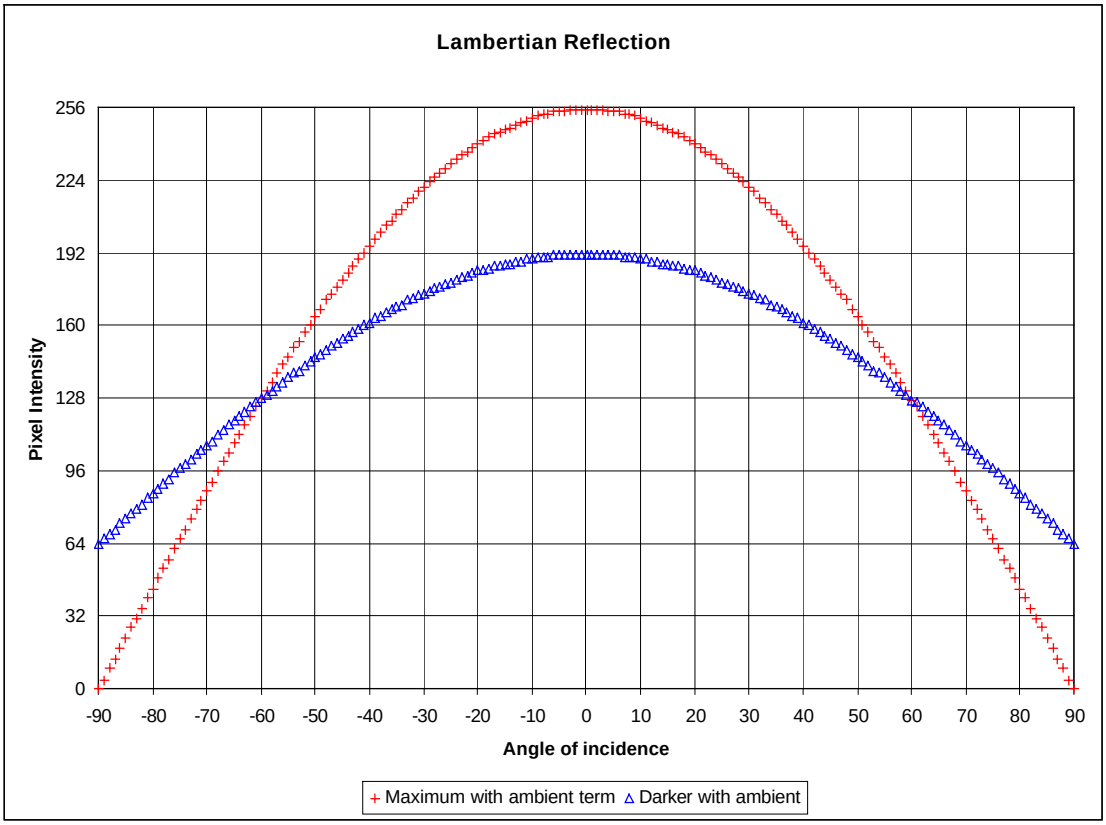

Figure 3-9: Example plots of Lambertian self-test results

Careful comparison between the test output and the expected float-point values of C_p may enable the rounding mode of the graphics system to be determined. Ideally round-to-nearest is used. Occasionally rounding errors due to lower-precision arithmetic may skew the results slightly.

3.4.7.2 Self-test back_project_00

The `back_project_00` self-test is used to validate the back-projection algorithm used in the **viewer** when responding to pixel-lookup requests (remotely via TCP/IP or locally via the F1 and F2 keys). Given the (x,y) coordinates of a pixel in the field of view and a square plane centred on the origin at a known distance p_z it is possible to compute the camera position $\mathbf{P}_{xy}$ at which the centre of the square appears under the specified pixel.

The test iterates over pixels of the middle row of the image $(x,0.5)$ with the camera positioned at the required $\mathbf{P}_{xy}$ comparing the horizontal coordinates obtained from back-projection with the expected (0,0,0) coordinate from the shifted camera. The difference between the two values is the back-projection error which is expected to be 0 for all pixels. If the model was not found under the specified pixel then the camera is moved around to see if the model can

be found. If it is then the back-projection error can be computed. The square plane is placed at a distance of p_z=1000 m from the camera.

The back-projection errors are written to the file `TEST_BACK_PROJECT_00.txt` in the current directory. The first column defines the general screen coordinate x, the second column defines the corresponding pixel coordinate within the image and the third column defines the x-coordinate of the camera required to position the centre of the plane model under the chosen pixel coordinates. The fourth column defines the expected camera position, the fifth column defines the absolute horizontal error in metres, the sixth column defines the actual range to the centre of the model in metres, the seventh column defines the expected range and the last column defines the absolute range error in metres.

The horizontal error and range error values are expected to be small (*e.g.* absolute value less than 0.0001).

3.4.7.3 Self-test back_project_01

The `back_project_00` self-test is used to validate the back-projection algorithm used in the **viewer** but it doesn't check that the rendered scene appears in the correct position in the images themselves. Test `back_project_01` is used to complete the loop.

The test iterates over the entire field of view at sub-pixel resolution. For each test location (*x*,*y*) the camera position $\mathbf{P}_{xy}$ is computed which is expected to place the top-left corner of the square at the sub-pixel location (*x*,*y*). The scene is then rendered, retrieved from the frame buffer and then examined by the self-test code to identify the integer pixel coordinates of the top-left corner of the square. The difference between (*x*,*y*) and these coordinates is a measure of the rendering accuracy. The test divides the field of view into a grid that is 3.1 times larger than the number of pixels in the **viewer** window. The factor of three means that at least nine samples are taken for each image pixel and the extra 0.1 term means that the sampling positions within the pixel vary slightly from pixel to pixel.

The test results are written to the file `TEST_BACK_PROJECT_01.txt` in the current directory: the first two columns hold the general pixel coordinates where the top-left corner of the square is expected to appear. The next two columns hold the integer coordinates of the pixel where the top-left corner of the square is detected. The last two columns hold the error between the expected and measured positions. The absolute errors must be less than one pixel.

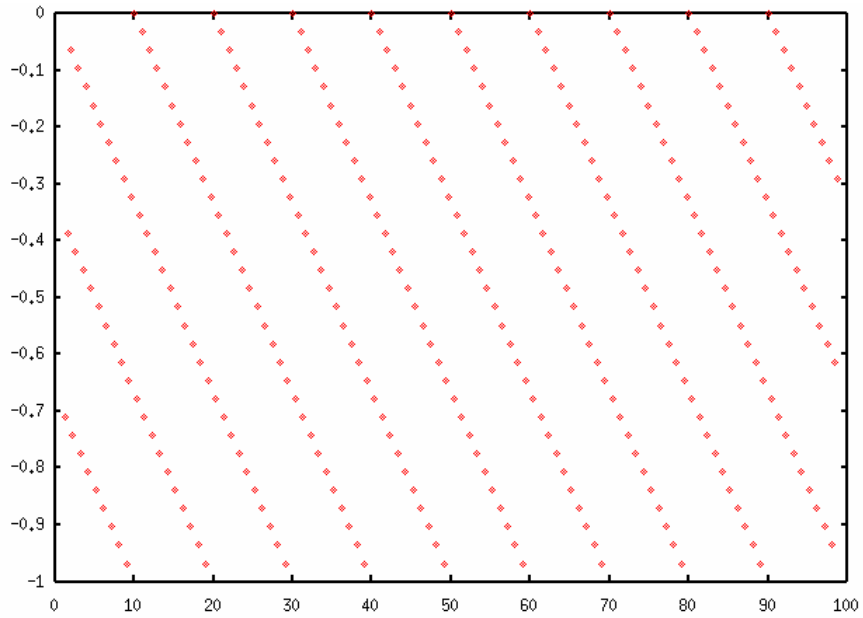

Figure 3-10: sub-pixel rendering position error (x-axis)

An example of the error in horizontal position is given in Figure 3-10.

4. SURFACE MODEL GENERATION COMMAND REFERENCE

This section details all surface generation commands. These are split between the two applications, **surfacegen** and **featurelistgen**. Each command is executed by calling the application using the required option and parameter text file. Example parameter files for all commands are given in the `testmodel1` directory. To automatically generate parameter files call the application with the –d option. For example:

```
../../bin/surfacegen –d Default
```

This command will generate all parameter files used by **surfacegen** and prefix the filenames with "Default".

4.1 SURFACEGEN

This program is used to create a surface model which may be rendered by the viewer. Fractal surfaces may be created and stored as DEMs, features may be added to DEMs and surfaces may be converted into polygon models. As with the other tools, this program may be invoked with the -d option to create default parameter files. Options are case insensitive:

`surfacegen` `[-option [`*`file`*`]]`

Options available are:

- -a, -A [A]dd craters to a surface: used to add craters to DEM in a single resolution model. Requires an add craters parameter file (Section 5.2.1).
- -c, -C [C]onvert a surface to a polygon model. This option should only be used if it is required to convert a DEM into a PANGU polygon file format: it is recommended that option [-L] is used instead. Requires a surface layers parameter file (Section 5.1).
- -d, -D Create [D]efault **surfacegen** parameter files. The *file* parameter is used as a prefix for each of the parameter files created and must be specified.
- -e, -E: Combine D[E]Ms: linearly adds two DEMs together to produce a new DEM. Requires an "Add DEM parameters" file (Section 5.2.5).
- -f, -F Create a [F]ractal surface: creates a single fractal DEM which can be used in a single resolution model or as the base for a hierarchical model. Requires a new surface parameters file (Section 5.2.2).
- -h, -H Display help text and exits **surfacegen**.
- -l, -L Add [L]ayers to a base surface: this is the recommended way to create hierarchical surface. Requires a surface layers parameter file (Section 5.1).
- -m, -M Create an albedo [M]ap. Not implemented.
- -p, -P Load [P]lanet properties: used to define radius of curvature.
- -r, -R Generate a ROAM model instead of a normal mesh model.
- -s, -S Generate an asteroid model using *file* to provide asteroid generation parameters.
- -u, -U Specify dune parameters filename to be used in model expansion for adding dunes.
- -w, -W Generate a whole-planet model using *file* to specify the model name, size and texture.
- -y, -Y Write DEM coordinates to *file*. If specified, this argument *must* be given before -L or -C options.
- -z, -Z Write .pan file lazily: this is much slower than the usual way of write .pan files but tends to use much less memory. This is useful when generating huge models.

4.2 FEATURELISTGEN

This program is used to generate lists of craters and boulders.

```
featurelistgen [-option file]
```

Options available are:

- -a, -A Cre[a]te a boulder list from layered crater lists: this is the recommended method to generate a boulder list for a hierarchical surface. Requires a surface layers parameter file (Section 5.1).
- -b, -B Create a [B]oulder list for a single resolution model: requires a "new boulders" file (Section 5.2.3).
- -c, -C Create a [C]rater list for a single resolution model: requires a "new craters" file (Section 5.2.4).
- -d, -D Create [D]efault **featurelistgen** parameter files. The *file* parameter is used as a prefix for each of the parameter files created and must be specified.
- -h, -H Display help text and exit **featurelistgen**
- -i, -I Same as -h/-H
- -l, -L Generate a [L]ayered crater list: this is the recommend option to generate crater lists for a hierarchical surface. Requires a surface layers parameter file (Section 5.1).
- -p, -P [P]rocess a crater list into sub crater lists. Requires a surface layers parameter file (Section 5.1). This option can be used to split a single resolution crater list into separate crater lists for a hierarchical surface. This method is only recommended if a pre-defined crater list exists. Option -L is normally used to generate crater lists for a hierarchical surface model.
- -s, -S Used to generate crater lists for asteroid models

5. SURFACE MODEL PARAMETER FILE REFERENCE

This section contains a description of each parameter file. The following applies to all PANGU parameter files.

- The first line in each parameter file defines its purpose. This line is used by PANGU to check the authenticity of the expected file. If the definition line is not there or incorrect then the operation will be aborted.
- The parameter files are self-documented as each parameter is defined inside the file.
- Lines starting with "//" are defined as comment lines and are ignored.
- Comment lines may be added as desired. Blank lines are not permitted.
- The files may be edited as desired but the order of the parameter definitions must remain unchanged as incorrectly ordering of the parameter definitions will cause PANGU to fail.

Distinct types of parameter files are:

- Layer parameters file: this is used by both **surfacegen** and **featurelistgen** when creating multi-layer models.
- Fixed parameters files: these contain a set number of parameters defined in order.
- Asteroid parameters file: this is used by **surfacegen** to generate asteroid models
- Whole planet parameters file: this is used by **surfacegen** to generate whole planet models
- Feature Lists: these contain parameter definitions and one or more feature (craters or boulders) definitions, one per line.
- Graph definitions: these contain a set number of parameters defined in order and one or more graph point definitions, one per line

A description of each parameter file will now be given. The file identifier is used as the file title and the name of the demonstration text file in the PANGU `testmodel1` directory is given in brackets.

5.1 LAYER PARAMETER FILE (LAYERS.TXT)

Multi-layer (multi-resolution) models are constructed using the information stored in a layers parameter file. An example of this is the `layers.txt` file in the `testmodel1` directory. A description of this file and instructions on how to use it are found in Section 3.2.

5.2 FIXED PARAMETER FILES

5.2.1 Add Craters Parameters (AddCraters.txt)

Defines the files required to add craters to a surface model. This includes input and output surface model files, a list of craters, the crater model properties and crater decay definition.

5.2.2 Surface Parameters (NewSurface.txt)

Defines parameters required to create a new surface DEM. This includes DEM size ($2^{magnitude}+1$), output file name and type, fractal parameters and random number seeding. Output file type 2 (Bitmap) is not yet supported

The last two parameters define the creation of a hierarchical surface. This feature is obsolete and not recommended since it does not allow the addition of craters to the hierarchical surface. Use the surface expansion method (as detailed in the tutorial) instead.

5.2.3 Boulder List Generation Parameters (NewBoulders.txt)

Defines a list of boulders which can be combined with a surface model and visualised using the viewer. The offset and area parameters define the surface area into which the boulders in the list are to be added. Each boulder is defined by four parameters; x position, y position, size and depth of burial. There should be at least one boulder definition in the file but there is no upper limit.

5.2.4 Crater List Generation Parameters (NewCraters.txt)

Defines parameter to randomly generate a single crater list. The random generator can be seeded, the area and offset into which craters can be positioned is defined and the number of craters to generate is specified. The output crater list

filename, diameter and age distributions and crater decay files are defined. The same crater decay file should be used to generate random crater lists as is used to add the craters onto a surface (Add Crater Parameters).

5.2.5 Add DEMs (AddDEMs.txt)

Defines the input and output files two DEMS. The output file is a new DEM with is the linear addition of the two input DEMs.

5.2.6 Crater Properties (CraterModel.txt)

This file is used to describe the properties of craters: their bowl, rim and ejecta profiles and the types of filters to use to remove or reduce artefacts introduced by fractal surface generation. These parameters have been optimised to produce realistic crater models. Altering parameters in this is file could significantly alter the crater models and so should be done with caution. ***Note that versions of this file distributed with PANGU 1.02 and earlier cannot be used with PANGU 1.50 and later.*** Some additional settings must be added to the end of the file to prevent invalid and degenerate models being generated. See section 5.2.6.22 for details.

5.2.6.1 Use cubic for bowl

This Boolean determines whether a cubic or a quadratic is used for the interior rim crater profile. The cubic creates a slightly more filled in crater profile shape then the quadratic which creates a slightly deeper bowl shape. The difference between the two model types is minor.

5.2.6.2 Crater Model Type

Two methods were created to minimise radial artefacts. The ejecta fix is best suited to fresh craters and the bowl fix is best suited to eroded craters. The mixed type is recommended as it utilises both model types depending on the erosional state of the crater.

5.2.6.3 Erosional state split for mixed type

Only used if a mixed model type is specified. This value determines the erosional state at which each model type is used. For example, if this parameter is set to 0.6 then any crater with an erosional state less than 0.6 will use the bowl fix model designed for eroded craters and any crater with an erosional state greater than 0.6 will use the ejecta fix model designed for fresh craters.

5.2.6.4 Use pre-Gaussian smoothing

Determines whether the rim area is smoothed prior to adding the crater. This reduces radial artefacts but can cause blurring if the crater overlay fractal parameters are not rough enough.

5.2.6.5 Use post-Gaussian smoothing for internal artefacts

Determines whether the interior rim is smoothed just before adding the fractal overlay

5.2.6.6 Use post-Gaussian smoothing for external artefacts

Determines whether the exterior rim is smoothed just before adding the fractal overlay.

5.2.6.7 Use angular fractal overlay

If set to true then the fractal overlay will be increased angularly where radial artefacts are likely to occur.

5.2.6.8 Number of Angles

Sets the number of angles used for all the angularly indexed arrays. This is used for rim smoothing and radius variation to create irregular craters. The number of angles sampled is the crater diameter multiplied by this parameter. A suitable value is p (3.14159) which ensures roughly one pixel difference between samples on the crater rim.

5.2.6.9 Rim variance

Rim variance min and max determine the range of irregularity of each crater. A random value is generated between rim variance max and min for each crater. This normalised value is used to determine the range of radius variation.

5.2.6.10 Rim variance filter width

A filtered one-dimensional random fractal is used to generate radius variation values. The greater the filter width value then the smoother the variations.

5.2.6.11 Rim smoothing filter width

Determines the width of the rim-smoothing filter.

5.2.6.12 Number of Gauss Filters

The number of different sized Gauss filters used.

5.2.6.13 Gauss filter width

Width of maximum Gaussian filter

5.2.6.14 Number of Pre-Gauss Filter runs

The number of times the Gaussian filter is applied for pre-smoothing the crater rim area.

5.2.6.15 Number of Post-Gauss External Filter runs

The number of times the Gaussian filter is applied for pre-smoothing the crater rim area.

5.2.6.16 Number of Post-Gauss Internal Filter runs

The number of times the Gaussian filter is applied for pre-smoothing the crater rim area.

5.2.6.17 Gauss filter range

The normalised radial range in which a Gaussian filter is applied to smooth the rim area is defined by the start and end values. The crater centre is –1, crater rim is 0, crater ejecta edge is 1.

5.2.6.18 Gauss bands

The Gauss bands determine the crater diameters for which the different sized Gaussian filters are to be applied to.

5.2.6.19 Minimum bowl overlay

The rest of the parameters define the fractal overlay. The minimum bowl overlay is the normalised minimum fractal overlay factor. This is required as the fractal overlay is reduced with crater decay and so a minimum is necessary to result in a realistic fractal surface of old craters.

5.2.6.20 Angular boost multiplier

The angular boost parameter is used to control the increase in the height factor used to add an angularity-weighted fractal to hide radial artefacts. The idea is to increase the height factor of the fractal overlay at areas of the rim where there is likely to be radial artefacts. These areas are determined by the difference between the ejecta and bowl at the rim. The threshold parameter is used to define the maximum angularly increased height factor. The rim difference threshold is used to determine what areas of the rim require the fractal overlay increased. The rim difference filter width determines the width of the filter used to smooth the rim difference values. The greater the filter with value the more general the area in which the height factor is increased.

5.2.6.21 Radial fractal bands

The next group of parameters from "Floor radius" to "Ejecta Factor" define the radial bands in which different fractal height factors are used in creating the fractal overlay. Regions of the crater are defined such as the floor, wall, rim and ejecta. A factor is defined for each band which is applied as a factor to the original height factor. This allows control of the roughness of the crater.

5.2.6.22 Crater profile equations

These are four new parameters introduced with PANGU 1.50. The maximum crater depth and the maximum crater rim height are defined by the equation kD^c. The "Crater depth constant" defines the value of "k" for the maximum crater depth equation while "Crater power constant" defines the value of "c". Likewise "Crater rim height constant" and "Crater rim power Constant" define the values of "k" and "c" for the maximum crater rim height equation. Users are advised to check their crater model parameter files before use to ensure that these parameters are defined: earlier versions of these files omit these parameters and will cause invalid and degenerate models to be generated.

5.2.6.23 Crater profile scale

This is a new parameter introduced with PANGU 2.00 and defines the scale of the units used in the crater profile equations of Section 5.2.6.22. A scale factor of 1 indicates that the units of the equations are measured in metres (NPAL profile equations); a scale factor of 1000 indicates that the units are in kilometres (Melosh profile equations).

5.2.6.24 Apply flat bottom fill ins

This is a new parameter introduced with PANGU 2.40 and is designed to simulate the effects of crater fill in found in some Martian craters. If this is set to true (1), then craters will be filled-in to give a flat-bottomed shape. If this is set to false (0), then the standard, lunar bowl-shape crater model will apply.

5.2.6.25 Flat age factor

This is a new parameter introduced with PANGU 2.40 and is designed to control the extent of crater fill and only applies if flat bottomed craters are enabled. This parameter controls the amount of crater fill in. The normalised amount of the crater fill-in is determined from the crater depth decay and the crater flat age factor. For example, a 100 m crater has a normalised age of 0.8 (where 1.0 is fresh and 0 is totally decayed). The translates into a depth decay of 0.8 meaning that the crater depth is 0.8 times maximum for a fresh 100 m crater. The depth of crater fill in will then be

0.8 * flatAgeFactor * maxCraterDepth

Flat age factor should always be greater than or equal to zero. Values of less than 1.0 will result in greater deposition and values of greater than 1.0 in less deposition.

5.2.6.26 Flat bottom curved width

Not currently used. The intention is to control the width of the curve at the border of the flat-bottom and the crater wall.

5.2.6.27 Flat age start

Not currently used. In future this may be the normalised crater age at which the crater begins to fill in. The fill-in rate is instead linked to the crater decay parameters.

5.2.7 Planet Properties (PlanetProperties.txt)

This file indicates whether the surface model will be flat or curved and defines the radius of curvature of the body that the PANGU surface is intended to simulate. If the "Curved surface" parameter is 0 then a flat surface will be generated. If this parameter is 1 then a curved surface will be generated assuming a spherical body with radius of curvature defined by the "Radius of curvature" parameter.

5.3 ASTEROID PARAMETER FILE

5.3.1 Output file name

This defines the name of the output `.pan` file which can be displayed using the **viewer**.

5.3.2 Asteroid Magnitude (N)

This defines the resolution of the asteroid model. The number of latitudes defined is $2^N + 1$. The number of longitude points on the equator is $6N$ and the total number of vertex points defined is $6N^2 + 2$.

5.3.3 Initial radius

This defines the radius of the initial spherical model.

5.3.4 Number of fractal iterations

This defines the number of Poisson Faults in the first stage.

5.3.5 Max fault size

This defines the maximum size of the first stage Poisson Faults.

5.3.6 Number of fractal iterations (2nd phase)

This defines the number of Poisson Faults in the second stage.

5.3.7 Max fault size (2nd phase)

This defines the maximum size of the second stage Poisson Faults.

5.3.8 Number of smoothing passes

This defines the number of times the smoothing algorithm is applied between the two Poisson Faulting stages.

5.3.9 Smoothing radius (in metres)

This defines the radius of the window used in the smoothing algorithm. The smoothed height value is the average of all height values within this radius. Larger values will result in a greater smoothing effect but will take longer to execute.

5.3.10 Fixed stretch on x-axis

This defines the stretch of the model along the x-axis as a factor of radius. A value of 1.0 means no distortion. A value of 2.0 would stretch the model to twice the radius along the x-axis and a value of 0.5 would compress the model to 0.5 times the radius.

5.3.11 Fixed stretch on y-axis

This defines the stretch of the model along the y-axis as a factor of radius. A value of 1.0 means no distortion. A value of 2.0 would stretch the model to twice the radius along the y-axis and a value of 0.5 would compress the model to 0.5 times the radius.

5.3.12 Fixed stretch on z-axis

This defines the stretch of the model along the z-axis as a factor of radius. A value of 1.0 means no distortion. A value of 2.0 would stretch the model to twice the radius along the z-axis and a value of 0.5 would compress the model to 0.5 times the radius.

5.3.13 Random stretch on all three axes.

If this parameter is set to zero (false) then the above three stretching parameters are used. If this value is set to 1 (true) then random stretches between 0.5 and 2.0 are applied to the three axes.

5.3.14 Seed Random numbers

The random number generator can either be seeded automatically by the program or by setting "Seed Random Numbers?" to 1 and setting "Seed" to the chosen seed. This can used to generate the same asteroid each time the program is run. The random seed must be between 0 and 32767.

5.3.15 Add Craters

If this parameter is set to 1 then a crater list will be added. The crater model, crater list and crater decay parameter file names are also defined here. These are used in the same way as adding craters to a flat DEM.

5.3.16 Fractal Parameters

This parameter defines the fractal number and height factor to be used when creating the fractal crater overlays. This is to allow craters to blend in to asteroids of varying roughness.

5.3.17 Boulder parameters

Boulders have not yet been implemented in the asteroid model.

5.3.18 Load model

If this parameter is set to 1 (true) then an asteroid model is loaded, otherwise an asteroid model will be generated using Poisson Faulting. The filename of the asteroid model to load should be specified. This is the intermediate asteroid file which is produced after Poisson Faulting.

5.3.19 Save Model

If this parameter is set to 1 (true) then an intermediate asteroid model file will be saved after Poisson Faulting. This file does not contain the axis distortion or the impacted craters. Note that it makes no sense to have both the "Load model "and "Save model" set to true: this will either overwrite the same model or create a duplicate.

5.4 WHOLE-PLANET PARAMETER FILE

5.4.1 Output filename

Defines the file name to be created which contains the whole-planet model.

5.4.2 Number of latitudes

This defines the number of lines of latitude used to construct the sphere. To be precise it defines the number of solid rings or disks which are stacked on top of each other to make the sphere. If this number is too small (< 10) the model generated will look polygonal in the **viewer**. For most purposes a value of 60–100 is sufficient while high resolution models can be constructed with values of 200 or more.

5.4.3 Number of longitudes

This defines the number of lines of longitude used to construct the sphere. To be precise it defines the number sections each latitude ring or disk is divided into. If this number is too small (< 20) the model generated will look polygonal in the **viewer**. For most purposes a value of 100–200 is sufficient while high resolution models can be constructed with values of 400 or more. As a general rule, there ought to be twice as many lines of longitude as there are lines of latitude.

5.4.4 Planet radius

This defines the radius of the whole-planet object in metres. Note that selecting a realistic value for this parameter will normally cause the viewer to start with the camera placed inside the planet object. Also since the physical size of the planet may not be relevant it may be more appropriate to select a small value for the radius and treat the units as kilometres instead of metres.

5.4.5 Texture filename

Without texture a planet object will be rendered as a diffuse white sphere in the **viewer**. This parameter allows a texture to be stored in the PANGU object file with the whole-planet object. The texture, if valid, will be used to paint the sphere when rendered in the **viewer**. The example model distributed with this documentation includes a texture map for the planet Jupiter.

If **surfacegen** is unable to locate the specified texture file then the planet will be generated without texture. If the **viewer** is unable to recognise the texture file format or if the dimensions of the texture image are not an integral power of two (e.g. 512x512 or 1024x1024) then the texture will not be rendered in the **viewer**. Currently the **viewer** only supports textures in PPM or PGM format.

5.5 FEATURE LIST FILES

5.5.1 Crater List (CraterList.txt, layer_0_craterlist.txt, layer_1_craterlist.txt, etc)

Defines a list of craters which can be added to a surface model with a surface model and visualised using the viewer. This file can be generated manually or through defined crater distributions using **featurelistgen**. The offset and area parameters define the surface area into which the boulders in the list are to be added. The horizontal scale value scales all position and size values in this file. It is recommended that the horizontal scale value is left as 1.

Each crater is defined by four parameters; x position, y position, diameter and age. There should be at least one crater definition in the file but there is no upper limit.

5.5.2 Boulder List (BoulderList.txt)

Defines a list of boulders which can be added to a model polygon file and visualised using the viewer. The offset and area parameters define the surface area into which the boulders in the list are to be added. The horizontal scale value scales all position and size values in this file. It is recommended that the horizontal scale value is left as 1.

Each boulder is defined by four parameters; x position, y position, size and depth of burial. There should be at least one boulder definition in the file but there is no upper limit.

5.6 GRAPH DEFINITION FILES

5.6.1 Crater Diameter Distribution (CraterDiamDist.txt)

This file defines the distribution of crater diameters and begins with the limits on crater diameters and then the maximum relative probability. The rest of the file contains pairs of numbers on each line corresponding to the crater diameter and the relative probability of that diameter occurring.

5.6.2 Crater Age Distribution (CraterAgeDist.txt)

This file has the same format as CraterDiamDist.txt and defines the probability distribution of crater ages. Each pair corresponds to a crater age and the relative probability of a crater of that age occurring.

5.6.3 Crater Decay (CraterDecay.txt)

This file has a similar format as `CraterDiamDist.txt` but contains two distribution graphs, separated by a line containing a single exclamation mark "!". The first graph defines the normalised depth according to its age while the

second defines the normalised rim height by crater age. The depth graphs apply to a crater of diameter "Base Crater Diameter". These decay graphs are then extrapolated to apply to craters of all diameters.

5.6.4 Boulder Size Distribution (BoulderSizeDist.txt, BoulderCRSizeDist.txt)

This file has the same format as `CraterDiamDist.txt` and defines the distribution of the sizes of boulders which are randomly distributed across the landscape. In the example files `bsizedist.txt` is used to define a global distribution of boulder sizes while the bcraterdepthdist.txt is used to define a crater-related boulder size distribution.

5.6.5 Boulder Depth Distribution (BoulderDepthDist.txt, BoulderCRDepthDist.txt)

This file has the same format as CraterDiamDist.txt and defines the distribution of the depths at which boulders which are randomly distributed across the landscape are buried. A depth of zero means the boulder lies on the surface. In the example files `bdepthdist.txt` is used to define the global distribution of boulder depths while the `bcraterdepthdist.txt` file is used to define the crater-related boulder depth distribution. Note that in PANGU 2.00 the depth measurement is relative to the size of the boulder: a depth of 1.0 indicates complete burial so that the top of the boulder lies on the surface.

5.6.6 Boulder Radial Distance (BoulderRadialDist.txt)

This file has the same format as `CraterDiamDist.txt` and defines the distribution of the radial distances at which boulders can be found for boulders associated with craters. The crater rim is at radial distance 1.0 irrespective of the definitions in CraterProperties.txt.

5.6.7 Boulder Frequency Distribution (BoulderFreqDist.txt)

This file has the same format as `CraterDiamDist.txt` and defines the distribution of the number of boulders to generate for a particular crater. The actual number of boulders to generate is usually defined by bfrequency.txt.

5.6.8 Boulder Frequency (BoulderFreq.txt)

This file has a similar format as `CraterDiamDist.txt` and defines the distribution of the number of boulders to generate according to crater diameter.

5.6.9 Boulder Depth (BoulderMinDepth.txt)

This file has the same format as `BoulderFreq.txt` and defines the distribution of the minimum depth at which boulders may be buried according to crater diameter.

5.6.10 Max Boulder Size (BoulderMaxSize.txt)

This file has the same format as `BoulderFreq.txt` and defines the distribution of boulder sizes to crater diameter.

5.7 DUNE PARAMETER FILES

The dune parameters file defines the basic parameters required for dune generation. These can be specified in two ways: process based and form based. In both cases the first six parameters detailed below are the. Then, depending on the Form Based Parameters Flag, the relevant set of parameters is specified.

It is also necessary to provide a wind slope angles file. If form based parameterisation is selected it is also necessary to provide a dune length parameters file, a horn length parameters file and a horn angle parameters file.

5.7.1 Add Dunes Flag

This is set to 1 to enable dune addition to a surface, 0 to disable dune addition.

5.7.2 Maximum Slope Angle

This specifies the maximum angle of slope (degrees) of areas which are to be considered as potential sites for dunes. Increasing this figure will increase the area covered by the dunes.

5.7.3 Proportion to Dune

This specifies the proportion of the area of potential dune sites actually used for dune sites. It takes the range 0 to 1. This is a statistical measure and the actual area used will vary, with a Normal distribution, around this figure.

5.7.4 First Layer to Dune

This specifies the first layer within a PANGU hierarchical model to which to add dunes. In large models, with relatively low resolution layers, the dunes will not be visible in these layers. Therefore it will speed up program execution not to generate dunes for these layers.

5.7.5 Wind direction

Specifies the direction from which the wind will blow in the model; measured in degrees clockwise from North.

5.7.6 Attraction

This defines the distance in meters at which nearby dunes will snap together to form ridges

5.7.7 Dunefield Border Factor

A border around the edge of each dune field is left free of dunes. Its width is described relative to the linear dimensions of the initial barchans used to create the dune field. This factor controls the width. 0 = no width, 1 = full width.

5.7.8 Landing Site

It is possible to specify a landing site in the dune parameter file. This ensures that the circular area defined will not have any dunes added to it. If this feature is not desired simply set the landing site radius to zero;

5.7.9 landing site centre x coordinate

5.7.10 landing site centre y coordinate

These specify the x and y coordinates of the centre of the proposed landing site. There are specified in meters with (0,0) representing the centre of the model.

5.7.11 Landing site radius

This specifies the radius of the landing site (in meters)

5.7.12 Lee slope Angle

This defines the angle of the downwind slope of the dunes. This is governed by the angle of repose of the dune forming material rather than the dynamic sculpting of the wind.

5.7.13 Brink Angle

This defines the angle to which the dune slopes downwind of its crest before the lee slope commences.

5.7.14 Wind Slope Angles Filename

This specifies the file which contains the data describing the variation of average windward slope angle with the size of the dune

5.7.15 Random Seed Flag

It is possible to seed the random number generator with a set value. This ensures that for identical terrains and parameters the same dunefield will be generated.

5.7.16 Random Seed Value

If the random seed flag is set to 1 this parameter is used to seed the random number generator. Otherwise this value has no effect.

5.7.17 Form based Parameters Flag

Set to 0 to generate dunes based on a wind strength and sand supply. Set to 1 to specify the form of the dunes directly.

5.7.18 Sand supply

This defines the quantity of sand available for the generation of dunes. The units are not defined. See Also Wind Strength below.

5.7.19 Wind Strength

This defines the speed of the wind that forms the dunes. As with sand supply the units are not defined.

5.7.20 Density

This defines the relative density of coverage of initial barchans for seeding the dune field. A density of 1 indicates that the coverage of the initial barchans equals the area of the dune field.

5.7.21 Dune Lengths Distribution Filename

This specifies the file which contains data describing the distribution of dune sizes in the dune field: this is taken to be the measurement along the ground from the brink line to the windward slope toe.

5.7.22 Horn Angle Distribution Filename

This specifies the file which contains data describing the distribution of the horn angles of the dunes.

5.7.23 Horn Lengths Filename

This specifies the file which contains data describing the relationship between dune length and horn length for the dune field.

5.8 SUPPLEMENTARY DUNE PARAMETER FILES

5.8.1 Wind Slope Angles File

This file contains the data describing the relation between average wind slope angle and the length of the dune (measured from its brink to its windward extent parallel to the wind direction).

The file simply contains a list of graph points (*x y*)

Where *x* = windward slope length (meters) and *y* = windward slope angle (degrees)

Values between the defined points are interpolated.

5.8.2 Dune Lengths Distribution File

The distribution of the dune lengths (The length of the windward slope) can be described in one of two ways: either as a cumulative frequency, or as a pseudo-Gaussian distribution. Which is used is specified by the first parameter in the file, the distribution flag.

5.8.2.1 Distribution Flag

This specifies which style of distribution is defined in the file. A value of 0 specifies that a cumulative distribution graph of the dune lengths will follow. A value of 1 indicates that a pseudo Gaussian distribution is to be used.

5.8.2.2 Pseudo Gaussian Distribution

For a pseudo Gaussian distribution the minimum and maximum dune lengths are then specified

5.8.2.3 Cumulative Frequency Distribution

For a cumulative frequency distribution points on a graph (*x y*) describing the distribution are specified.

Where *x* = the probability of a dune of length less than *y* occurring,

And *y* = windward slope length in meters.

5.8.3 Horn Angle Distribution File

The distribution of the angles between the dune horns can be described in one of two ways: either as a cumulative frequency, or as a pseudo-Gaussian distribution. Which is used is specified by the first parameter in the file, the distribution flag.

5.8.3.1 Distribution Flag

This specifies which style of distribution is defined in the file. A value of 0 specifies that a cumulative distribution graph of the horn angles will follow. A value of 1 indicates that a pseudo Gaussian distribution is to be used.

5.8.3.2 Pseudo Gaussian Distribution

For a pseudo Gaussian distribution the minimum and maximum horn angles are then specified

5.8.3.3 Cumulative Frequency Distribution

For a cumulative frequency distribution points on a graph (*x y*) describing the distribution are specified.

Where *x* = the probability of a horn angle of less than *y* occurring,

And *y* = Angle between the dunes horns in degrees.

5.8.4 Horn Length File

This file defines the relationship between the windward slope length of the dune and the lengths of its horns.

This takes the form of a linear equation with random variation.

Horn length = **M** (windward slope length) + **C** +OR* **V**

5.8.4.1 Slope

This defines M in the above equation.

5.8.4.2 Intercept

This defines C in the above equation

5.8.4.3 Variation

This defines V in the above equation

5.8.4.4 Relative Variation Flag

If this flag is set to 0 the variation, a random number in the range +/-V, is added to the result.

If this flag is set to 1 the result is multiplied by a random number in the range +/-V.

6. PANGU VIEWER COMMAND REFERENCE

6.1 VIEWER

6.1.1 Advanced keyboard use

Due to the lack of graphical user interface elements the viewer makes heavy use of the keyboard. All of the settings accessible as keyboard controls can be defined in the `pangu.ini` file. However, the keyboard controls can be used to help identify suitable values for placing in `pangu.ini`.

When the viewer starts a list of keys is written to the error stream (see the **log_errors** setting in `pangu.ini`). These keys and the use are described below. Note that the rendering window must be selected for these keys to work. If the control panel is selected then they will have no effect or may change a control panel setting instead.

Use the SHIFT key with an alphabetic key to reverse its operation (if appropriate). For example pressing A will increase the ambient light level of a scene: holding SHIFT while pressing A will decrease the ambient light level.

ESCAPE	Quit the program.
ENTER	The ENTER key changes the texture mode between nothing, surface texture and surface detail.
TAB	Switch between camera attitude and position editing for keyboard control.
=	The current camera and viewer settings are written to the error stream (*e.g.* command window or the `PANGU.log` file). The results can be used to update the `pangu.ini` file.
?	Writes the list of key controls to the error stream.
#	Toggle flat/smooth shading. By default the planet surface is rendered using smooth shading while boulders are rendered using flat shading. This key will change from smooth shading to flat shading and back again. If both surfaces and boulders are smooth shaded then they will both become flat shaded and vice versa.
/	This is an experimental feature which accesses two false-colour modes. The first is a colour-coded elevation (contour) map of the surface while the second is a range (distance) map. Red areas are the lowest (nearest) then yellow, green and through the colours of the rainbow to blue. If the surface appears all in one colour even though there is significant variation in elevations (or distances), try pressing T repeatedly to change the current scale factor. If the surface colour changes cycles from red to blue several times, press SHIFT-T until colours are more evenly spread. See the **viewer.contour_map** and **viewer.range_map** INI options for more details.
< and >	These can be used to increase or decrease the size of the viewer window when in expert mode (see the **viewer.expert** setting).
A	Increase or decrease the level of ambient light.
B	Toggles the display of boulders: the viewer can display images faster without boulders.
SHIFT-B	Toggles the use of boulder texture.
C	This key resets the camera to the original settings when the viewer started.
SHIFT-C	Enable/disable rendering of the sample-return canister.
D	Toggle colour/depth mode.
SHIFT-D	Toggle depth-buffer optimisation.
E	Increase/decrease the ROAM triangle size factor.
F	Increase/decrease the ROAM quality measure.
G	Toggle experimental on-screen display facility.
H	This key enables the Hapke BDF approximation and can significantly slow rendering speed.
I	Re-load the .INI files.
J	Increase/decrease the field of view angle
K	Increase/decrease the fog density
L	Increase or decrease the brightness of the Sun. This can be useful if texture is applied and has darkened the surface too much. However, there is a limit to the amount of brightness which can be applied and the colours can saturate and still produce a dark image.
M	Enable/disable model-view target marker (a cube).

SHIFT-M	Enable/disable the user-defined landing site marker.
O	Enable/disable the surface coordinate origin marker.
P	The current camera and view settings are written to the error stream as a POV-Ray script.
Q	Quit the viewer.
R	Roll the camera about the view direction.
S	Save the current view as a PPM image. The first image will be called `scr_000.ppm`, the next `scr_001.ppm` etc. See the **save_format** setting in `pangu.ini` for details). PPM images can be viewed using the **imgview** tool or a suitable graphics program. Under Linux try xv, display, gimp or a PBMPlus utility; under Windows try an application such as PaintShop.
T	When in false-colour (contour) mode this key changes the scaling of the contours. Otherwise this key changes the scaling of the surface texture, detail and boulder texture by the same amount. It is not possible to change the surface texture, detail and boulder texture independently in the viewer. Instead, use this key to get the surface texture correct and then press the = key. The current texture scaling will be written to the error stream (e.g. `PANGU.log`). Then enable surface detail and use T and = again to identify that scaling.
U	When in model-view mode this raises or lowers the target vertically with respect to the surface. In craft-view mode it raises or lowers the camera vertically just like the Z key.
V	This key only works in expert mode (see the **viewer.expert** setting) and can be used to switch between a model-based view and craft-based view.
W	This key cycles between filled polygon mode, wire frame mode and point cloud mode.
Z	In model-view mode this changes the zoom (moves the camera closer to or further away from the target). In craft-view mode it raises the camera vertically with respect to the surface.
F1	Display the coordinates of the point on the model under the mouse.
F2	Make the point on the model under the mouse the target of the fixed camera.
F3	Enable/disable culling of objects outside the field of view (performance optimisation).
F4	Enable/disable depth rescaling (avoid limitations of the depth buffer).
F5	Change the way that the sky background is rendered (*e.g.* to enable stars).
F6	Increase the scaling of textured sky maps.
F7	Decrease the scaling of textured sky maps.
F8	Create or wipe a ray-traced depth image (warning: this make take many minutes to complete).
F9	Toggle the use of vertex arrays (if supported)
F10	Display the limits of the current field of view
F11	Cycle through fog modes (used to be experimental toggling of sphere-mapping of boulders)
F12	Evict all objects from the memory manager cache

The arrow keys were described in Section 3.3.3.2 and 3.3.4.2.

6.1.2 Complete INI option reference for the viewer

6.1.2.1 Model and shadow map options

model *fil*

viewer.model *fil*

This option instructs the **viewer** to load the model from the file *fil* unless the user specifies an alternative model on the command line.

shadowmap *fil*

viewer.shadowmap *fil*

Computing cast shadows for surfaces with more than a few thousand vertices is extremely time consuming and we are forced to use an offline shadow map generator to compute shadows for a particular model and specific Sun position and attitude. The **mkshadows** and **mergeshadows** tools are provided for this purpose.

The shadow map file *fil* must correspond to a shadow map generated from the model that the viewer will display. Otherwise the viewer application may terminate abnormally shortly after starting.

smooth_shadows true | false.

viewer.smooth_shadows true | false.

If true the viewer will attempt to anti-alias shadow edges.

This option should not be used with models generated by PANGU 1.50 or later.

viewer.ignore_hashcode true|false

Each shadow map contains a unique hash code number which identifies the model from which the map was created. If the hash code stored in a shadow map file does not match the corresponding code in the model then the viewer will refuse to load the shadow map. The user may force the viewer to load the shadow map by setting this parameter to true. However, doing so may cause the viewer to terminate abnormally.

lazy_load true | false.

viewer.lazy_load true | false.

If true the **viewer** will attempt to load the model lazily using the memory management system. This reduces the amount of memory used by the viewer but may increase model load times.

6.1.2.2 User interface properties

angle_step *f*

viewer.angle_step *f*

Changes in yaw, pitch, azimuth and elevation angles using the keyboard will be made in steps of *f* degrees.

full_screen true | false

viewer.full_screen true | false

If true, the viewer will occupy the whole screen rather than use a window.

log_errors *f*

All error and information messages will be written to the file *f* (e.g. `PANGU.log`). The special file name "-" will redirect messages to standard output while the name "+" will redirect them to standard error.

move_step *f*

viewer.move_step *f*

Changes in position using the keyboard will be made in steps of *f* units.

mouse_rotate_scale *f*

viewer.mouse_rotate_scale *f*

Changes in yaw, pitch, azimuth and elevation angles using the mouse will be made in steps of *f****angle_step** degrees for each pixel of mouse movement. This is allows finer movement control compared to the keyboard.

view_mode model | craft

This option tells the **viewer** to start in model-view or craft-view mode.

viewer.report_keys true | false

If this setting is true then the **viewer** will display a message describing the effect of each key when pressed. For example, pressing S will display the message "s: save image to file". This allows users of slow machines to know what is happening and can be used to educate new users. The message will disappear when the next view is rendered and so some messages may not be visible for very long.

window_size *width* *height*

When not in full-screen mode the rendering area will be *width* pixels by *height* pixels in size.

6.1.2.3 Surface properties

surface.detail *f* *s* *b*

Close-up views of polygonal meshes such as those used by the viewer are often unrealistic but applying a suitable texture map to the surface can alleviate this problem. This setting instructs the viewer to load a texture image from the file *f* with scaling *s*. If *b* is true then the texture would be enabled although this facility is currently disabled. See Section 6.1.1 for a description of the ENTER and T keys for more details.

surface.diffuse.colour *r* *g* *b*

The fraction of RGB intensities reflected by the surface under the Lambert model is *r*, *g* and *b*.

surface.hapke.colour *r* *g* *b*

The fraction of RGB intensities reflected by the backscatter component of the Hapke BDRF approximation of the surface is *r*, *g* and *b*.

surface.hapke.coefficient *f*

The backscatter response of the Hapke BDRF approximation falls off according to $\cos(\theta)^f$ where θ is the angle of the camera relative to the Sun direction. A value of 125 provides a narrow angle backscatter response.

surface.specular.coefficient *f*

This option would not normally be used: it controls the angular width of highlights on highly reflective surfaces based on a $\cos(\theta)^f$ function.

surface.specular.colour *r* *g* *b*

This option would not normally be used: it allows highly reflective surfaces to be modelled: the value of *r*, *g* and *b* define the colour of specular highlights on such surfaces.

surface.texture *f* *s* *b*

Smooth shading of polygonal meshes such as those used by the viewer can cause low-detail features such as small to medium-sized craters to appear blurred when viewed from certain distances. Applying a suitable texture map to the surface can alleviate this problem but only for a limited range of viewing distances. This setting instructs the viewer to load a texture image from the file f with scaling s. If b is true then the texture would be enabled although this facility is currently disabled. See Section 6.1.1 for a description of the ENTER and T keys for more details.

6.1.2.4 Boulder properties

boulders.detail *f* *s* *b*

boulders.diffuse.colour *r* *g* *b*

boulders.specular.colour *r* *g* *b*

boulders.specular.coefficient *f*

boulders.texture *f* *s* *b*

These parameters define the properties of boulders just as for surfaces: see Section 6.1.2.3 for details.

do_boulders true | false.

viewer.show_boulders true | false.

If true, boulders will be displayed. Use the B key to enable/disable boulders at runtime.

6.1.2.5 Reflection model options

ambient *r g b*

viewer.ambient *r g b*

This defines the red, green and blue (RGB) intensity of ambient light as *r*, *g* and *b*. Every part of the scene receives this amount of light even if it is totally shadowed. It is used to model general light scattering.

sun.colour *r g b*

Define the RGB intensity of the Sun as *r*, *g* and *b*. If any part of the surface directly faces the Sun then the colour in the rendered image under the Lambert model will be the product of **sun.colour** with **surface.diffuse.colour** plus the product of **ambient** with **surface.diffuse.colour** clamped to the range [0, 1].

sun.position *r t p*

This defines the Sun position in spherical polar coordinates: distance *r*, azimuth angle *t* and elevation angle *p*.

viewer.combine_method accum | blend

To generate the Hapke BDRF approximation the viewer renders each scene twice and adds the two images together. There are two ways of doing this: one using blending and the other using the accumulation buffer. If you have a fast CPU then setting combine to **blend** is generally faster but will produce poor results on displays with less than 24 bits of colour depth. If you have a slow CPU and fast graphics processor or low colour depth then **accum** will probably be better. In PANGU 2.00 this option is ignored: images are always blended.

viewer.hapke_coefficient *f*

The Hapke BDRF is implemented in the viewer by generating a view of the model under the Lambert reflection model and adding it to a view of the model under the approximation of a backscatter component. The ratio of the contribution of the backscatter image to the Lambert image is *f*: if *f* is 1 then there will be no Lambert component to the image; if *f* is 0 then there will be no backscatter component. The backscatter image is controlled by the **surface.hapke.colour** and **surface.hapke.coefficient** properties.

viewer.reflection_model lambert | hapke

This option defines the reflection model to use: the **lambert** model uses a standard Lambert reflection function for diffuse surfaces while the **hapke** model is an approximation of the Hapke BDRF.

6.1.2.6 Camera properties

fixed_camera *x y z d azi alt*

The model-view (fixed) camera will be positioned at a distance *d*, azimuth angle *azi* and elevation angle *alt* relative to the target at (*x*, *y*, *z*).

free_camera *x y z a p r*

The craft-view (free) camera will be positioned at a (*x*, *y*, *z*) with yaw angle *a*, pitch angle *p* and roll angle *r*.

viewer.aspect_ratio *f*

The horizontal angular size of a pixel divided by the vertical angular size of a pixel will always be *f*. The viewer will adjust the vertical field-of-view of the camera using the **viewer.field_of_view** setting to ensure that the pixel aspect ratio is maintained irrespective of the size and shape of the viewing area.

viewer.far_distance *f*

Objects further from the viewer than *f* units will not be shown or will be "sliced" in half unless depth rescaling is enabled. With depth rescaling, objects between distance *f*/2 and 8 will be rescaled and repositioned using a power law so that they lie in the range *f*/2 to *f* and so no object would be too far away to be drawn. In technical terms this option defines the position of the far clipping plane (the back of the depth buffer).

viewer.field_of_view *fov*

The horizontal field of view width will be *fov* degrees.

viewer.landing_site *file x y w z n*

This setting enables the user-defined landing-site marker. The marker will be represented by a square section of the model *w* units across lifted by *z* units above the true model surface centred on the coordinates (*x, y*). The marker will be created by sampling the model using a grid of *n* by *n* samples. The image in *file* will be used as a texture. The value of *z* should be a small positive number such as 0.1 while *n* should be large enough to ensure that an accurate copy of the underlying model can be made. If the mesh resolution is 1 unit then $n = w+1$ or $n = 2w+1$ can be used.

viewer.near_distance *f*

Objects closer to the viewer than *f* units will not be shown or will be "sliced" in half (see Figure 3-1). In technical terms this option defines the position of the near clipping plane (the front of the depth buffer). This value must be greater than 0 or strange effects will occur.

viewer.optimise_depth true|false

If this setting is enabled then the viewer will automatically adjust the near and far planes (within the limits set by **viewer.near_distance** and **viewer.far_distance**) to ensure that the depth buffer is used optimally. This may cause the scene to be rendered twice when the camera position changes significantly (once to determine the optimal view plane distance and a second time to generate the correct view) but does not impose a performance penalty on every image produced.

viewer.origin *file w z n*

This setting enables the surface coordinate origin marker. The marker will be represented by a square section of the model *w* units across lifted by *z* units above the true model surface. The marker will be created by sampling the model using a grid of *n* by *n* samples. The image in *file* will be used as a texture. The value of *z* should be a small positive number such as 0.1 while *n* should be large enough to ensure that an accurate copy of the underlying model can be made. If the mesh resolution is 1 unit then $n = w+1$ or $n = 2w+1$ can be used.

viewer.output_scale *f*

This setting defines the scale factor applied to all internal distance units before they are displayed. By default all internal units are measured in metres and so *f* defaults to 1.

viewer.output_units *s*

This setting defines the units placed after distances displayed by the viewer. The default for *s* is "m".

viewer.recenter true|false

Often the origin of the PANGU coordinate system lies above or below the surface of the model. If this option is set to true then the origin of the coordinate system will be raised or lowered to ensure that it lies on the surface of the model. That is it ensures that the point with coordinates (0, 0, 0) lies on the model surface.

viewer.rescale_depth true|false

Perform depth rescaling so that objects an infinite distance from the camera will be repositioned to the back of the depth buffer (distance **viewer.far_distance**). This can be used to circumvent the problem whereby even modern computers can only accurately render models in which **viewer.far_distance** is at most 1000 times the value of **viewer.near_distance** even though PANGU may need to render models from even greater distances. This technique imposes a large performance penalty and depth-buffer optimisation (**viewer.optimise_depth**) may be a better alternative.

viewer.show_landing_site true|false

If true the user-defined landing site mark will be displayed.

viewer.show_origin true|false

If true the surface coordinate origin mark will be displayed.

viewer.show_target true|false

If true the model-view target box will be displayed.

viewer.target *file w h d*

This setting enables the model-view target marker. The marker will be represented by a box with sides *w* units by *h* units by *d* units. The image in *file* will be used as a texture: quadrants of the image are used to texture each of the six sides.

viewer.zoffset *f*

This option allows the user to define the distance to raise or lower the coordinate origin by. Generally users should use the **viewer.recenter** option instead.

6.1.2.7 Rendering properties

boulder_shade flat | smooth

Normally boulders are flat shaded to emphasize their faceted nature.

fps_gamma *f*

The frame rate displayed by the view is computed using a decaying average. If the reported frame rate is currently P and a new frame was drawn at a rate F then the new reported frame rate will be *f**P + (1-*f*)*F. If *f* is zero then the reported frame rate is the speed at which the last frame was drawn and may vary too rapidly to read. As *f* approaches 1 sharp variations in the frame rate will be smoothed out.

shade flat | smooth

Normally the model should be viewed using smooth shading to disguise the polygonal mesh used.

viewer.angular_cutoff *f*

Objects whose angular size (in pixels) is smaller than f will not be displayed. This can be used to ensure that boulders are not displayed when the camera is far away from them. If *f* is 0 then objects will always be shown.

viewer.contour_map *f s*

The experimental 1D contour texture map will be loaded from the file *f* and applied with scaling *s*.

viewer.range_map *f s*

The experimental 1D range texture map will be loaded from the file *f* and applied with scaling *s*.

viewer.range_offset *f*

When the experimental range-map view is enabled (press / twice when in normal mode) the value *f* will be subtracted from the camera-model distance before range-mapping is applied. This can be used to achieve a better spread of range values.

viewer.roam_limit *f*

This value specifies the ROAM quality limit used to determine when to stop splitting the triangle mesh. A value of 1.0 is intended to prevent the error in a triangle from exceeding one pixel. The F key in the viewer can be used to adjust this value dynamically.

viewer.roam_pool *f*

This value defines the fraction of unused triangles that can remain in the ROAM allocation pool before the pool is trimmed in size. A typical value is 0.1.

viewer.roam_size_factor *f*

This value is used to scale the apparent size of a ROAM triangle on screen to prevent the ROAM algorithm from splitting triangles too small. The default is 1/400. The E key in the viewer can be used to adjust this value dynamically.

viewer.render dot | wire | filled

Normally the model is displayed using filled polygons. However a wire-frame view may be useful to enable the user to estimate the resolution of the model as seen by the camera. In dot mode the positions of the DEM sample points are drawn.

viewer.resolution_limit *f*

If part of the model has a low-resolution version and a high-resolution version then the viewer will switch between them when the angular size of triangles in the mesh are f pixels. This can be used to ensure that the model is rendered at the most appropriate resolution. If *f* is 0 then the highest-resolution will always be shown.

viewer.sky black | painted | raw | red | green | blue

This option enables users to specify the sky background type. The default is a uniform black background which may be explicitly obtained by the "black" value. A "painted" background may be used instead using the texture image defined by the **viewer.skymap** option although this background does not react to camera roll in the current version of the viewer and the texture may be rendered incorrectly (albeit very slightly) away from the centre of the image. A "raw" background is used to render individual stars loaded from a star catalogue (**viewer.star_catalogue**) as point sources. Users are unable to specify the relative brightness of stars at present. The remaining values of "red", "green" and "blue" are designed to assist horizon sensors: if the model is always rendered in grey-scale then the sky (and thus the horizon) may be trivially detected.

viewer.star_catalogue *f*

Positions and brightness of stars for the "raw" **viewer.sky** setting will be loaded from the file *f*.

viewer.skymap *f s*

The experimental star texture map for the "painted" **viewer.sky** setting will be loaded from the file *f* and applied with scaling *s*.

viewer.trigger_detail *f*

If the camera is less than *f* units above the model surface then surface detail texturing will be enabled. If the camera is more than *f* units above the surface then surface detail texturing will be disabled. Use negative or zero trigger values to disable detail triggering.

viewer.trigger_texture *f*

If the camera is less than *f* units above the model surface then surface texturing will be enabled. If the camera is more than *f* units above the surface then texturing will be disabled. Use negative or zero trigger values to disable texture triggering.

viewer.use_vertex_arrays true|false

This is an experimental setting which can sometimes double the rendering speed. However, note that the colours of boulders may be affected when this option is enabled so it must be used with caution.

6.1.2.8 Fog/dust cloud properties

viewer.fog.sky_distance *f*

This property defines the distance to the sky dome *i.e.* the distance associated with the sky. This is used to compute the correct amount of fog which is projected against the sky background. This is a new parameter which was introduced with PANGU 2.70 and overrides all other sky distance parameters.

viewer.global_fog.enable true | false

Enables/disables global fog.

viewer.global_fog.mode linear | exp | exp2

Defines the global fog extinction profile to be linear, exponential or exponential-squared. See Section 3.4.6 for the extinction equations used by these fog/dust modes.

viewer.global_fog.linear_start f_0

viewer.global_fog.linear_end f_1

This defines the range over which linear fog/dust is applied. Any point whose distance is less than f_0 will be unaffected by fog/dust; any point between f_0 and f_1 will be obscured by an amount based on its distance relative to f_0; any point beyond f_1 will be completely obscured by fog/dust. These parameters are ignored when an exponential fog/dust profile is used.

viewer.global_fog.density *f*

This property defines the density parameter of exponential fog (see the equations in Section 3.4.6). This property is ignored when linear fog/dust profile is used.

viewer.global_fog.colour *r g b*

This property defines the RGB colour of the fog/dust as *r*, *g* and *b*. The parts of the scene with maximum fog/dust extinction will be assigned this colour in the final image.

viewer.plane_fog.enable true | false

Enables/disables local-planar fog/dust.

viewer.plane_fog.density *f*

This property defines the exponential density parameter of local-planar fog/dust (see ection 3.4.6).

viewer.plane_fog.height *f*

This property defines the height of the fog/dust cloud.

viewer.plane_fog.colour *r g b*

This property defines the RGB colour of the fog/dust as *r*, *g* and *b*. The parts of the scene with maximum fog/dust extinction will be assigned this colour in the final image.

viewer.sphere_fog.enable true | false

Enables/disables local-spherical fog/dust.

viewer.sphere_fog.density *f*

This property defines the exponential density parameter of local-spherical fog/dust (see ection 3.4.6).

viewer.sphere_fog.radius *f*

This property defines the radius of the spherical fog/dust cloud.

viewer.sphere_fog.origin *x y z*

This property defines the location of the centre of the spherical fog/dust cloud as (*x*, *y*, *z*).

viewer.sphere_fog.colour *r g b*

This property defines the RGB colour of the fog/dust as *r*, *g* and *b*. The parts of the scene with maximum fog/dust extinction will be assigned this colour in the final image.

viewer.general_fog.enable true | false

Enables/disables general fog/dust clouds.

viewer.general_fog.density *f*

This property defines the exponential density parameter of general fog/dust (see ection 3.4.6).

viewer.general_fog.origin *x y z*

This property defines the location of the origin of the general fog/dust cloud as (*x*, *y*, *z*).

viewer.general_fog.colour *r g b*

This property defines the RGB colour of the fog/dust as *r*, *g* and *b*. The parts of the scene with maximum fog/dust extinction will be assigned this colour in the final image.

viewer.general_fog.model *f*

This property defines the name of the PANGU `.pan` file containing the model to be used to define the general fog/dust cloud volume. The model must contain one or more closed solids (*i.e.* like asteroid models rather than models generated from DEMs).

6.1.2.9 Save options

save_format *s*

This instructs the viewer to use the C `printf`() format string *s* to generate names for saved files. Each file name can be numbered starting from zero by including the two characters "`%d`" in the position where the number is required. To ensure that the file number always has *N* digits with leading zeros use "`%0`*N*`d`". For example, to ensure that every saved image is stored in the directory `images` with names `f_0000.ppm`, `f_0001.ppm` *etc.* use the format "`images/f_%04d.ppm`". Be extremely careful with this option and never use the `%` character in a name except to specify the file number as described here. If every image should be saved in the same file (*e.g.* `image.ppm`) then a format without a `%d` specification (such as "`image.ppm`") should be used.

viewer.save_scale *n*

Whenever an image is saved from the viewer it will be scaled to dimensions *n* times larger than the visible window on screen. The value of *n* must be an integer greater than 0. The default value is 1 (no scaling). This option can be used to generate images that are larger than the size of the screen on the host computer.

6.1.2.10 Blink comparison options

viewer.blink_delay *n*

The viewer will pause for *n* milliseconds between blinks when in blink comparison mode.

viewer.blink_image *f*

This instructs the viewer run in blink-comparison mode alternately displaying the PPM image *f* and the normal view of the model. The delay between "blinks" is set by **viewer.blink_delay**.

6.1.2.11 Animation/flight path options

viewer.auto_animation true|false

If set to true the viewer will continuously move the camera along a pre-defined trajectory. This option was designed for single layer 513x513 models and may not be appropriate or useful for other models.

viewer.flight_file *f*

Specifies the file to read flight file commands from in flight mode. See Section 6.1.4 for more details.

viewer.flight_mode true|false

If true the viewer will continuously move the camera along the trajectory defined by **viewer.flight_file**. See Section 6.1.4 for more details.

viewer.flight_speed *n*

This causes the viewer to skip *n* camera position/attitude changes in the current trajectory. This can be used to follow a specified trajectory more quickly if the viewer cannot render images fast enough.

viewer.save_frames true|false

If this option is true then the viewer will save each image generated while following a trajectory defined in **viewer.flight_file** to a file name generated from the **save_format** option.

6.1.2.12 Memory management options

viewer.cache_size *n*

If *n* is non-zero then a cache file will be created containing up to *n* bytes.

viewer.cache_ceiling *n*

If *n* is non-zero then the memory manager will attempt to ensure that no more than *n* bytes of cachable objects are kept in RAM at any time.

viewer.cache_burst_limit *n*

The memory manager will never write more than *n* objects into the cache in one attempt. This prevents the system from grinding to a halt when lots of objects need to be cached/uncached but may cause the cache ceiling setting to be violated.

viewer.cache_name_prefix *s*

The text "s" will be added to the front of the name used for the memory manager disk cache file. Use this setting to specify the directory in which the cache file is to be stored if it isn't the current directory. The default value is "`./.VIEWER_`".

viewer.cache_name_suffix *s*

The text "s" will be added to the end of the name used for the memory manager disk cache file. Use this setting to specify the file suffix. The default is "`.cache`".

6.1.2.13 Server options

viewer.remote_update true | false

If true then changes to the view made by remote PANGU clients will affect the user's camera.

server_mode true | false

viewer.server_mode true | false

If true the viewer will run in server mode and accept camera coordinates and return images over a network connection. See Section 6.1.5 for more details.

server_port *n*

viewer.default_port *n*

The viewer will listen for remote connections (TCP or UDP) on port *n* when in server mode.

server_tcp_port *n*

viewer.tcp_port *n*

The viewer will listen for remote TCP connections on port *n* when in server mode.

server_udp_port *n*

viewer.udp_port *n*

The viewer will listen for remote UDP connections on port *n* when in server mode (not supported yet).

6.1.2.14 Other options

depth_buffer_hack true | false

viewer.depth_buffer_hack true | false

On one Windows platform it has been discovered that the **viewer** only has access to a low-precision (16-bit) depth buffer while under Linux on the same hardware a high-precision (32 bit) depth buffer was available. A simple hack has been added to the **viewer** that might increase the depth buffer precision: on the test platform a 24-bit depth buffer was obtained by changing this setting to `true`. Note that on some platforms setting this to true may significantly reduce the depth buffer size and seriously affect image quality.

6.1.2.15 Note on texture maps

Texture maps must be square with a width that is an integral power of 2. Some systems may not be able to handle texture maps larger than 1024 pixels wide: 512 pixels wide is a good compromise. Obtaining suitable textures is difficult and so tools have been provided with the viewer to help (**mktexture** and **edtexture** which are described in Sections 6.1.7 and 6.5.2 respectively). Unfortunately applying a texture darkens the resulting images: this is unavoidable but can be alleviated to a certain degree using **edtexture**.

6.1.3 Command line options

Almost every aspect of the viewer can be controlled using command line switches. These override settings from any `pangu.ini` files. Note that switches which take no arguments such as -auto can be prefixed by "no" to negate their meaning e.g. -noauto. The notation <n> stands for a single integer, <m> for a mode keyword, <f> for a floating point number and <s> for a filename.

`viewer` `[glutopts] [-noini] [options] [<model>]`

where [`glutopts`] are XWindows/GLUT options (not Windows), <`model`> is the name of a model to display and where [`options`] represents one or more of:

6.1.3.1 Camera/viewpoint/general options

-att[itude] <f> <f> <f>	craft view camera yaw, pitch and roll angles.
-[no]boulders	[don't] show boulders.
-cam[era] <f> <f> <f>	model view camera distance, azimuth and elevation angles.
-cutoff <f>	angular cutoff limit (0 for no cutoff).
-[no]depth_hack	[don't] use a depth-buffer precision hack.
-[no]exp[ert]	[don't] start in expert mode.
-fov <f>	field-of-view (in degrees).
-[no]full	[don't] use full screen.
-hei[ght] <n>	define height of rendering area.
-move_step <f>	unit of movement/zoom.
-[no]optimise_depth	[don't] optimise the depth buffer.
-pos[ition] <f> <f> <f>	coordinates of craft view camera.
-[no]recenter	[don't] recenter the coordinate origin.
-[no]rescale	[don't] perform depth rescaling.
-res[olution] <f>	mesh resolution change over in pixels (0 for no change).
-roam_limit <f>	set ROAM quality limit to <f> pixels
-roam_size_factor < f>	set ROAM triangle size scale factor to <f> (default is 0.0025)
-roam_pool < f>	set ROAM triangle pool free-space faction to <f> (default is 0)
-[no]smooth	[don't] smooth shadow edges.
-sun <f> <f> <f>	distance, azimuth and elevation of Sun.
-targ[et] <f> <f> <f>	coordinates of model view target.
-view <m>	set view mode to <m> (model|craft)
-w[idth] <n>	define width of rendering area.
-zfar <f>	ignore objects farther from the camera than <f> units.

-znear <f>	ignore objects closer to the camera than <f> units.
-zoffset <f>	raise the coordinate origin by <f> units.

6.1.3.2 Texture mapping options

-btex[ture] <s> <f>	load boulder texture from file <s>, scale <f>
-con[tour_map] <s> <f>	load contour map from file <s>, scale <f>
-det[ail] <s> <f>	load detail from file <s>, scale <f>
-range_map <f>	range map from file <s>, scale <f>
-range_offset <f>	subtract <f> from camera ranges in range mapping mode
-nobtex[ture]	start without boulder texture
-nodet[ail]	start without detail texture
-notex[ture]	start without surface texture
-tex[ture] <s> <f>	load texture from file <s>, scale <f>
-trigger_detail <f>	enable surface detail when camera altitude is below <f> units
-trigger_texture <f>	enable surface texture when camera altitude is below <f> units

6.1.3.3 Illumination options

-amb[ient] <f> <f> <f>	set the ambient light colour
-col[our] <f> <f> <f>	set the Sun colour
-[no]round	[don't] use smooth shaded boulders (make them look round)
-hapke[_coeff] <f>	Use 1-<f> Lambert and <f> Hapke
-[no]flat	[don't] use flat shading for the model surface
-mode <m>	<m>=filled|wire|dot
-refl[ection] <m>	<m>=lambert|hapke

6.1.3.4 Shadow map options

-ign[ore_hash]	force-load a shadow map
-shadowmap <s>	load shadow map from file <s>

6.1.3.5 General fog/dust cloud options

-fog_sky_radius <f>	define the distance to the sky dome to be <f> units

6.1.3.6 Global fog/dust cloud options

-[no]global_fog	[don't] use global fogging
-global_fog_mode <m>	global fog mode: <m>=linear|exp|exp2
-global_fog_colour <r> <g> <b>	define the RGB colour of the fog/dust
-global_fog_start <f>	define the start of linear fog to be <f> units
-global_fog_end <f>	define the end of linear fog to be <f> units
-global_fog_density <f>	define the density of exponential fog to be <f>

6.1.3.7 Local-planar fog/dust cloud options

-[no]plane_fog	[don't] use local-planar fogging
-plane_fog_colour <r> <g> <b>	define the RGB colour of the fog/dust
-plane_fog_height <f>	define the height of the fog cloud to be <f> units
-plane_fog_density <f>	define the density of the planar exponential fog to be <f>

6.1.3.8 Local-spherical fog/dust cloud options

-[no]sphere_fog	[don't] use local-spherical fogging

-sphere_fog_colour <r> <g> <b>
define the RGB colour of the fog/dust
-sphere_fog_origin <x> <y> <z>
define the location of the centre of the fog/dust cloud
-sphere_fog_radius <f> define the radius of the fog cloud to be <f> units
-sphere_fog_density <f> define the density of the spherical exponential fog to be <f>

6.1.3.9 General shaped fog/dust cloud options

-[no]general_fog [don't] use general fogging
-general_fog_model <f> load the general fog volume from model file <f>
-general_fog_colour <r> <g> <b>
define the RGB colour of the fog/dust
-general_fog_origin <x> <y> <z>
define the location of the centre of the fog/dust cloud
-general_fog_density <f> define the density of the general fog volume to be <f>

6.1.3.10 Memory management options

-cache_size <n> set the disk cache size to <n> bytes
-cache_burst_limit <n> set the maximum number of objects for cache writing to <n>
-cache_ceiling <n> set the maximum amount of RAM to be used to be <n> bytes
-cache_prefix <s> the string <s> will be added to the front of the cache file name
-cache_suffix <s> the string <s> will be added to the end of the cache file name
-[no]mm [don't] use the memory management system
-[no]lazy [don't] attempt to load the model lazily using minimum RAM

6.1.3.11 Sky options

-sky <s> set sky type to <s> (raw|painted|black|white|red|green|blue)
-starcat[alogue] <s> load star catalogue from file <s> for sky mode "raw"
-skymap <s> load sky texture from file <s>
-skymap_scale <n> scale sky texture by $2^{<n-1>}$ (positive <n>) or $2^{<n>}$ (negative <n>)

6.1.3.12 Animation/blink comparison/snapshot options

-[no]a[uto] [don't] start in automatic fly-by mode
-blink <s> use <s> as a blink comparison image
-delay <d> wait <d> milliseconds between blinks
-faster increase speed of flight path travel
-flight <s> render using flight details from <s>
-log <s> write flight details to file <s>
-noflight don't process a flight path file
-[no]movie [don't] save every frame to disk
-[no]quit [don't] quit after rendering
-noshowflight don't render any flight path graphically
-save <s> save first frame to file <s>
-savefmt <s> use the C `printf()` format <s> for save names
-save_scale <n> saved images are <n> times larger than the window/screen size
-showflight <s> render the flight path from <s> graphically
-[no]verify_sun [don't] execute verify_sun commands in flight path files

6.1.3.13 PANGU server options

-port <n> use network port <n> in server mode
-[no]remote_update [don't] allow PANGU clients to affect the user's view.
-[no]server [don't] run in server mode

-tcpport <n> use TCP/IP port <n> in server mode
-udpport <n> use UDP port <n> in server mode (not supported yet)

6.1.3.14 Marker options

-[no]show_landing_site [don't] display the user-defined landing site marker
-[no]show_origin [don't] display the surface coordinate origin marker
-[no]show_target [don't] display the model-view target marker
-landing_site <s> <x> <y> <w> <z> <n>
Landing-site square marker texture <s>, centre <x> <y>, width <w>, elevation <z> and sample width <n>
-origin <s> <w> <z> <n> Origin square marker texture <s>, width <w>, elevation <z>, sample width <n>
-target_marker <s> <w> <h> <d>
Model-view target texture <s> and dimensions <w> <h> <d>

6.1.3.15 Other options

--help send a summary of the command line options to the error stream
-err[orfile] <s> send error output to file <s> (use + or - for stderr and stdout)
-err[orstream] <s> send error output to file <s> (use + or - for stderr and stdout)
-output_scale <s> all internal distance units are multiplied by <s> before they are displayed
-output_units <s> all distances are displayed with the units marker <s> (default "m")
-list_test[s] display the names of all available self-tests
-test <s> execute self-test <s> after start-up
-[no]vertex_arrays experimental feature to increase rendering speed (may affect colours though)
-report_keys display the effect of each key as it is pressed

The above options may be specified in any order.

6.1.3.16 INI file options

The `-noini` option prevents the viewer from processing default INI files (see Section 8.1). This must appear before any other PANGU command line options (after any GLUT options).

The `-ini <s>` option instructs the viewer to immediately load default settings from the file `<s>` before processing the command line arguments following it. To load a specific section *X* from the INI file foo.ini instead of the defaults use:

`-ini=`*X* `foo.ini`

6.1.3.17 GLUT options

Under XWindows the viewer accepts options which are passed directly to the GLUT toolkit (`[glutops]`). At the time of writing these are the standard X -geometry option and `-[in]direct` to force [in]direct OpenGL rendering. These options must appear before the viewer options described above.

6.1.3.18 Special note on the -errorfile/-errorstream options

Two special filenames can be used with the `-errorfile` and `-errorstream` options: `+` and `-`. The former sends error output to standard error while the latter sends it to standard output (normally the terminal window).

6.1.4 Flight path/trajectory files

The viewer can read a pre-defined flight path and display the resulting image sequence. This facility is only available from the command line. To create a flight path, run the viewer with the `-log` option specifying the name of the file to record the details in. Every change to the camera position and orientation will be recorded in the file until the viewer is terminated. A saved flight path can be displayed by using the `-showflight` option: it will appear as a yellow line. To replay a saved flight path use the `-flight` option. To save each frame of an animation specify the `-movie` option. You may need to adjust the **save_format** `pangu.ini` setting or use the `-savefmt` option.

For example, to save a viewer session in the file `foo.fli` use the command:

```
viewer -log foo.fli
```

All camera movements will be recorded in `foo.fli` until you quit the viewer. To generate movie stills from this file with each frame called `frame`*NNNN*`.ppm` (where *NNNN* is the frame number with leading zero digits):

```
viewer -movie -savefmt 'frame%04d.ppm' -flight foo.fli -quit
```

The -quit option instructs the viewer to terminate as soon as the last frame has been written.

For small surfaces the -auto command line option can be used to follow a pre-compiled flight path around the surface. This was designed for 513x513 surfaces and produces about 450 frames. Use the `-log` option to save it:

```
viewer –auto -log auto.fli -quit
```

Note that the format of flight path files may change in future versions of the viewer: image sequences for specific camera positions and trajectories would normally be obtained through the TCP/IP socket interface (see Section 6.1.5). PANGU v1.02 flight files consist of zero or more commands, one per line. Blank lines and lines beginning with "#" or "`//`" are ignored as are unrecognised or invalid commands.

The following commands are currently supported:

ambient_colour *r g b*	Set the colour and intensity of ambient light in the red green and blue channels to (*r*, *g*, *b*).
aspect_ratio *f*	Set the pixel aspect ratio to *f*.
boulder_colour *r g b*	Set the colour and intensity of boulders in the red green and blue channels to (*r*, *g*, *b*).
boulder_view *n t*	Set the boulder rendering method to *n* and enable or disable boulder texturing depending on whether *t* is true or false. If *n* is 0 then boulders will not be rendered; if it is 1 then they will be rendered with flat shading and if it is 2 then they will be rendered with smooth shading.
field_of_view *f*	Set the angular field of view width to *f* degrees.
quaternion *x y z q0 q1 q2 q3*	Position the camera at (*x*, *y*, *z*) and set the attitude according to the quaternion [*q0*, *q1*, *q2*, *q3*].
pause *N*	Pause for *N* seconds before processing the next command.
print *msg*	Display the text *msg* on the error/information stream of the viewer.
quit	Terminate the viewer immediately.
reset_frame_count	Reset the frame counter (used to generate save file names) to zero.
save	Save the current frame in the next file defined by the current save format (see the **save_format** INI option, -savefmt command line option or save_fmt flight file command).
save_as *f*	Save the current frame in the file *f*.
save_fmt *f*	Change the current save format to the C `prinf`() string *f*. See the **save_format** INI option for more details.
sky_type *n*	Set the sky rendering method to *n*. If *n* is 0 then a uniform black background is used; if it is 1 then the background will be painted with the sky texture; if it is 2 then raw stars will be rendered from the star catalogue; if it is 3, 4 or 5 then a uniform red, green or blue background will be used.

start *x y z a b c*	In model view mode this positions the camera target at (*x, y, z*) and sets the distance, azimuth angle and altitude angle to *a*, *b* and *c* respectively. In craft view mode the camera position is set to (*x, y, z*) and the yaw, pitch and roll are set to *a*, *b* and *c* respectively.
sun_colour *r g b*	Set the colour and intensity of Sun light in the red green and blue channels to (*r, g, b*).
surface_colour *r g b*	Set the colour and intensity of the surface in the red green and blue channels to (*r, g, b*).
surface_view *n t d*	Set the surface rendering method to *n*, enable or disable surface texturing depending on whether *t* is true or false and enable or disable surface detail depending on whether *d* is true or false. If *n* is 1 then the surface will be rendered with flat shading and if it is 2 then it will be rendered with smooth shading. Surface detail takes priority over surface texture.
verify_sun *r t p dr dt dp*	An error message will be displayed if the polar coordinates of the Sun are more than *dr* units from the radial distance *r*, *dt* degrees from the azimuth angle *t* and *dp* degrees from the elevation angle *p*. This can be used to avoid accidentally using the wrong Sun position for the current trajectory.
view *mode*	set the view mode to *mode* (model or craft)

Any other line is expected to contain six floating-point values using the same format as the **start** command. These values specify changes in the relevant camera properties. For example, in craft mode the command:

```
0 0 0      0 -10 0
```

will reduce the pitch angle by 10 degrees without changing the camera position, yaw or roll.

6.1.5 Server mode

The viewer can be run in server mode whereby remote clients can connect on a specific TCP/IP port and request a view of the current model for a given set of camera settings (position and attitude). Normal user interaction can occur while in server mode although the user will be interrupted when remote connections are processed. UDP support is being added to the viewer but has not yet been completed.

To start in server mode either change the **server_mode** setting in `pangu.ini` to true or use the `-server` command line option. The default port number is 10363 but this may be changed by the **server_port** `pangu.ini` setting or by the `-port` command line option.

A simple remote client for Windows and Linux is provided with the viewer and is described in Section 7.1.

6.1.6 Star catalogues

When rendering the sky background in “raw” mode the viewer uses star positions and intensities taken from a star catalogue. The catalogue file may be specified using the **viewer.star_catalogue** INI option in `pangu.ini` or on the command line with the `-star_catalogue` option. Each star is rendered as a single pixel although anti-aliasing by the display system may alter this slightly. In PANGU 1.50 there is no facility to allow users to define the brightness of stars relative to the primary light source. As a result stars will almost certainly be rendered at the wrong brightness (although the brightness of stars relative to other stars will be correct). A magnitude 2.26 star will be rendered using the highest possible pixel value to enable a reasonable number of stars to be seen. Additionally there is no option to specify the orientation of the star field with respect to the PANGU coordinate system. In PANGU 1.50 the star field will be display as it would be seen from the equator on Earth with the PANGU x-axis pointing east and the y-axis pointing north.

The format of the star catalogue is simple: it is a text file with one star definition per line. Each definition consists of three white-space separated columns specifying (in order) the azimuth angle of the star in degrees, the elevation angle of the star in degrees and the apparent magnitude. An electronic copy of the Yale Bright Star catalogue for epoch J2000 has been converted into this format and may be found in the `pangu/stars` directory. It contains 9096 bright stars visible to the naked eye; it excludes non-stars such as nebulae and galaxies (usually Messier objects) which are not point light sources. With PANGU 1.50 this catalogue will be displayed with the constellation of Orion visible in craft-view mode at yaw/pitch of (84°, 0°), the Pleiades cluster in Taurus at (58°, 24°) and the Big Dipper asterism (part of Ursa Major) at (187°, 58°). In model-view mode these coordinates are (96°, 0°), (122°, -24°) and (353°, -58°). Note that you may need to use the "U" key to raise the camera high enough above the model surface to see properly.

Use the **viewer.sky** INI option or the `-sky` command line option to enable the raw sky rendering mode when the **viewer** starts. When it is running press F5 to cycle through the different sky backgrounds.

6.1.7 Texture triggers

The viewer can now be instructed to enable or disable surface texture and/or surface detail depending on the current altitude of the camera relative to the model surface. When texture triggering is enabled (by specifying the `-trigger_texture`, `-trigger_detail` command line options or by specifying the **viewer.trigger_texture** or **viewer.trigger_detail** INI options) the viewer will first disable surface texture and surface detail. If the camera height is less than the trigger level for surface texture then surface texture will be enabled; if the camera height is less than the trigger level for surface detail then surface detail will be enabled instead. This means that by specifying a trigger level only for surface detail, for example, the viewer can automatically enable surface detail when the camera is close to the model and disable all texture when it is far from the model.

6.1.8 Landing site and coordinate origin markers

A feature introduced with PANGU 2.00 is the ability to mark the position of the coordinate origin of the model and a user-defined landing site. Additionally the position of the model-view target can also be marked. The origin and landing site markers are indicated by a square section which is curved to follow the model underneath it. The model-view target is a box since the target is free to move in 3D space.

The coordinate origin of a model is the defined by the 2D position (0, 0): users must define the physical size of the mark to be placed at the coordinate origin, a small vertical offset to ensure that the mark is visible above the surface and a sample width to define how accurately the underlying model is followed. If the physical distance between samples is less than the mesh resolution of the underlying model then the vertical offset may need to be increased to avoid the model surface penetrating the marker. If the marker samples match the underlying model precisely then an offset of 1cm or less can be used. Users must also specify the name of a texture file which will be rendered on top of the origin marker.

In a similar manner a user-defined landing site may be specified. The 2D position of the centre of the landing site must be specified in addition to the parameters for the origin marker.

The model-view target is represented by a box whose dimensions may be chosen by the user. Additionally a texture image file may be specified. The four quadrants of the image are used to paint over the top, bottom and two opposing sides of the box.

6.2 MKSHADOWS: CREATING SHADOW MAPS

To create a view-independent shadow map for a particular model, use the **mkshadows** tool. Its settings are found in the [mkshadows] section of `pangu.ini` and may also be specified on the command line. Note that shadow maps may add little or no detail to views of boulderless models with high Sun elevations (perhaps >45 degrees). This is because OpenGL shadows approximate true cast shadows at high illumination angles.

The shadow maps used in PANGU releases before release 1.50 contained only data with no header information. As a result the viewer would often terminate abnormally if the wrong shadow map was used with a model. In this release the shadow maps contain a code number designed to identify a model: if the code in the map does not match the code from the model then the viewer will ignore the shadow map. To generate shadow maps for use by older versions of the viewer you must use the `-old` command line option (see below).

Area light sources may be simulated by specifying the `-area_light` and `-rings` command line options along with a specification of the physical radius of the light source and the number of samples to take.

6.2.1 INI file settings

do_boulders true | false.

Boulders can significantly slow the shadow map generator so this option allows them to be ignored. Note that without shadows, boulders often appear to float above the planet surface at most viewing angles.

lazy_load true | false.

mkshadows.lazy_load true | false.

If true **mkshadows** will attempt to load the model lazily using the memory management system. This reduces the amount of memory used by **mkshadows** but may increase model loading times.

log_errors *f*

All error and information messages will be written to the file *f* (e.g. `PANGU.log`).

model *f*

Generate shadows for the model in the file *f* if none is specified on the command line.

overwrite true | false

If this is true the shadow map generator will overwrite any existing maps defined by **save_shadowmap**.

save_shadowmap *f*

The shadow map will be saved in the file *f* unless the user specifies an alternative name on the command line. If *f* already exists and the **overwrite** setting is false then the shadow map generator will not generate a map.

shadow_time_limit *f*

Shadow generation can take hours or even days for large models so this setting allows the user to restrict the amount of time allocated to shadow generation. If more than *f* seconds have been spent computing shadows then mkshadows will save the shadow information it as computed already and leave the rest unshadowed.

show_progress true|false

If this setting is true then a progress meter will be displayed indicating the fraction of shadow map points processed relative to the total number of shadow map points. This can be used to estimate the completion time of the program.

sun.position *r t p*

This specifies the Sun position in spherical polar coordinates: distance *r*, azimuth angle *t* and elevation angle *p*.

sun.radius *r*

This specifies the Sun radius as *r* units (for area light source approximations).

sun.sample_method point|rings

This setting controls the way area light sources are sampled. If the setting is “point” then the light source will be sampled once irrespective of the **sun.samples** setting. If it is set to “rings” then the visible disk will be sampled evenly using concentric rings of sample points.

sun.samples *n*

The area light source approximation will sample the disk of the Sun *n* times. Note that this may cause **mkshadows** to take approximately *n* times as long as it would take to produce a point source shadow map.

6.2.2 Command-line options

The command line argument syntax is:

```
mkshadows [-noini] [options] [<model>]
```

where <model> is model to generate the shadow map for. The [options] are:

6.2.2.1 Output options

-[no]boulders	[don't] test boulders for shadows
-[no]old	[don't] generate old-style shadow maps
-o[utput] <f>	write the modified texture to file <f> (can be the same as the input file)
-shadow_time <f>	restrict shadow map generation to <f> seconds

6.2.2.2 Illumination options

-[no]area[_light]	[don't] use an area light source approximation
-radius <r>	Sun radius <r> in same units as Sun distance
-[no]rings	[don't] use concentric rings for an area light source approximation
-samples <n>	Sample the Sun <n> times for the area light source approximation
-sun <r> <t> <p>	Sun at distance <r>, azimuth angle <t> and elevation angle <p>

6.2.2.3 Memory management options

-cache_size <n>	set the disk cache size to <n> bytes
-cache_burst_limit <n>	set the maximum number of objects for cache writing to <n>
-cache_ceiling <n>	set the maximum amount of RAM to be used to be <n> bytes
-cache_prefix <s>	the string <s> will be added to the front of the cache file name
-cache_suffix <s>	the string <s> will be added to the end of the cache file name
-[no]lazy	[don't] attempt to load the model lazily using minimum RAM

6.2.2.4 General options

--help	send a summary of the command line options to the error stream
-err[orfile] <f>	send error output to file <f> (use + or - for stderr and stdout)
-err[orstream] <f>	send error output to file <f> (use + or - for stderr and stdout)
-[no]progress	[don't] show a progress indicator

6.2.2.5 INI file options

-noini	prevent default INI file processing
-ini <f>	read INI settings from file <f> immediately
-ini=<s> <f>	read INI settings from section <s> of file <f> immediately

6.3 MERGESHADOWS: MODELLING AREA LIGHT SOURCES/PENUMBRA

The Sun is not a point source and so the shadow maps produced by **mkshadows** without the `-area_light` option can appear unreal (although smoothing by the viewer disguises this). If users wish to use their own method for area light approximation rather than the `-rings` method of mkshadows then they can run **mkshadows** multiple times with different Sun positions and combine the resulting shadow maps with **mergeshadows**. See Section 3.3.8 for an explanation and an example.

The settings for this program are in the [mergeshadows] section of `pangu.ini`.

6.3.1 INI file settings

load_shadowmaps *f1 f2 ...*

Read shadow maps from the files *f1*, *f2* etc. unless maps are specified on the command line. Each file name must be separated using white space not commas and all must have been generated from the same model. Use the \ continuation character at the end of a line to spread file names over multiple lines in the INI file.

log_errors *f*

All error and information messages will be written to the file *f* (e.g. `PANGU.log`).

overwrite true | false

If this is true the shadow map generator will overwrite any existing maps defined by **save_shadowmap**.

save_shadowmap *f*

The shadow map will be saved in the file *f* unless the user specifies an alternative name on the command line. If *f* already exists and the **overwrite** setting is false then the shadow map merger will not generate a map.

6.3.2 Command-line options

The command line argument syntax is:

```
mergeshadows [-noini] [options] [<map> [<map> ...]]
```

where <map> is a shadow map. The [options] are:

6.3.2.1 Output options

-verbose	display extra information about the contents of the shadow maps
-o[utput] <f>	save the merged shadow maps to the file <f>
-sample <f>	fix the sample count for the map as <f> (internal use only)

6.3.2.2 General options

--help	send a summary of the command line options to the error stream
-err[orfile] <s>	send error output to file <s> (use + or - for stderr and stdout)
-err[orstream] <s>	send error output to file <s> (use + or - for stderr and stdout)

6.3.2.3 INI file options

-noini	prevent default INI file processing
-ini <f>	read INI settings from file <f> immediately
-ini=<s> <f>	read INI settings from section <s> of file <f> immediately

6.4 IMGVIEW: VIEWING TEXTURES AND SCREEN SHOTS (WITH BLINK COMPARISON)

The PPM images produced by the viewer and the texture map generators might not be viewable in some graphics programs. The **imgview** tool is a simple OpenGL application for displaying bitmaps. Features of interest include the ability to enable and disable specific colour planes, support for blink comparison of two images and a simple video mode for viewing a sequence of images generated by the viewer from a trajectory. The settings are in the [imgview] section of `pangu.ini`. Note that if a valid blink comparison image is defined in the INI file then the image viewer will always run in blink comparison mode. The only way to prevent this is to specify `-blink` with a file which doesn't exist (or remove or invalidate the **blink_image** setting).

6.4.1 INI file settings

blink_delay *f*

Pause for *f* milliseconds between blinks.

blink_image *f*

The blink image to be displayed will be read from the file *f*.

full_screen true | false

If true the image viewer will use the full screen rather than a window.

log_errors *f*

All error and information messages will be written to the file *f* (e.g. `PANGU.log`).

main_image *f*

The image to be displayed will be read from the file *f*.

scale_to_fit true | false

If true the image viewer will scale the image to fit the chosen window size.

window_size *w h*

Use a default window size of *w* pixels wide and *h* pixels high.

6.4.2 Command-line options

The command line argument syntax is:

```
imgview [glutopts] [-noini] [options] [<file> [<file> …]]
```

where [`glutopts`] are XWindows/GLUT options and are described in Section 6.1.3.17 and where <`file`> is a PANGU texture file or a PPM image. The [`options`] are:

6.4.2.1 View options

-full	use the full screen rather than a window
-[no]scale	[don't] scale the image to fit the window
-w[idth] <f>	use a window <f> pixels wide
-hei[ght] <f>	use a window <f> pixels high

6.4.2.2 Blink comparison options

-blink <f>	use image <f> as comparison image (dimensions must match main image)
-delay <f>	pause <f> milliseconds between each blink

6.4.2.3 Animation/video options

-movie	all files named on the command line will be rendered in sequence

6.4.2.4 General options

--help	send a summary of the command line options to the error stream
-err[orfile] <s>	send error output to file <s> (use + or - for stderr and stdout)
-err[orstream] <s>	send error output to file <s> (use + or - for stderr and stdout)

6.4.2.5 INI file options

-noini	prevent default INI file processing
-ini <f>	read INI settings from file <f> immediately
-ini=<s> <f>	read INI settings from section <s> of file <f> immediately

6.5 MKTEXTURE: GENERATING NEW TEXTURE MAPS

As mentioned earlier, locating suitable texture maps for cratered astronomical bodies is not easy. To assist the user we have provided a tool (**mktexture**) to automatically generate textures in the correct format for the PANGU viewer by rendering a cratered digital elevation model taking the current Sun position and attitude into account.

This program should be used to obtain texture maps in which the Sun azimuth angle of the texture is within a few tens of degrees of the Sun azimuth angle used to view a model (so that the shadows are cast in the same direction). The elevation of the Sun should not be too low when generating texture maps otherwise the texture will be unrealistic and affect the images produced by the viewer. We suggest fixing the Sun elevation angle to 20 degrees irrespective of the elevation angle used in the viewer.

Settings for **mktexture** can be specified on the command line or in the [mktexture] section of `pangu.ini`. Currently only one digital elevation model is provided for generating textures (samples/texture_surface.ppm) and the sample `pangu.ini` file contains the relevant settings.

Texture maps may be viewed using the **imgview** and manipulated by **edtexture**.

6.5.1 INI file settings

ambient *r g b*

The RGB intensity of ambient light is *r*, *g* and *b*. This light falls evenly over the entire surface.

height_resolution *f*

The DEM used to generate the texture from stores elevations as scaled integer values. A scaled integer elevation of 1 corresponds to a true elevation of *f* units (e.g. metres).

log_errors *f*

All error and information messages will be written to the file *f* (e.g. `PANGU.log`).

mipmap_filter average | maxbias

This setting controls the generation of mipmap textures (when **texture_format** is set to **mip**).

overwrite true | false

If this is true the texture generator will overwrite any existing textures defined by **save_texture**.

quit_when_done true | false

If this is true the texture generator will terminate as soon as the texture has been saved.

save_texture *f*

The generated texture will be saved in the file *f* unless the user specifies an alternative name on the command line. If *f* already exists and the **overwrite** setting is false then the texture generator will not generate a texture.

show_progress true|false

If this setting is true then a progress meter will be displayed indicating the amount of work completed relative to the total amount of work to be done. This can be used to estimate the completion time of the program.

sun.colour *r g b*

sun_colour *r g b*

The RGB intensity of the Sun is *r*, *g* and *b*. Note that the **sun_colour** option is obsolete use **sun.colour**.

sun.position *r t p*

This specifies the Sun position in spherical polar coordinates: distance *r*, azimuth angle *t* and elevation angle *p*.

sun.radius *f*

To simulate a Sun with angular size set *f* to be the radius of the Sun using the same units as its distance and set the **sun.samples** setting to a value greater than 1.

sun.samples *n*

To simulate a Sun with angular size set *n* to be a number greater than 1 and define the **sun.radius** setting. A value of 3 or 5 for *n* is sufficient for most purposes: the Sun's disk will be sampled using an *n* by *n* grid.

surface.diffuse.colour *r g b*

surface_colour *r g b*

The fraction of RGB intensities reflected by the surface under the Lambert model is *r*, *g* and *b*. Note that the **surface_colour** option is obsolete and that **surface.diffuse.colour** takes precedence.

texture_format ppm | mip

This defines the file format used to store the texture: always use **ppm** for this release of PANGU. The **mip** format is an experimental PANGU texture format and won't be documented in this release.

texture_model *f*

Generate a texture using the DEM in file *f*.

texture_size *n*

The texture generated will be *n* pixels square where *n* must be an integral power of 2 (e.g. 128, 256, 512).

6.5.2 Command-line options

The command line argument syntax is:

```
mktexture [glutopts] [-noini] [options] [<dem>]
```

where `[glutopts]` are XWindows/GLUT options and are described in Section 6.1.3.17 and `<dem>` is the DEM defining the surface from which the texture is generated. The `[options]` are:

6.5.2.1 Output options

-filter <f>	apply the box filter <f> when generating mip-map textures
-format <f>	output file format (<f> must be ppm or mip; use ppm for this release)
-o[utput] <f>	write the generated texture to the file <f>
-[no]quit	[don't] terminate after saving the generated texture

6.5.2.2 Texture/view options

-sun <r> <t> <p>	Sun at distance <r>, azimuth angle <t> and elevation angle <p>
-w[idth] <n>	Generate a texture <n> pixels by <n> pixels
-z[scale] <f>	The height resolution of the DEM is <f> units per elevation step

6.5.2.3 Illumination options

-amb[ient] <r> <g> <b>	RGB intensity of ambient light is <r>, <g> and <b>
-col[our] <r> <g> <b>	RGB intensity of the Sun is <r>, <g> and <b>
-surface <r> <g> <b>	RGB colour of the surface is <r>, <g> and <b>
-sample <n> <r>	Sample the Sun <n> by <n> times over a radius of <r>

6.5.2.4 General options

--help	send a summary of the command line options to the error stream
-err[orfile] <s>	send error output to file <s> (use + or - for stderr and stdout)
-err[orstream] <f>	send error output to file <f> (use + or - for stderr and stdout)
-[no]progress	[don't] show a progress indicator

6.5.2.5 INI file options

-noini	prevent default INI file processing
-ini <f>	read INI settings from file <f> immediately
-ini=<s> <f>	read INI settings from section <s> of file <f> immediately

6.6 EDTEXTURE: BRIGHTENING TEXTURE MAPS

The textures produced by **mktexture** often cause the viewer to produce very dark images. Increasing the Sun intensity or surface colour won't help. To solve this problem the **edtexture** tool can be used to scale the texture map colours. Multiplying by a factor of 1.2 or 1.5 is usually sufficient. Settings for the **mktexture** program can be specified on the command line or in the "[edtexture]" section of `pangu.ini`. Note that the editing settings are cumulative: if a scaling of 1 and a division of 2 is specified the effect is a division by 0.5. Also beware that settings from INI files will still be applied if they are not overridden on the command line. Thus if **texture_divide** is 2 and -scale 2 is specified on the command line then the texture will not be changed.

6.6.1 INI file settings

load_texture *f*

Read the texture to modified from the file *f* if none is specified on the command line.

log_errors *f*

All error and information messages will be written to the file *f* (e.g. `PANGU.log`).

overwrite true | false

If this is true the texture editor will overwrite any existing textures defined by **save_texture**.

save_texture *f*

The modified texture will be saved in the file *f* unless the user specifies an alternative name on the command line. If *f* already exists and the **overwrite** setting is false then the texture editor will not modify a texture.

texture_divide *f*

Divide the value of each pixel in the texture map by the factor *f*. If *f* is -1 then no division will occur.

texture_gamma *f*

Apply a gamma correction of *f* to the value of each pixel in the texture map. If *f* is -1 then no gamma correct will occur.

texture_multiply *f*

Multiply the value of each pixel in the texture map by the factor *f*. If *f* is -1 then no scaling will occur.

6.6.2 Command-line options

The command line argument syntax is:

`edtexture` `[-noini] [options] [<texture>]`

where `<texture>` is texture file to modify. The `[options]` are:

6.6.2.1 Output options

-div[ide] <f>	divide every pixel value by <f>
-gamma <f>	apply a gamma correction of <f> to the image
-info	send information about the texture to the error stream
-mul[tiply] <f>	multiply (scale) every pixel value by <f>
-o[utput] <f>	write the modified texture to file <f> (can be the same as the input file)
-sca[le] <f>	multiply (scale) every pixel value by <f>

6.6.2.2 General options

--help	send a summary of the command line options to the error stream
-err[orfile] <s>	send error output to file <s> (use + or - for stderr and stdout)
-err[orstream] <s>	send error output to file <s> (use + or - for stderr and stdout)

6.6.2.3 INI file options

-noini	prevent default INI file processing
-ini <f>	read INI settings from file <f> immediately
-ini=<s> <f>	read INI settings from section <s> of file <f> immediately

6.7 PANDUMP: CHECKING PANGU OBJECT FILES

A .pan file can be checked for sanity without loading it into the viewer by using the **pandump** tool. The creation and modification times of a .pan file can also be seen. Its settings are found in the "[pandump]" section of `pangu.ini`:

6.7.1 INI file settings

log_errors *f*

All error and information messages will be written to the file *f* (e.g. `PANGU.log`).

model *f*

Read the model from the file *f* unless one has been specified on the command line.

overwrite true | false

If this is true then pandump will overwrite existing files defined by **pandump.output_file**.

output_file *f*

pandump.output_file *f*

The details of the model will be written to the file *f* unless the user specifies an alternative name on the command line. If *f* already exists and the **overwrite** setting is false then pandump will do nothing. Note that the **output_file** setting is obsolete and **pandump.output_file** takes precedence.

6.7.2 Command-line options

The command line argument syntax is:

pandump `[-noini] [options] [<model>]`

where <`model`> is a PANGU model. The [`options`] are:

6.7.2.1 Output options

-o[utput] <f>	save the information to the file <f> (or stdout if <f> is "-")

6.7.2.2 General options

--help	send a summary of the command line options to the error stream
-err[orfile] <s>	send error output to file <s> (use + or - for stderr and stdout)
-err[orstream] <s>	send error output to file <s> (use + or - for stderr and stdout)

6.7.2.3 INI file options

-noini	prevent default INI file processing
-ini <f>	read INI settings from file <f> immediately
-ini=<s> <f>	read INI settings from section <s> of file <f> immediately

6.8 PAN2POV: CREATING POV-RAY SCRIPTS FROM PANGU OBJECT FILES

Occasionally users may wish to compare PANGU viewer images with those produced by the POV-Ray ray tracer. To assist this task the **pan2pov** tool will naively convert a .pan file into a POV-Ray script. A sample .pov file may be created using settings obtained by pressing P in the viewer and then rendered using POV-Ray. The settings for this tool are in "[pan2pov]" of `pangu.ini`.

Note that each boulder in a model will be represented by its own POV-Ray object rather than defining three library objects and representing boulders as transformations of those objects. This means that **pan2pov** may generate massive files for models containing boulders.

Also note that POV-Ray may not be able to read huge mesh objects efficiently: careful use of the `-max` option can reduce the time it takes POV-Ray to parse the mesh file by a factor of 20 or more.

6.8.1 INI file settings

do_boulders true | false.

Boulders can cause massive POV-Ray files to be generated so this option allows them to be ignored.

log_errors *f*

All error and information messages will be written to the file *f* (e.g. `PANGU.log`).

model *f*

Read the model from the file *f* unless one has been specified on the command line.

overwrite true | false

If this is true then pan2pov will overwrite existing files defined by **save_povray**.

povray_smooth true | false

If this is true then pan2pov will generate a smooth-shaded POV-Ray mesh.

save_povray *f*

The POV-Ray script will be written to the file *f* unless the user specifies an alternative name on the command line. If *f* already exists and the **overwrite** setting is false then a POV-Ray file will not be created.

surface.diffuse.colour *r g b*

surface_colour *r g b*

The fraction of RGB intensities reflected by the surface under the Lambert model is *r*, *g* and *b*. Note that the **surface_colour** option is obsolete and that **surface.diffuse.colour** takes precedence.

6.8.2 Command-line options

The command line argument syntax is:

pan2pov `[-noini] [options] [<model>]`

where <`model`> is a PANGU model. The [`options`] are:

6.8.2.1 Output options

-[no]boulders	[don't] include boulders in the POV-Ray file
-col[our] <r> <g> <b>	RGB colour of the surface is <r>, <g> and <b>
-max <f>	generate a maximum of <f> triangles per output object
-o[utput] <f>	write the POV-Ray script to the file <f>
-smooth	generate a mesh using smooth-shaded triangles

6.8.2.2 General options

--help	send a summary of the command line options to the error stream
-err[orfile] <s>	send error output to file <s> (use + or - for stderr and stdout)
-err[orstream] <s>	send error output to file <s> (use + or - for stderr and stdout)

6.8.2.3 INI file options

-noini	prevent default INI file processing
-ini <f>	read INI settings from file <f> immediately
-ini=<s> <f>	read INI settings from section <s> of file <f> immediately

7. REMOTE TCP/IP ACCESS TO PANGU SERVERS

As mentioned elsewhere in this manual, the PANGU viewer can be run in server mode. This means that it will accept TCP/IP connections from remote PANGU clients, read camera position and attitude information and return an image of the view of the current model from the specified position. Currently the image is always in PPM format and the image dimensions match those of the viewer rendering area precisely. This means that resizing the viewer window will change the dimensions of images returned to remote clients.

A sample program **pangu_client** has been provided to demonstrate the PANGU TCP/IP server facility which can connect to a PANGU server (such as the **viewer** application running in server mode) and obtain a sequence of images from any camera position and attitude.

7.1 PANGU_CLIENT: AN OPENGL PANGU CLIENT

This is a command-line driven application which displays a single image from a PANGU server in an OpenGL window. Pressing the arrow keys will move change the camera attitude and retrieve a new image from the server. The name or address of the server is specified on the command line (if the server is not running on the same host) along with the camera position and orientation. When the program is run it will connect to the specified server, retrieve an image corresponding to the chosen camera settings and display it in an OpenGL window. The program can be instructed to use version 0 or version 1 of the PANGU network protocol (see Section 7.3). The command line argument syntax for is tool is:

```
pangu_client [glutops] [-noini] [options]
```

where [`glutopts`] provides control over the positioning and rendering mode of the client display window:

-geometry +xorig+yorig:	position of the left corner of the window
-[in]direct:	use (in)direct X/OpenGL rendering.

and where [`options`] provides control over other aspects of the window:

7.1.1 INI file settings

log_errors *f*

All error and information messages will be written to the file *f* (e.g. `PANGU.log`).

pangu_client.flight_file *f*

Read and process commands from the flight file *f*.

free_camera *x y z yaw pitch roll*

pangu_client.free_camera *x y z yaw pitch roll*

The initial camera position will be (*x, y, z*) with attitude (*yaw, pitch, roll*).

full_screen true|false

pangu_client.full_screen true|false

If this setting is true then images will be displayed in full-screen mode.

pangu_client.protocol *n*

Use PANGU network protocol *n* when communicating with the remote server.

save_frames true|false

pangu_client.save_frames true|false

If this setting is true then images will be saved to disk using the current save format.

save_format *fmt*

pangu_client.save_format *fmt*

Defines the current save format using C `printf()` notation. The default is "`scr_%03d.ppm`" which means that files will be saved as `scr_000.ppm`, `scr_001.ppm` *etc.*

scale_to_fit true|false

pangu_client.scale_to_fit true|false

If this setting is true then images will be scaled to fit the current window or screen size.

server_name *f*

pangu_client.server_name *f*

Connect to the remote server *f*.

server_port *n*

pangu_client.server_port *n*

Connect to the remote server on network port *n*.

window_size *w* *h*

pangu_client.window_size *w* *h*

The initial window size will be *w* pixels wide and *h* pixels high.

7.1.2 Command line options

7.1.2.1 Camera/viewpoint options

-att[itude] <y> <p> <r>	define camera attitude (yaw, pitch, roll) as (<y> <p> <r>)
-full	use full screen
-hei[ght] <n>	define window height in pixels
-pos[ition] <x> <y> <z>	define camera position <x> <y> <z>
-[no]scale	[don't] scale image to fit window
-w[idth] <n>	define window width in pixels

7.1.2.2 Network options

-port <n>	define host network port
-proto[col] <n>	use PANGU network protocol <n> (0 or 1)
-server <n>	define host server or IP address

7.1.2.3 Flight file options

-flight <f>	process flight file <f>
-[no]quit	[don't] quit after processing the flight file
-[no]save	[don't] save images obtained from flight file processing
-savefmt <s>	format for naming images from flight file processing
-trajectory <s>	alias for -flight <s>

7.1.2.4 General options

--help	send a summary of the command line options to the error stream
-err[orfile] <s>	send error output to file <s> (use + or - for stderr and stdout)
-err[orstream] <s>	send error output to file <s> (use + or - for stderr and stdout)

7.1.2.5 INI file options

-noini	prevent default INI file processing
-ini <f>	read INI settings from file <f> immediately
-ini=<s> <f>	read INI settings from section <s> of file <f> immediately

7.1.3 Flight File Processing

Version 2.00 of **pangu_client** is able to read and process a flight file (see Section 6.1.4). Each change in the camera position or attitude from the flight file will cause **pangu_client** to request a new image from the PANGU server. If the `-save` command line option is specified or if the **pangu_client.save_frames** INI option is set to true then retrieved images will be saved in a file whose name is based on the current save format.

The default save format is "`scr_%03d.ppm`" where the `%03d` will be replaced by the three digit integer frame number counting from 0. For example, the first image will be called `scr_000.ppm`, the second `scr_001.ppm` *etc*. Use the `-savefmt` command line option or **save_format** INI option to change the save format. For example, to save each image in the directory images with names `f_0.ppm`, `f1.ppm` *etc.* use the format "`images/f_%d.ppm`".

To process a flight file use the `-flight` command line option or the **pangu_client.flight_file** INI option. Note that images obtained in this way will not be saved unless either the `-save` command line option or the **pangu_client.save_frames** INI option are specified.

7.2 NOTES

If you are running the client and the viewer as a PANGU server on the same machine you must ensure that the **viewer** window is not obscured. Otherwise the client may receive partial or damaged images.

7.3 THE PANGU NETWORK PROTOCOL (TECHNICAL DETAILS)

The client connects to the server on port 10363 and sends the protocol version number it wants to use to communicate with the server. If the server cannot handle the requested protocol it must either disconnect immediately or it send back an error record in a format that the client can understand. Note that a client may request protocol version 0.00 but a server might only support version 1.00. In which case the server must be careful about how it replies (disconnect or send a 0.00 error record).

The protocol version number is an unsigned long integer. If the version number is written as MAJOR.MINOR (e.g. 1.32) then the value to send is MAJOR * 256 + MINOR (e.g. 1*256 + 32 = 288).

All integer values are passed in network byte order. That means the client must use `htonl()` to encode all long integer values and use `ntohl()` to decode them. Long integers are four bytes long irrespective of the size of machine word size.

7.3.1 Version 0.00

Version 0.00 of the protocol is the original protocol and is now obsolete. It is described below for completeness:

The client sequentially:

- connects to server on port 10363 (or some other agreed port)
- sends the unsigned long integer protocol version number (0)
- sends the position and orientation of the viewpoint as a sequence of six signed long integers (x y z p q r) where x, y and z are the camera position and p, q and r are pitch, yaw and roll orientation respectively. The pitch, yaw and roll are measured in degrees.
- reads the unsigned long integer response code

A response code of zero indicates that the server accepts the specified protocol and has generated an image from the specified camera position. A non-zero response code indicates an error.

If an error did occur the client must sequentially:

- read 256 bytes of error message
- treat the error message as a human-readable NUL-terminated text string
- disconnect

If an error did not occur the client must sequentially:

- read the image file length (an unsigned long integer)
- read that many bytes of image file data

- disconnect

The current version of the server will only return images in PPM raw format. However, clients MUST treat the image file just as they would a file loaded from disk by checking the magic number etc.

7.3.2 Version 1.06

Version 0.00 of the protocol is somewhat limited. Positions and orientations must be integer values, error messages are intended to be human rather than machine-readable and the client/server interaction is almost non-existent. Version 1.06 of the protocol (indicated by sending the long integer version 256+6 or hexadecimal 0x106) is more flexible and will hopefully be future-proof.

The basic model is an object-oriented message-passing system: clients send a message identified by name (number) containing the parameters they wish to use. The server sends a reply also identified by name (number) containing the resulting data. The minor version number of the protocol may be changed as new messages are added to the protocol.

Since many servers may handle at most one client at a time, clients are expected to send their messages and disconnect as quickly as possible. However, if a server is to be used in a high-speed closed-loop situation then the client may wish to remain connected all the time.

The outline of a simple client-server exchange is:

- Client:
  - connects to server
  - sends the four-byte integer protocol version 256
- Server:
  - reads the four-byte integer protocol version
  - if the version is acceptable then an Okay message is returned
  - otherwise an Error message is returned or the connection dropped
- Client:
  - reads the server response (**Okay** or **Error**)

If the response is an **Error** then the client must disconnect. Otherwise the client is allowed to enter a loop sending a message to the server and reading the reply. When finished the client may send a **Goodbye** message or simply disconnect. Note that for every message the client sends to the server a reply message must be read from the server before proceeding.

All messages are transmitted by sending the name (number) of the message as a long integer followed by any parameters/data associated with the message. The format of client and server messages is defined below using the layout:

MessageName (MessageNumber)
Parameters:
x:long (explanation/meaning)
y:float (explanation/meaning)
z:double (explanation/meaning)
Expected response (only for client messages):
Okay
Description:
This message is used to send an integer x, single-precision float y and double-precision float z to the server.
Rationale (client messages only):
Why this message type is necessary.

If the client wished to send the message above (assuming it is number 23) then the client would send the long integer 23 followed by the long integer value of x, the single-precision value of y and the double-precision value of z. It would then read the response which must be **Okay** or **Error** (any message may generate an **Error** response). Server responses, like client requests, consist of the message number as a long integer followed by any data.

The format of data types used to describe this protocol version are:

byte: 8 bit signed integer

boolean: represented as an 8-bit byte: zero means false, non-zero means true

short: 16 bit signed integer sent in network byte order

long: 32 bit signed integer sent in network byte order

float: 32 bit single-precision floating point number encoded into a 32 bit long sent as a long integer. Bits 0-7 hold the 127-biased exponent, bit 8 holds the sign bit and bits 9-31 hold the 23 bit mantissa. The implicit 1 in the mantissa is not stored.

string: the length stored as a short followed by that many bytes stored as a NUL terminated string. For convenience the string length (including NUL terminator) will be padded to an even number.

Unsigned versions of the integer types are ubyte, ushort and ulong. Items enclosed in {} explain the data type in words.

7.3.2.1 Server response messages

Below we define the name and format of the server response messages.

Okay (0)

Parameters:

(none)

Description:

Used to indicate that a request has been accepted and processed without error.

Error (1)

Parameters:

`code:long`

`msg:string`

Description:

Used to indicate unsuccessful message receipt/processing. The code number is a unique error identifier used to assist electronic clients. The msg string is a human readable description of the error and may vary between responses which share the same code. Codes that are defined at the time of writing are:

0: generic/unspecified error: see the msg field for details
1: protocol version not supported
2: message number not recognised
3: badly formed message

Image (2)

Parameters:

`size:long`

`data:`{size bytes of data representing the image file}

Description:

Used to return an entire file of image data. The client must examine the file to determine the format used. For example, image data may be returned as PPM bitmaps or GIFs or a special PANGU image format.

Float (3)

Parameters:

`x:float`

`f:boolean`

Description:

Return a single floating point value. Normally the result of a **GetElevation** request message. If `f` is true then the value is valid: this is used to indicate whether a lookup request was successful or not.

FloatArray (4)

Parameters:

`n:long` (number of points in the vector)
`x0:float`, `f0:boolean` (first value)
`x1:float`, `f1:boolean` (second value)

Description:

Return an array/vector of **Float** values.

3DPoint (5)

Parameters:

`x:float`, `y:float`, `z:float` (3D point)
`f:boolean`

Description:

Return a single 3D position. Normally this would be the response to a **LookupPoint** request message. The point is only valid when `f` is true.

3DPointArray (6)

Parameters:

`n:long` (number of points in the vector)
`x0:float`, `y0:float`, `z0:float`, `f0:boolean` (first 3D point)
`x1:float`, `y1:float`, `z1:float`, `f1:boolean` (second 3D point)
...

Description:

Return an array/vector of **3DPoint** values.

MemoryBlock (7)

Parameters:

`size:long`
`data:`{size bytes of data representing the memory block}

Description:

Used to return a block of data. Identical to **Image** except that there is no pre-defined meaning for the data. Clients may wish to read **Image** messages as **MemoryBlock** messages and examine the contents later.

EchoReply (8)

Parameters:

`size:long`
`data:`{size bytes of data being returned}

Description:

Used to return a block of data that was sent to the server via an **Echo** message. The format is identical to **MemoryBlock** so clients may wish to read the data as a **MemoryBlock** message.

LidarPulseResult (9)

Parameters:

`range:float`
`cos_theta:float`

Description:

The range to the surface is returned along with the cosine of the incidence angle. If the beam does not intersect with any surface then the range will be a negative value.

LidarMeasurement (10)

Parameters:

{32 words containing the SetLidarParameters parameter values}
`nbytes:ulong` (number of bytes in the rest of the message)
`r0:float`, `c0:float` (result pair for the first pixel)
`r1:float`, `c1:float` (result pair for the second pixel)
…

Description:

The header parameters define the LIDAR emitter/receiver properties used to generate the LIDAR image in SetLidarParameters format. After the samples parameter there will be a size field (`nbytes`) which defines the number of bytes used by the (`ri`,`ci`) pairs including any padding bytes. This field is followed by `k` blocks of (`nx`*`ny`)*(`n`*`m`) pairs of LIDAR pulse results (`ri`, `ci`) where `k` is the number of bits set in the `flags` field of **SetLidarParameters**. Each block is returned in order of the bits in the `flags` field of **SetLidarParameters**. For example, if bit 2 is set then `k`=1 and only the corner cube range/slope block will be returned. If bits 1 and 2 are set then `k`=2 and the azimuth/elevation block will be returned followed immediately by the corner cube range/slope block. For blocks containing range/slope values the first value of each pair is the range and the second is the slope. For blocks containing azimuth and elevation values the first value of each pair is the azimuth and the second is the elevation. Negative ranges indicate that there was no beam/surface intersection. For blocks containing time-of-pulse emission results the time is stored in the first value of each pair: the second value in each pair is unused. For type 0 and 1 the result blocks match a rectangular pixel array starting from the top left corner with the coordinate axes in the centre of the image, the x-axis increasing to the right and the y-axis increasing upwards. Note that results for the type 0 and 1 raster scans are returned in pixel order not scan order (type 1 is a zig-zag scan). Future scan modes may return results in a different format.

RadarResponse (11)

Parameters:

`status:ulong` (result code with 0 meaning success)
`maxv:float` (maximum strength value in the response)
`totv:float` (total strength of the response)
`offr:float` (range offset: the range associated with the first range histogram bin)
`offs:float` (speed offset: the speed associated with the first speed histogram bin)
`rbwid:float` (width of the range histogram bins)
`sbwid:float` (width of the speed histogram bins)
`minr:float` (minimum range value sampled)
`maxr:float` (maximum range value sampled)
`mins:float` (minimum speed value sampled)
`maxs:float` (maximum speed value sampled)
`used:ulong` (number of samples used *i.e.* hit the model and lie within histogram limits)
`nr:ulong` (number of range bins in the result histogram)
`ns:ulong` (number of speed bins in the result histogram)
{10 words reserved for future use}
{`nr`*`ns` floats, one for each histogram bin}

Description:

This message defines the 2D histogram representing a RADAR response: the individual samples from the footprint of the RADAR beam are collected into a 2D array based on the range and relative speed of the sample point on the model. The `status` field of this message is used to indicate success/failure: the value will be zero to mean success. The `maxv` field provides the maximum signal value in the response so that the client doesn't need to scan the histogram to find it; similarly the `totv` field provides the sum of all the histogram bins and would normally be 1. The `offr` and `offs` fields define the range associated with the start of the first range bin and the speed associated with the start of the first speed bin. The `rbwid` and `sbwid` fields define the width of the range and speed histogram bins respectively. The end of the first

range bin is defined by `offr+rbwid` and the end of the first speed bin by `offs+sbwid`. The `minr` and `maxr` fields define the minimum and maximum ranges obtained from all the samples while the `mins` and `maxs` fields define the minimum and maximum speeds obtained from all the samples. The `used` field indicates the number of samples that are included in this histogram. Samples corresponding to parts of the RADAR footprint which didn't intersect with the model are not used nor are samples which lie outside the histogram limits. The number of range and speed bins in the histogram are returned in the `nr` and `ns` fields respectively. The 2D histogram is represented as a single array of floating point values. The first `nr` floats define the range histogram values for the first speed histogram bin. The next `nr` floats define the range histogram values for the second histogram bin and so on. If `ns` is one then the results will be a contiguous sequence of `nr` range values; if `nr` is one then the results will be a continguous sequence of `ns` speed values.

7.3.2.2 Client request messages

Below we define the name and format of the client request messages.

Goodbye (0)

Parameters:
(no parameters)
Expected response:
(none)
Description:
Tell the server that we are going to disconnect and that they can drop their end of the connection.
Rationale:
Provided for robustness.

GetImage (1)

Parameters:
(no parameters)
Expected response:
Image(2) (can be read as a **MemoryBlock(7)**)
Description:
Ask the server for the view from the current viewpoint and camera settings.
Rationale:
Image retrieval is an important requirement for VBNAT and other systems.

GetElevation (2)

Parameters:
(no parameters)
Expected response:
Float(3)
Description:
The value returned defines the altitude of the camera relative to the surface. This is the vertical distance from the camera to the surface measured along Z_LDS.
Rationale:
Allow true, surface-relative altitude to be obtained.

GetElevations (3)

Parameters:

`n:ulong` (number of 2D points to look up)

`x0:float`, `y0:float`, `z0:float` (coordinates of the first camera position)

`x1:float`, `y1:float`, `z1:float` (coordinates of the second camera position)

...

Expected response:

FloatArray(4)

Description:

The array of elevations maps directly to the input array of camera positions.

Rationale:

Reduce network overheads by grouping **GetElevation** requests.

LookupPoint (4)

Parameters:

`x:float`, `y:float` (coordinates of the point)

Expected response:

3DPoint(5) corresponding to the 3D position of the input image point.

Description:

Convert 2D screen coordinates (`x`,`y`) into a 3D position of the closest point of the surface of the model under the specified 2D point. The position (0,0) is the bottom left corner of the bottom left pixel of the image and (1,1) is the top-right corner of the top-right pixel.

Rationale:

Allow line-of-sight surface position measurements to be calibrated.

LookupPoints (5)

Parameters:

`n:ulong` (number of 2D points to look up)

`x0:float`, `y0:float` (coordinates of the first point)

`x1:float`, `y1:float` (coordinates of the second point)

...

Expected response:

3DPointArray(6) corresponding to the 3D positions of the specified points.

Description:

Same as **LookupPoint** except that `n` points are processed in one step.

Rationale:

Reduce network overheads by grouping **LookupPoint** requests.

GetPoint (6)

Parameters:

`x:float`, `y:float`, `z:float` (direction to look along)

Expected response:

3DPoint(5) corresponding to the 3D position visible along (`x`,`y`,`z`).

Description:

A ray is projected from the current camera position along the direction (`x`,`y`,`z`) and the point on the model hit by the ray is returned. If the ray does not hit the model then the returned point will have the valid flag set to false. Can be regarded as the first step towards LIDAR support.

Rationale:

View-independent version of **LookupPoint**.

GetPoints (7)

Parameters:

`n:ulong` (number of directions to look along)
`x0:float`, `y0:float`, `z0:float` (first direction)
`x1:float`, `y1:float`, `z1:float` (second direction)
...

Expected response:

3DPointArray(6) corresponding to the 3D positions visible.

Description:

Same as **GetPoint** except that `n` points are processed in one step.

Rationale:

Reduce network overheads by grouping **GetPoint** requests.

Echo (8)

Parameters:

`size:ulong` (number of bytes to send)
`data:{size` bytes of data}

Expected response:

EchoReply(8) containing the data sent to the server.

Description:

This message is used to send data to the server which is returned to the client in the reply.

Rationale:

Can be used to test the speed of the connection or to check that the server is still listening.

GetRangeImage (9)

Parameters:

`offset:float`
`scale:float`

Expected response:

Image(2) (can be read as a **MemoryBlock(7)**)

Description:

Ask the server for a range image from the current viewpoint and camera settings. The colour of each pixel in the range image determines the range to a surface along the direction covered by that pixel. If the physical range is R then D = (R - `offset`)×`scale`. D is then clamped to lie in the range [0,1] so that D<0 becomes 0 and D>1 becomes 1. If the range texture has W pixels then the colour assigned to the pixel will be round(D×(W-1)). Use **GetRangeTexture** to determine the value of W and the colour map used.

Rationale:

Useful for simple LIDAR image sensor simulation.

GetRangeTexture (10)

Parameters:

(no parameters)

Expected response:

Image(2) (can be read as a **MemoryBlock(7)**)

Description:

Ask the server for the 1-D image used for range textures. The width and pixel information of this image can be used to reverse-map the images obtained by **GetRangeImage**. If the returned image has width (and height) of zero then range texture mapping is not available.

Rationale:

Useful for simple LIDAR image sensor simulation.

GetViewpointByAngle (11)

Parameters:

`x:float`, `y:float`, `z:float` (viewpoint position)

`pitch:float`, `yaw:float`, `roll:float` (camera attitude)

Expected response:

Image(2) (can be read as a **MemoryBlock(7)**)

Description:

Tell the server that the viewpoint is at position (`x`,`y`,`z`) and attitude (`pitch`,`yaw`,`roll`) and receive an image in response. This is equivalent to **SetViewpointByAngle** followed by **GetImage**.

Rationale:

Optimisation to reduce the amount of client-server communication during the heavily used case of set-viewpoint/read-okay/get-image/read-image.

GetViewpointByQuaternion (12)

Parameters:

`x:float`, `y:float`, `z:float` (viewpoint position)

`q0:float`, `q1:float`, `q2:float`, `q3:float` (quaternion describing camera attitude)

Expected response:

Image(2) (can be read as a **MemoryBlock(7)**)

Description:

Define the camera/viewpoint position and attitude using a position and quaternion and then receive the image from that viewpoint. The quaternion defines the rotation from PANGU/LDS frame into the camera frame. For a rotation of T radians about the axis defined by the unit vector (Dx, Dy, Dz) the q values are:

`q0` = cos(T/2)

`q1` = Dx*sin(T/2)

`q2` = Dy*sin(T/2)

`q3` = Dz*sin(T/2)

The identity quaternion [1,0,0,0] represents no rotation and causes the camera to point at the zenith. The quaternion [-1/v2,1/v2,0,0] rotates the camera to provide the view with no yaw, pitch or roll. Post-multiplying this quaternion by [cos(θ/2),0,-sin(θ/2),0], [cos(φ/2),sin(φ/2),0,0] and [cos(α/2),0,0,-sin(α/2)] in turn will rotate the camera so that it corresponds to yaw θ, pitch φ and roll α.

Rationale:

Optimisation to reduce the amount of client-server communication during the heavily used case of set-viewpoint/read-okay/get-image/read-image.

GetLidarPulseResult (13)

Parameters:

`x:float`, `y:float`, `z:float` (beam origin)

`dx:float`, `dy:float`, `dz:float` (beam direction)

Expected response:

LidarPulseResult(9)

Description:

Tell the server that the LIDAR emitter is at position (`x`,`y`,`z`) and that the beam direction is (`dx`,`dy`,`dz`). The range to the surface and the incidence angle are returned.

Rationale:

Allow fine-grained control of data returned by the LIDAR sensor.

GetLidarMeasurement (14)

Parameters:

`px:float`, `py:float`, `pz:float` (beam origin)
`q0:float`, `q1:float`, `q2:float`, `q3:float` (attitude of the LIDAR emitter)
`vx:float`, `vy:float`, `vz:float` (linear velocity of the LIDAR emitter)
`rx:float`, `ry:float`, `rz:float` (angular velocity of the LIDAR emitter)
`ax:float`, `ay:float`, `az:float` (linear acceleration of the LIDAR emitter)
`sx:float`, `sy:float`, `sz:float` (angular acceleration of the LIDAR emitter)
`jx:float`, `jy:float`, `jz:float` (linear jerk of the LIDAR emitter)
`tx:float`, `ty:float`, `tz:float` (angular jerk of the LIDAR emitter)

Expected response:

LidarMeasurement(10)

Description:

Tell the server that when scanning the centre of the frame the LIDAR emitter has position (`px`,`py`,`pz`) and attitude (`q0`,`q1`,`q2`,`q3`). The linear velocity of the craft is (`vx`,`vy`,`vz`), the linear acceleration is (`ax`,`ay`,`az`) and the linear jerk is (`jx`,`jy`,`jz`). Angular velocity is defined by the vector (`rx`,`ry`,`rz`), angular acceleration by the vector (`sx`,`sy`,`sz`) and angular jerk by the vector (`tx`,`ty`,`tz`). The server will compute the position and attitude of the craft using these parameters and various equations of motion for each beam generated by the LIDAR emitter. The results are packaged and returned as a **LidarMeasurement**.

Rationale:

Allow LIDAR response images to be generated taking the linear and angular motion of the LIDAR emitter/detector into account.

GetRadarResponse (15)

Parameters:

`flags:ulong`

`n:ulong` (number of samples to take)

`nr:ulong` (number of histogram bins for range integration)

`ns:ulong` (number of histogram bins for speed integration)

`rbs:float` (size of range histogram bins for fixed-sized bins)

`sbs:float` (size of speed histogram bins for fixed-sized bins)

`rcentre:float` (central range for centred range histograms)

`scentre:float` (central speed for centred speed histograms)

`ox:float`, `oy:float`, `oz:float` (origin of the RADAR beam)

`vx:float`, `vy:float`, `vz:float` (linear velocity of the RADAR emitter)

`q0:float`, `q1:float`, `q1:float`, `q3:float` (attitude quaternion of the emitter)

`bw:float` (full beam width in degrees)

{ 13 `ulongs` all set to zero (reserved for future expansion) }

Expected response:

RadarResponse(11)

Description:

Request a RADAR response from the server based on an emitter at position (`ox`,`oy`,`oz`) moving with linear velocity (`vx`,`vy`,`vz`) and with attitude quaternion (`q0`,`q1`,`q2`,`q3`). The position and velocity must be specified in the PANGU coordinate frame using the same units (*e.g.* metres). The attitude quaternion defines the rotation from the PANGU frame to the RADAR emitter frame with the beam axis along the z-axis of the RADAR emitter frame (see Section 2.2). The full angular width of the beam is `bw` measured in degrees. The response is constructed by integrating the response from `n` point samples over the beam foot print using a 2D histogram with `nr` by `ns` bins. Bits in the `flags` field control the way in which integration is performed as defined by the following table:

Bit	Meaning
0	if set then zero align the left edge of the range histogram
1	if set then centre align the range histograms on `rcentre`
2	if set then round the range histogram width up to an integer power of 10
3	if set then use range histogram bins of size `rbs`
4	if set then zero align the left edge of the speed histogram
5	if set then centre align the speed histograms on `scentre`
6	if set then round the speed histogram width up to an integer power of 10
7	if set then use speed histogram bins of size `sbs`
8	if set then surface slope effects are ignored
9	if set then each sample is worth 1 not 1/samples_used

All other bits are unused in the present version and must be zero. By default the histogram width and bin sizes are optimised to cover the extremes of ranges and speeds of the individual samples. If bit 3 is set then the `rbs` parameter defines the size of the range bins; if bit 7 is set then `sbs` defines the size of the speed bins. If these bits are clear then `rbs` and/or `sbs` will be ignored. With a zero-aligned histogram the start of the histogram is always 0. With a centre aligned histogram the range of the middle of the histogram will be `rcentre` and the middle of a speed histogram will be `scentre`. If bit 1 is not set then `rcentre` will be ignored; likewise if bit 5 is clear then `scentre` will be ignored. Normally the strength of each sample is scaled by the cosine of the surface slope relative to the sample direction and by the reciprocal of the number of samples which hit the surface. These effects can be disabled by setting bits 8 and 9 respectively of `flags`. The results are returned as a **RadarResponse**.

Rationale:

To support RADAR altimeter modelling.

SetViewpointByAngle (256)

Parameters:

`x:float`, `y:float`, `z:float` (viewpoint position)

`pitch:float`, `yaw:float`, `roll:float` (camera attitude)

Expected response:

Okay(0)

Description:

Define the camera/viewpoint position and attitude using craft-view coordinates. Use before a **GetImage** or **LookupPoints** request to define the camera position and attitude. Yaw is applied before pitch which is applied before roll.

Rationale:

Rotation angles are easier than quaternions to visualise.

SetViewpointByQuaternion (257)

Parameters:

`x:float`, `y:float`, `z:float` (viewpoint position)

`q0:float`, `q1:float`, `q2:float`, `q3:float` (quaternion describing camera attitude)

Expected response:

Okay(0)

Description:

Define the camera/viewpoint position and attitude using a position and quaternion. The quaternion defines the rotation from PANGU/LDS frame into the camera frame. For a rotation of T radians about the axis defined by the unit vector (Dx, Dy, Dz) the q values are:

`q0` = cos(T/2)

`q1` = Dx*sin(T/2)

`q2` = Dy*sin(T/2)

`q3` = Dz*sin(T/2)

The identity quaternion [1,0,0,0] represents no rotation and causes the camera to point at the zenith. The quaternion [-1/v2,1/v2,0,0] rotates the camera to provide the view with no yaw, pitch or roll. Post-multiplying this quaternion by [cos(θ/2),0,-sin(θ/2),0], [cos(φ/2),sin(φ/2),0,0] and [cos(α/2),0,0,-sin(α/2)] in turn will rotate the camera so that it corresponds to yaw θ, pitch φ and roll α.

Rationale:

NPAL/VBNAT require a quaternion interface to PANGU.

SetAmbientLight (258)

Parameters:

`r:float`, (normal range [0,1])

`g:float`, (normal range [0,1])

`b:float`, (normal range [0,1])

Expected response:

Okay(0)

Description:

Set the ambient light level for the scene. Every part of the model will receive this amount of light irrespective of its orientation. Normally the `r`, `g` and `b` components are identical. Increasing the ambient light will reduce the contrast between shadowed and non-shadowed areas.

Rationale:

Clients may wish to compare images with varying levels of ambient light.

SetSunColour (259)

Parameters:

`r:float`, (normal range [0,1])

`g:float`, (normal range [0,1])

`b:float`, (normal range [0,1])

Expected response:

Okay(0)

Description:

Set the Sun intensity for the scene. A value of (1,1,1) represents unit Solar radiance while (0,0,0) represents no radiance.

Rationale:

Clients may wish to compare images with varying levels of Sun intensity.

SetSkyType (260)

Parameters:

`type:ulong`

Expected response:

Okay(0)

Description:

Define the way the sky background is rendered. Possible values of `type` are:

0: no sky (black)

1: painted sky (sphere-mapped textured sky)

2: raw sky (single-pixel stars of varying intensity)

3: red sky (maximum red, no green, no blue)

4: green sky (no red, maximum green, no blue)

5: blue sky (no red, no green, maximum blue)

Rationale:

Starry backgrounds may confuse image processing algorithms so it is necessary to be able to enable or disable a realistic sky. To check horizon-sensitive algorithms it will be useful to ensure that the sky is rendered in a known uniform colour that is not used to render any part of the model (to support colour-separation/overlay). Still images for human viewing may require a painted sky where the lack of roll support is not an issue. A raw sky implements an unprocessed star field in which every star is a point source and occupies at most one pixel.

SetFieldOfView (261)

Parameters:

`fov:float`

Expected response:

Okay(0)

Description:

Set horizontal angular width of the field of view measured in degrees.

Rationale:

The field of view is an important parameter which affects the images produced. Large angles may cause significant distortion. Clients may wish to compare images from the same viewpoint with different camera field-of-view angles.

SetAspectRatio (262)

Parameters:

`ratio:float`

Expected response:

Okay(0)

Description:

Set the ration between the horizontal angular size of a pixel and the vertical angular size of a pixel. A ratio of 1 ensures that pixels are square,

Rationale:

Non-square pixels may be of interest to the client.

SetBoulderView (263)

Parameters:

`type:ulong`
`texture:boolean`

Expected response:

Okay(0)

Description:

Define the way the boulders are rendered. Possible values of `type` are:

0: no boulders (but their shadows may still be visible)
1: flat-shaded boulders (default)
2: smooth-shaded boulders

If `texture` is true then boulders will be textured (if possible).

Rationale:

Boulders may affect image processing algorithms. They also have an impact on rendering performance.

SetSurfaceView (264)

Parameters:

`type:ulong`
`texture:boolean`
`detail:boolean`

Expected response:

Okay(0)

Description:

Define the way the surface is rendered. Possible values of `type` are:

1: flat-shaded surface
2: smooth-shaded surface (default)

If `texture` is true then the surface will be textured (if possible). If `detail` is true then surface detail texturing will be used (if possible). Note that if `detail` is true then the `texture` setting is ignored.

Rationale:

Texturing may affect image processing algorithms. The use of texture and surface detail texture causes images to be darker than without their use.

SetLidarParameters (265)

Parameters:

`fx:float`, `fy:float` (angular field of view)
`nx:ulong`, `ny:ulong` (number of pixels per scan)
`tx:float`, `ty: float` (pixel and line scan times)
`n:ulong`, `m:ulong` (super-sampling factors)
`type:ulong` (scan type)
`flags:ulong` (flags controlling the scan results)
`azi:float`, `ele:float` (azimuth and elevation offset of the centre of the scan)
`theta:float` (beam half-angle width)
`wx:float`, `wy:float` (angular size of each detector pixel)
`faz:float` (sinusoidal scan frequency for type 3 and 4 scans)
{16 ulongs all set to zero: reserved for future expansion}

Expected response:

Okay(0)

Description:

Tell the server about the LIDAR emitter/receiver properties. These are used by **GetLidarMeasurement** to control the LIDAR beam generation and beam/surface interaction. The horizontal and vertical field-of-view angles are defined by (`fx`,`fy`) measured in degrees. The number of horizontal and vertical pixels in the scan are defined by (`nx`,`ny`) while the pixel-scan and line-scan times are defined by (`tx`,`ty`). The `tx` value defines the time between the emission of the LIDAR beam for one pixel on a line and the emission of the next pixel on the same line. The `ty` value defines the time T_{row} between the emission of the LIDAR beam for the first pixel of one line and the emission for the first pixel on the next line. The dead time between lines is `ty-nx*tx`. Each pixel may be subsampled over an area of (`n`,`m`) subpixels: the craft position and attitude is fixed for each pixel during subsampling. The `type` parameter is the scan type:

0 : TV-scan (left-to-right, top-to-bottom)
1 : LiGNC project zig-zag scan (azimuth mirror before elevation)
2 : LiGNC project zig-zag scan (elevation mirror before azimuth)
3 : LiILT project 1D sinusoidal scan (azimuth mirror before elevation)
4 : LiILT project 1D sinusoidal scan (elevation mirror before azimuth)

The central azimuth and elevation of the type 1–4 scans is given by the (`azi`,`ele`) values while the half-angle of the beam width is given by `theta` and are measured in degrees.

For type 3 and 4 scans `nx=n=1` with `ny` holding the number of pulse groups and m the number of channels to scan for each pulse group. The `wx` value will be ignored while the `ty` value holds the Tscan parameter (total scan time) rather than the T_{row} parameter used in scan types 1 and 2. The frequency of the sinusoidal scan in azimuth is passed via the `faz` parameter.

The `flags` bit vector determines the format of the results and the direction of type 3 and 4 scans:

bit 0: return range/slope results if set
bit 1: return azimuth/elevation values if set
bit 2: return corner cube range/slope values if set
bit 3: return time of pulse emission values if set
bit 16: if set the type 3 and 4 azimuth scan starts with decreasing azimuth
bit 17: if set the type 3 and 4 elevation scan starts with decreasing elevation
bit 18: if set the type 3 and 4 time decreases along the trace

All other bits must be set to zero. Bits 16–18 are only used by the type 3 and 4 scans. They control the direction of the azimuth and elevation of the scan. Normally bit 16 is zero. if bit 17 is 1 then the scan will be from the bottom of the frame to the top of the frame and the results will be returned in that order. As an alternative, bit 18 can be set instead of bit 17: this will cause the scan results to be returned in top-to-bottom order but ht epulse times will be working backwards in time. This allows the direction of the elevation scan to alternate top-to-bottom and bottom-to-top while the results are always returned in top-to-bottom order to simplify processing by the remote client.

The `wx` and `wy` values specify the angular size of a detector pixel in degrees.

Rationale:

Clients may wish to vary the properties of the LIDAR emitter during a simulation run but not every time that they request a LIDAR image.

SetCornerCubes (266)

Parameters:

`n:ulong` (number of corner cubes)
`f:ulong` (reserved: must be zero)
`s:ulong` (size of the rest of the message)
`px0:float`, `py0:float`, `pz0:float` (position of the first corner cube)
`nx0:float`, `ny0:float`, `nz0:float` (surface normal of the first corner cube)
`r0:float` (effective radius of the first corner cube
`px1:float`, `py1:float`, `pz1:float` (position of the second corner cube)
…

Expected response:

Okay(0)

Description:

This message is used to tell the server the details of any number of corner cube reflectors for all future GetLidarMeasurement scans. The parameter `n` defines the number of corner cubes and is followed a reserved field `f` which **must** be zero. The `s` field defines the number of bytes in the rest of the message which contain `n` blocks of seven floating-point values, one for each corner cube. The (px_i,py_i,pz_i) values define the position of the ith corner cube, the (nx_i,ny_i,nz_i) values define its normal and the ri values define its effective radius. During a LIDAR scan in which corner cubes are used the server will test each corner cube to see if any part of the LIDAR beam falls on it based on the assumption that each cube occupies a spherical volume. Corner cubes which face away from the beam as determined by their normal direction are ignored and the range and slope of the closest corner cube in the beam will be used.

Rationale:

Corner cube reflectors are used in rendezvous missions to enhance the LIDAR signal response. The server will use volume/volume intersection tests (or a close approximation) instead of ray/volume tests to ensure that corner cubes are detected even if their angular size is smaller than the angular resolution of the detector.

SetCornerCubeAttitude (267)

Parameters:

`q0:float`, `q1:float`, `q2:float`, `q3:float` (attitude of the corner cube lattice)
`rx:float`, `ry:float`, `rz:float` (angular velocity of the corner cube lattice)
`ax:float`, `ay:float`, `az:float` (angular acceleration of the corner cube lattice)
`jx:float`, `jy:float`, `jz:float` (angular jerk of the corner cube lattice)

Expected response:

Okay(0)

Description:

Tell the server that when scanning the centre of the frame the corner cube lattice has attitude (`q0`,`q1`,`q2`,`q3`). The angular velocity of the lattice is (`rx`,`ry`,`rz`), the angular acceleration is (`ax`,`ay`,`az`) and the angular jerk is (`jx`,`jy`,`jz`). The server will compute the position of all corner cubes using these parameters and equations of motion for all future uses of **GetLidarMeasurement**. The corner cube lattice has zero linear motion so all rotations are around the origin.

Rationale:

Allow rotating rendezvous canisters to be modelled.

7.3.2.3 Compatability with protocol version 0.00

Clients using protocol version 1.00 can replicate the version 0.00 protocol by starting with the **SetViewpointByAngle** message followed by a **GetImage** request. Most clients are expected to use either **SetViewpointByAngle** or

SetViewpointByQuaternion to position the camera and then issue a **GetImage** request. This may be followed by a GetElevation and a LookupPoint or LookupPoints message. The other messages are provided for flexibility.

7.3.2.4 TCP and UDP Sockets

Sockets can be implemented to operate with TCP or UDP. TCP provides a connection-oriented reliable link, where any datagrams that arrive in error or go missing are detected and retransmitted. UDP provides an unreliable datagram delivery service. Packets that go missing, arrive out of order or arrive in error are not corrected. TCP provides a safer mechanism for connecting to PANGU. If the underlying communication medium is inherently reliable then UDP will provide a slightly faster data delivery service although this speed improvement may be obliterated by the overhead required to manage packets within PANGU. Unfortunately neither PANGU 1.50 nor PANGU 2.00 support UDP.

7.3.2.5 The pangu_protocol_test program

A simple test program has been provided to allow users and developers to check and examine the facilities of protocol version 1.00. This program is called **pangu_protocol_test** and is distributed in source code form in the `client` directory on the PANGU installation CD. The executable program is installed in the same directory as the **viewer** and the other PANGU executables and will perform the following tasks (the **viewer** must be running in server mode first) saving each image in the current directory:

- position the camera, fly towards the model, change the pitch, change the roll, change the yaw.
- test different field-of-view angles, change the sky rendering mode, change the Sun colour, change the surface colour, get the camera altitude relative to the model, perform a simple 3D position lookup and several back projections.
- compute the camera altitude relative to the model for several camera positions.
- perform several back-projection lookups.
- perform several 3D position lookups.

Note that the results from this test will vary according to the size and shape of the model used.

This page has been left blank intentionally.

8. PANGU VIEWER AND TOOLS INI FILES

8.1 ADVANCED INI FILE USAGE FOR THE VIEWER AND ASSOCIATED TOOLS

The astute user will notice when each tool is used there are several messages about INI files. For example, when running the viewer under Linux we might see:

```
Unable to open INI file /usr/local/lib/pangu/pangu.ini
Processing section [_linux] from /home/mnd/.pangu/pangu.ini
Processing section [_defaults] from /home/mnd/.pangu/pangu.ini
Processing section [viewer] from /home/mnd/.pangu/pangu.ini
Unable to open INI file pangu.ini
Unable to open INI file /usr/local/lib/pangu/viewer.ini
Unable to open INI file /home/mnd/.pangu/viewer.ini
Unable to open INI file viewer.ini
```

This provides a few clues about the way that the viewer and tools process INI files.

8.1.1 INI file handling under Windows

These files are processed in the following order for a tool called FOO.exe stored in F:\PANGU\BIN\WIN32\:

```
F:\PANGU\BIN\WIN32\pangu.ini
C:\Program Files\PANGU\pangu.ini
pangu.ini (in the current directory)
F:\PANGU\BIN\WIN32\FOO.ini
C:\Program Files\PANGU\FOO.ini
FOO.ini
```

For each INI file found the [_win32] section is read, then the [_defaults] and then the [FOO] section.

8.1.2 INI file handling under Linux

These files are processed in the following order for a tool called FOO:

```
/usr/local/lib/pangu/pangu.ini
~/.pangu/pangu.ini
./pangu.ini
/usr/local/lib/pangu/FOO.ini
~/.pangu/FOO.ini
./FOO.ini
```

For each INI file found the [_linux] section is read, then the [_defaults] and then the [FOO] section.

8.1.3 Summary

In the previous sections we highlighted the fact that the INI file sections processed by the **viewer** and tools are based on the name of the tool itself. This means that a user could make several copies of the **viewer** application giving them names such as **viewer_model1**, **viewer_model2** and **viewer_modelbig**. By defining INI sections [viewer_model1], [viewer_model2] and [viewer_modelbig] in `pangu.ini` the user can have different viewer settings for each copy. Alternatively (or additionally) the user could create the INI files `viewer_model1.ini`, `viewer_model2.ini` and `viewer_modelbig.ini`.

8.2 ISSUES FOR WINDOWS 95 AND WINDOWS 98 USERS

Some older versions of Windows may use short filenames. For example, the **mkshadows** tool might be invoked using a strange name such as `MKSHAD~1.EXE`. If this occurs then the INI file processing will not work as intended. Instead of looking for the section [mkshadows] the program will look for [MKSHAD~1]. You will be able to see this by looking in the `PANGU.log` file.

There are two solutions for this problem: either rename tools which have long names and update the `pangu.ini` sections accordingly or just update `pangu.ini` so that the [mkshadows] section, for example, is called [MKSHAD~1] instead. The latter solution has been adopted by PANGU 2.00: the default `pangu.ini` files contain sections with both long and short names.

8.3 ISSUES FOR UNIX USERS

The section names in `pangu.ini` files are not case-sensitive so if the viewer is available under the names `Viewer` and `viewer`, both will use the section [viewer] for their settings.

8.4 FORMAT OF PANGU VIEWER INI FILES

8.4.1 General INI file format

PANGU INI files consists of zero or more INI lines divided into zero or more sections. Each section begins with a "[" character in the first column followed by the section name (leading and trailing spaces are ignored) and terminated by a "]" character. Section names are *not* case sensitive and duplicate section names are not allowed. The [and] must be on the same text line and only spaces or comments may appear after the closing "]".

Comments begin with either "`;`", "#" or "`//`" and continue for the rest of the text line.

Each section is comprised of zero or more entities. Normally each text line of the file represents a single entity but if the last character of a text line, after removing comments and trailing spaces, is a "\" character than the following text line is part of the entity. Spaces may appear before and/or after an entity.

An entity consists of a keyword and a value. Keywords are any sequence of characters excluding spaces. Empty lines are considered to be null entities having no keyword and no value.

String values are enclosed in '' or "" marks and may contain spaces. To include single-quote characters in strings enclosed in '' you must precede them with a backslash character "\"; similarly to include double-quote characters in strings enclosed in "" you must precede them with a backslash. String values may not be continued across multiple text lines: the opening and closing quote character (which is not part of the string value) must appear on the same text line. That is, you may not use the "\" character to split the value across multiple text lines.

8.4.2 PANGU Viewer INI file format

The format of INI files used by the viewer and associated tools is that defined in Section 8.4.1. However, some section names are reserved for use by this and future versions of PANGU. The names of all reserved sections will begin with the "_" character and users may not define their own sections preceded by this character.

`[_defaults]`

This section contains default settings which will be used by all tools which process these INI files. A common use is to specify the **model** setting here since several tools would normally use the same model.

`[_linux]`

Settings specific to Linux platforms can be defined here. The **log_errors** setting is usually defined here.

`[_win32]`

Settings specific to Windows platforms can be defined here. The **log_errors** setting is usually defined here.

8.5 A SAMPLE PANGU.INI

Below is a sample `pangu.ini` file split into several parts for readability.

8.5.1 Start of the file: magic marker

The INI file normally begins with the line "`#PANGU_INI`". This line is ignored (treated as a comment) by current versions of PANGU but future versions may require it to be present.

```
#PANGU_INI
```

8.5.2 Defaults and system-specific settings

As described in Section 8.4.2, the [_win32] and [_linux] sections provide platform-specific initialisation. Here all error and information messages are sent to `PANGU.log` under Windows and to standard output under Linux.

```
; Win32-specific settings.
[_win32]
        log_errors                      "PANGU.log"

; Linux-specific settings.
[_linux]
        log_errors                      -
```

8.5.3 Global defaults

Settings used by all tools can be specified in the [_defaults] section. Although most of the settings shown below pertain only to the viewer, listing them in the [_defaults] section instead of the [viewer] section of the INI file avoids accidental duplication. A common yet hard-to-detect error is to change a setting in the [_defaults] section (such as the model name) and find that there is no effect on the **viewer** because the setting is overridden in the [viewer] section.

```
; Defaults: these settings will be used as defaults by all programs although
; almost all of them are only used by the viewer. The settings are ordered
; so that the most used settings ought to be listed near the top.
[_defaults]
        ; Default model and shadowmap used by all tools. Boulders are
        ; always used (for rendering, shadow casting etc).
        model                           sample.pan
        shadowmap                       sample.smap
        do_boulders                     true

        ; Start viewer in model-view mode (use fixed_camera settings).
        view_mode                       model ; craft

        ; Define the model-view camera position and attitude. The
        ; fixed camera looks at the origin (0, 0, 0), distance 600m,
        ; azimuth 180, altitude 90. Note that the azimuth angle is the
        ; angle of the camera position relative to the y-axis of the
        ; model when viewed from the target hence 180 means that the
        ; camera points due north (NPAL notation).
        fixed_camera                    0 0 0       600 180 90

        ; Define the position and attitude of the craft-view camera.
        ; This free camera has been set to provide exactly the same
        ; view as the fixed_camera definition above. Note that the
        ; pangu_client application uses the free_camera setting.
        free_camera                     0 0 600     0 -90 0

        ; By default programs don't run in full-screen mode.
        full_screen                     false

        ; We could specify a default window size for all applications but
        ; then imgview wouldn't choose a window size to fit the image being
        ; displayed.
        ; window_size                   512 512

        ; The viewer will use a window large enough to render images
        ; of 512x512 pixels.
        viewer.window_size              512 512

        ; Default ROAM parameters.
        viewer.roam_limit               0.25   ; 1/4 pixel accuracy required
        viewer.roam_size_factor         0.0025 ; 1/400 triangle size factor
        viewer.roam_pool                -1     ; No pool

        ; Default memory manager parameters: the default is no cache.
        cache.size                      0
        cache.ceiling                   10000000 ; Approx 10 Mb
        cache.burst_limit               1000
        cache.name_prefix               "./.PANGU_"
        cache.name_suffix               ".cache"

        ; Use a reasonable sized cache for the viewer.
```

```
viewer.cache.size                52428800 ; 50 Mb
viewer.cache.ceiling             10485760 ; 10 Mb

; We don't normally run in server mode.
viewer.server_mode               false
viewer.remote_update             false ; Don’t mess with the user’s view.
server_port                      10363 ; Default port for TCP and UDP.
server_tcp_port                  10363 ; Overrides server_port.
server_udp_port                  10363 ; Overrides server_port.

; The poll_interval is the time (in milliseconds) that the viewer will
; wait between checks on the socket for new connections and for new
; messages following a period of inactivity. The lower this parameter
; the quicker the viewer will respond to incoming messages but may
; cause the server to become heavily loaded.
poll_interval                    10

; The poll_timeout is the time (in microseconds) that the viewer will
; wait for a new message from remote clients after an earlier message.
; If no message arrives by this time the viewer will respond to user
; interaction (mouse/keyboard) for another poll_interval milliseconds
; before trying to see if there is a new message. Increasing this
; value will cause the viewer to respond better to remote commands at
; the expense of poor user interaction.
poll_timeout                     2000000

; When saving files the first image will be called scr_000.ppm, the
; next scr_001.ppm etc. NEVER use \ as a directory separator because
; it won't work and may generate file names that contain invisible
; characters. NEVER use any other % characters in the format.
; Always use / as the directory separator.
save_format                      "scr_%03d.ppm" ; (uses C printf() format)

; When images are saved to disk they will be scaled by the integer
; factor defined below.
viewer.save_scale                1

; Help new users by display key actions on screen.
viewer.report_keys               true

; Don't display origin, landing site or model-view markers.
viewer.show_origin               false
viewer.show_landing_site         false
viewer.show_target               false

; Camera properties: 30 degree field of view with square pixels.
viewer.field_of_view             30
viewer.aspect_ratio              1.0

; It is convenient to shift the model vertically so that the 3D
; point (0, 0, 0) lies on the surface. Deep craters or high crater
; rims at the surface origin (0, 0) can cause the elevation at the
; surface origin to be several kilometres above or below zero.
viewer.recenter                  true

; Always use the raw star mode.
viewer.sky                       raw ; painted ; black ; white ; red ; green ; blue

; Star catalogue for raw star mode.
viewer.star_catalogue            "../stars/bright_stars.txt"

; Texture for painted sky mode: 1024x1024 forged night sky.
viewer.skymap                    "../textures/stars_1024x1024.ppm"

; No ambient light - harsh shadows.
ambient                          0.000 0.000 0.000

; Properties of the primary light source used by the viewer,
; mkshadows and mktexture.
sun.colour                       1.000 1.000 1.000
sun.position                     6.97e10 55  15 ; Mercury aphelion
;sun.position                    4.50e10 55  15 ; Mercury perihelion
sun.radius                       6.96265e8
sun.samples                      1
sun.visible                      false ; Currently ignored anyway
```

```
; Properties of the model surface including textures. Specular
; reflection should not be used for realistic images so the
; specular colour is set to black (0 0 0).
surface.diffuse.colour            1.000 1.000 1.000
surface.specular.colour           0.000 0.000 0.000
surface.specular.coefficient      128
surface.hapke.colour              1.000 1.000 1.000
surface.hapke.coefficient         128
surface.texture                   "../textures/surface_texture.ppm" 0.0016 true
surface.detail                    "../textures/surface_texture.ppm" 0.16 true

; Properties of boulders including texture. Specular reflection
; is disabled by setting the specular colour to black (0 0 0).
boulders.diffuse.colour           1.000 1.000 1.000
boulders.specular.colour          0.000 0.000 0.000
boulders.specular.coefficient     128
boulders.texture                  "../textures/boulder_texture.ppm" 0.32 true

; New parameter over-riding all other sky distances (for fog/dust clouds).
viewer.fog.sky_distance           1e18

; Global fog settings: coloured red to make it obvious!
viewer.global_fog.enable          false
viewer.global_fog.colour          1.000 0.000 0.000
viewer.global_fog.mode            exp ; exp2 ; linear
viewer.global_fog.linear_start    10
viewer.global_fog.linear_end      800
viewer.global_fog.density         0.001

; Local-planar fog properties: coloured green to make it obvious!
viewer.plane_fog.enable           false
viewer.plane_fog.colour           0.000 1.000 0.000
viewer.plane_fog.height           100
viewer.plane_fog.density          0.04

; Spherical fog properties: coloured blue to make it obvious!
viewer.sphere_fog.enable          false
viewer.sphere_fog.colour          0.000 0.000 1.000
viewer.sphere_fog.radius          50
viewer.sphere_fog.density         0.04
viewer.sphere_fog.origin          0 0 0

; Local-planar fog properties: coloured yellow to make it obvious!
viewer.general_fog.enable         false
viewer.general_fog.colour         1.000 1.000 0.000
viewer.general_fog.density        0.04
viewer.general_fog.origin         0 0 0
; Do NOT specify a fog model unless the specified file exists and
; is readable. Otherwise the viewer will fail to start up.
;viewer.general_fog.model         "general_fog_model.pan"

; Light attached to the camera.
viewer.camera_light               false
viewer.camera_light.position      500 500 -500
viewer.camera_light.colour        0.800 0.200 0.200
viewer.camera_light.half_angle    180
viewer.camera_light.exponent      80

; Surface coordinate origin marker: we use a compass image to paint
; over the decal making it visible. The decal covers an area of 3x3
; metres, is lifted 10cm above the model and is sampled using a grid
; of 31x31 points.
viewer.origin                     "../textures/compass.ppm" 3 0.10 31

; Same idea for the landing site marker. In this example we place
; the landing site over the origin (0, 0), make it slightly smaller
; than the origin mark (2x2 metres) and position it above the origin
; mark (20cm above the model). The sample grid is 21x21 points.
viewer.landing_site               "../textures/target.ppm"  0 0     2 0.20 21

; Model-view target box: we define a cube 2x2x2 metres.
viewer.target                     "../textures/target_box.ppm"      2 2 2
```

```
; Allowing the viewer to trigger textures automatically according
; to the height of the camera above the surface is not advisable
; so we disable triggers by using negative trigger altitudes.
viewer.trigger_detail             -1 ; Disabled
viewer.trigger_texture            -1 ; Disabled

; Unfortunately we have to worry about depth limits: objects that
; are closer to the camera than viewer.near_distance will not be
; rendered. Nor will objects further than viewer.far_distance from
; the camera. Graphics cards with 16 bit depth buffers often produce
; "jittery" images if:
;
;    viewer.far_distance/viewer.near distance > 1000
;
; With 24 bit depth buffers this ratio can be much higher.
;
; With depth optimisation disabled or with depth rescaling enabled
; the following two settings define the position of the near and far
; clipping planes. These values assume a 16-bit depth buffer.
viewer.near_distance              4
viewer.far_distance               4000

; If depth optimisation is enabled (and depth rescaling disabled)
; the viewer will select the best near/far distances for a group of
; image frames within the following limits. Users with 16 bit depth
; buffers must avoid setting the near limit too small as the near/far
; ratio will almost certain exceed the capabilities of their system.
; Users with 24-bit depth buffers could use a near limit of 0.1 and
; not suffer any obvious artifacts.
viewer.near_limit                 0.1
viewer.far_limit                  1e20

; More depth buffer parameters: to avoid moving the near or far plane
; during rendering (at the cost of an extra rendering pass) we allow
; the current near and far planes to lie within a zone on one side of
; the ideal position (model limits). If the current near or far plane
; move outside the zone then they are adjusted when depth optimisation
; is enabled (viewer.optimise_depth).
;
; If the closest point on the model is zmin then the near plane at
; znear must satisfy: zmin/(1+a) > znear > zmin/(1+b) where (a < b)
; and (a > 0). Here a = viewer.near_A and b = viewer.near_B:
;
; Likewise if the farthest point on the model is zfar then the far
; plane at zfar must satisfy: zmax*(1+a) < zfar < zmax*(1+b) where
; (a < b) and (a > 0). Here a = viewer.far_A and b = viewer.far_B:
viewer.near_A                     0.1
viewer.near_B                     2.0
viewer.far_A                      0.1
viewer.far_B                      2.0

; Depth rescaling should be avoided and should never be used when
; depth optimisation is on. In fact the 2.00(9) viewer and later will
; disable depth optimisation if depth rescaling is enabled.
viewer.optimise_depth             true
viewer.rescale_depth              false

; Objects whose angular size is smaller than angular_cutoff pixels
; will not be shown. The angular size is determined by examing the
; bounding sphere of the object. Use a size of zero to disable the
; object culling (no object is smaller than zero pixels).
viewer.angular_cutoff             4.00

; If the angular size of the measure of bi-resolution object is
; less than resolution_cutoff the low resolution version of the
; object will be rendered. Normally this means that the size of
; the largest triangle in a section of the model will never drop
; below resolution_cutoff pixels. Unfortunately in PANGU 1.50 and
; PANGU 2.00 each layer of a hierarchical model is a section so
; low-angle views of a model will have triangles in the distance
; that are much smaller than this limit due to perspective. Use
; zero to disable this resolution changing and note that values
; greater than 1 may cause visible jumps when zooming into or out
; of a model.
viewer.resolution_limit           1.00
```

```
; The viewer can be used in movie mode whereby a user-defined flight
; file is processed to determine the position and orientation of
; the camera along with various other rendering properties. This is
; disabled and we specify a (hopefully) non-existant file.
viewer.flight_mode              false
viewer.flight_file              "............."

; If the viewer is rendering a large model then it may not be able to
; process a flight file quickly enough for the user. The user can solve
; this problem by instructing the viewer not to render images for every
; camera position/attitude. The number of images to skip is defined
; to be 1 here which doubles the rate at which the flight file is
; processed (for every two camera positions only one will be rendered).
viewer.flight_speed             0

; Normally the images generated from a flight file will not be saved
; to disk unless the flight file contains explicit save or save_as
; commands. To save every image that is generated when processing a
; flight file set the following parameter to true. The names of the
; files will be created according to the current save format.
viewer.save_frames              false

; Ray-traced depth image support: we define the grid size over which
; the model is sampled and the maximum depth range possible.
viewer.raytrace_depth_grid      512 512
viewer.raytrace_depth_range     50.0

; ============================================================================
; The rest of this section contains less-used (ie. more advanced) settings
; ============================================================================

; We don't use expert mode so we have the control panel on the
; right hand site of the viewer image. Expert mode allows us to
; resize the window using the < and > keys and to swap between
; model and craft view mode by pressing V.
viewer.expert                   false

; On some platforms (currently only observed on a Win2000 machine)
; the viewer is given a low-precision depth buffer instead of the
; full precision buffer that might be available. Try changing this
; flag and checking the "Depth buffer" size that is displayed when
; the viewer quits or when "=" is pressed.
viewer.depth_buffer_hack        false

; This is an experimental setting which, if set to true, can give
; a 2x speed-up for rendering. HOWEVER, note that enabling this
; setting may cause boulder and/or textures to have the wrong colour.
viewer.use_vertex_arrays        false

; User interface: the angle_step defines the number of degrees to
; rotate when an arrow key is pressed in rotation mode. In the same
; way move_step defines the distance to move when an arrow key is
; pressed in movement mode.
viewer.angle_step               4
viewer.move_step                10

; Camera rotations with the mouse are more accurate than with the
; keyboard. A single pixel of mouse rotation is converted into
; angle_step*mouse_rotate_scale degrees of rotation.
viewer.mouse_rotate_scale       0.5

; By default all units are measured and displayed in metres by the
; viewer. To display distance units in feet instead set the output
; scale to 3.280839895 and the output units to "ft".
viewer.output_scale             1.0
viewer.output_units             "m"

; Planet surfaces use smooth shading to hide the mesh facets but
; boulders use flat shading to emphasise their rugged nature.
shade                           smooth ; flat
boulder_shade                   flat   ; smooth
```

```
; We always use the Lambert (diffuse) reflection model. Currently
; the only alternative is an approximation to Hapke: if that is
; used then each image will consist of 30% (0.3) of a Hapke response
; and 70% (1 - 0.3) of a Lambert respsonse.
viewer.reflection_model          lambert ; hapke
viewer.hapke_coefficient         0.3

; We disable shadowmap smoothing since it must not be used with
; PANGU version 1.50 or later. We never ignore shadowmap hash
; codes as this prevents model/shadowmap mismatches.
smooth_shadows                   false
viewer.ignore_hashcode           false

; If model recentering is disabled (viewer.recenter) then users
; can shift the model vertically by the following amount.
viewer.zoffset                   0.00

; We always render with filled polygons. The alternatives are wire
; for wire-frame mode and dot for point-cloud mode.
viewer.render                    filled ; wire ; dot

; The first false colour mode uses vertical elevation relative to
; the 3D origin scaled by 0.1 to define the contour colour to use.
; The contour map is a 1D texture. The scale factor of 0.1 can be
; changed to suit the altitude range of the particular model.
viewer.contour_map               "../textures/contour.ppm" 0.1

; The second false colour mode uses camera-model distances (after
; subtracting viewer.range_offset) scaled by 0.01 to define the
; contour colour to use. The contour map is a 1D texture. The scale
; factor of 0.01 and the range of 500m can be changed to suit the
; distance range of the particular model and camera position.
viewer.range_offset              500
viewer.range_map                 "../textures/contour.ppm" 0.01

; Blink comparison is not normally used but we can set the default
; delay to be 2 seconds. To disable blink comparison we set the blink
; image to the name of something that (hopefully) cannot exist.
blink_delay                      2000 ; imgview and viewer
viewer.blink_delay               2000 ; just viewer
; blink_image                    ".........."
; viewer.blink_image             ".........."

; The "automatic" animation mode of the viewer can be used as a
; simple demonstration of the speed of the viewer. Note that the
; trajectory used in this mode was designed for a 513x513 single
; layer model.
viewer.auto_animation            false

; Frame rate decay constant: the average frame rate is fps_gamma
; times the previous average plus (1-fps_gamma) times the current
; frame rate. Values of fps_gamma close to one hide small-scale
; fluctuations but require many frames before the average settles
; on the true decaying average. Values close to zero are strongly
; linked the the frame rate of the current frame but may vary
; dramatically from frame-to-frame.
; Same idea for polygon rate : poly_gamma decay constant.
fps_gamma                        0.85
poly_gamma                       0.85
```

8.5.4 Viewer settings

All the settings for the **viewer** have been defined above in the [_defaults] section. We don't need to override any settings so this section can be left empty. We could change the log file used to report errors so that viewer errors always go to `viewer.log` instead of `PANGU.log`:

```
; Viewer-specific settings.
[viewer]
        ; Any settings defined here will override those in the [_defaults] section.
        ; log_errors                    "viewer.log"
```

8.5.5 Shadow map generator settings (UNIX, WinNT, Win2000, WinXP)

The name "mkshadows" is too long to fit into the default 8.3 naming pattern used by Windows95, Windows 98 and WindowsMe so we first define the shadow map generator section for UNIX, WindowsNT, Windows2000 and WindowsXP users.

```
; Settings for the shadowmap generator for UNIX, WinNT, Win2k and WinXP.
[mkshadows]
        ; Specify the model name here to be absolutely certain that we
        ; generate shadows for the correct model (which may be different
        ; to the one we are currently viewing).
         model                          sample.pan

        ; To avoid accidentally overwriting an existing shadowmap we set
        ; the overwrite flag to false and choose a shadowmap name that is
        ; unlikely to be associated with the current model. Once the map
        ; has been created the user can rename it to 4_layers.smap etc.
        save_shadowmap                  output.smap
        overwrite                       false

        ; Shadow maps may take a long time to produce - if you want to
        ; place a limit on the length of time spent computing shadows then
        ; set the following time limit (in seconds) accordingly. We set a
        ; large time limit to ensure that the map is always completed.
        shadow_time_limit               36000000 ; More than a year!

        ; You can speed up shadowmap generation by disabling boulders by
        ; setting do_boulders to false. However, if you render the model
        ; with boulders they will appear to float even when they aren't.
        do_boulders                     true

        ; We could define a specific Sun position here.
        ; sun.position                  6.97e10 55  5 ; Mercury aphelion
        ; sun.radius                    6.96265e8

        ; For area light source support we define the number of times that
        ; we want to sample the light source and how to do it. Note that
        ; the default sun.sample_method is "point" which treats the Sun as
        ; a single point light source.
        ; sun.samples                   40
        ; sun.sample_method             rings ; point

        ; We can choose whether or not we want the progress meter here
        ; without affecting the setting for mktexture.
        show_progress                   true

        ; We could redirect output for mkshadows to a separate log file.
        ; log_errors                    mkshadows.log
```

8.5.6 Shadow map generator settings (Win95, Win98, WinMe)

In the previous section the settings for **mkshadows** were given for various platforms. For the platforms which use 8.3 DOS naming schemes we repeat the settings (without explanations) using the 8.3 name for **mkshadows** (`mkshad~1`):

```
; Settings for the shadowmap generator for Win95, Win98, WinMe.
[mkshad~1]
         model                          sample.pan
        save_shadowmap                  output.smap
        overwrite                       false
        shadow_time_limit               36000000 ; More than a year!
        do_boulders                     true
        ; sun.position                  6.97e10 55  5 ; Mercury aphelion
        ; sun.radius                    6.96265e8
        ; sun.samples                   40
        ; sun.sample_method             rings ; point
        show_progress                   true
        ; log_errors                    mkshadows.log
```

8.5.7 Texture map generator settings

Here are all the **mktexture** settings: the model and height resolution should not be changed although increasing the height resolution can be used to produce a rougher texture.

```
; Settings for the texture generator for UNIX, WinNT, Win2k and WinXP. For
; Win95, Win98 and WinMe see the [mktext~1] section below.
[mktexture]
        ; Define the DEM model used to cast shadows over and then image
        ; to produce the texture. The model could be created with surfacegen
        ; but ought to be as large as possible with only small craters.
        texture_model                   "../samples/texture_surface.ppm"

        ; The output of surfacegen will tell you the height resolution of
        ; the DEM (the number of metres per integer height step). This is
        ; the correct setting for the sample texture_surface.ppm DEM.
        height_resolution               0.0006111

        ; We don't want any ambient light for the texture: the ambient light
        ; component will be introduced by the viewer when rendering. We want
        ; maximum Sun illumination though and maximum surface albedo: we are
        ; generating a lightmap texture not an image of the surface.
        ambient                         0.000 0.000 0.000
        sun.colour                      1.000 1.000 1.000
        surface.diffuse_colour          1.000 1.000 1.000

        ; We can define the Sun position here if we wanted to. We need to
        ; define the sample count otherwise we may get an area light source
        ; when we only wanted a point source or vice versa.
        ;sun.position                   4.50e10 55  15 ; Mercury perihelion
        sun.radius                      6.96265e8
        sun.samples                     1

        ; Always use PPM image format for saving textures.
        texture_format                  ppm

        ; Name of the file to save the new texture in.
        save_texture                    "new_texture.ppm"

        ; When generating mipmaps we there is a choice of filters. For now
        ; just ignore this setting (read the user manual for more details).
        mipmap_filter                   average

        ; What are the dimensions of the texture to be generated? Must be
        ; an integral power of two greater than or equal to 64.
        texture_size                    512

        ; We want mktexture to quit as soon as it has saved the texture.
        quit_when_done                  true

        ; For safety we don't overwrite an existing texture file.
        overwrite                       false

        ; We can choose whether or not we want the progress meter here
        ; without affecting the setting for mkshadows.
        show_progress                   true

        ; We could redirect output for mktexture to a separate log file.
        ; log_errors                    mktexture.log

; Settings for the texture generator for Win95, Win98, WinMe. See the
; [mktexture] section above for descriptions.
[mktext~1]
        texture_model                   ../samples/texture_surface.ppm
        height_resolution               0.0006111
        ambient                         0.000 0.000 0.000
        sun.colour                      1.000 1.000 1.000
        surface.diffuse_colour          1.000 1.000 1.000
        ;sun.position                   4.50e10 55  15 ; Mercury perihelion
        sun.radius                      6.96265e8
        sun.samples                     1
        texture_format                  ppm
        save_texture                    "new_texture.ppm"
        mipmap_filter                   average
        texture_size                    512
        quit_when_done                  true
        overwrite                       false
        show_progress                   true
        ; log_errors                    mktexture.log
```

8.5.8 Texture map editor settings

The default settings here are fairly meaningless since the scale factors required depend heavily on the average intensity of the texture map being edited. The example here brightens textures by 20% (a scale factor of 1.2).

```
; Settings for the texture manipulator for UNIX, WinNT, Win2k and WinXP. For
; Win95, Win98 and WinMe see the [edtext~1] section below.
[edtexture]
        ; Specify the name of the texture to edit.
        load_texture                    "new_texture.ppm"

        ; Specify the name of the file in which the editted texture will
        ; be written.
        save_texture                    "ed_texture.ppm"

        ; For safety we don't overwrite an existing texture file.
        overwrite                       false

        ; The arithmetic operations that we want to apply: division and/or
        ; multiplication of each pixel value by the specified amount.
        ; Negative values mean that the operation WON'T be applied
        texture_divide                  -1
        texture_multiply                1.2         ; Brighten by 20%

        ; We can also apply gamma correction to the image (non-linear scale).
        texture_gamma                   -1

        ; We could redirect output for edtexture to a separate log file.
        ; log_errors                    edtexture.log

; Settings for the texture manipulator for Win95, Win98, WinMe. See the
; [edtexture] section above for details.
[edtext~1]
        load_texture                    "new_texture.ppm"
        save_texture                    "ed_texture.ppm"
        overwrite                       false
        texture_divide                  -1
        texture_multiply                1.2         ; Brighten by 20%
        texture_gamma                   -1
        ; log_errors                    edtexture.log
```

8.5.9 Shadow map merger settings

In this example we merge three separate maps into a single map.

```
; Settings for the shadowmap combiner for UNIX, WinNT, Win2k and WinXP. For
; Win95, Win98 and WinMe see the [merges~1] section below.
[mergeshadows]
        ; Specify the names of all the shadowmaps. The \ character is a
        ; line continuation character which allows us to put the name of
        ; each shadowmap on a single line. Use comments after the line
        ; continuation character to describe each map.
        load_shadowmaps \ ; This is a continued line
                                        u_55_15.smap      \ ; 55 azimuth, 15 altitude
                                        u_54.2_15.smap    \ ; 54.2 azimuth, 15 altitude
                                        u_55.8_15.smap    \ ; 55.8 azimuth, 15 altitude
                                        u_55_14.2.smap    \ ; 55 azimuth, 14.2 altitude
                                        u_55_15.8.smap      ; 55 azimuth, 15.8 altitude

        ; The combined maps will be saved in the following file. Choose
        ; a name that reflects the Sun parameters (eg azimuth and altitude
        ; angles, number of area light source samples and the angular width
        ; of the light source).
        save_shadowmap                  u_merged.smap

        ; For safety we don't overwrite an existing shadowmap file.
        overwrite                       false

        ; We could redirect output for mergeshadows to a separate log file.
        ; log_errors                    mergeshadows.log
```

```
; Settings for the shadowmap combinerfor Win95, Win98, WinMe. See the
; [mergeshadows] section above for details.
[merges~1]
        load_shadowmaps \ ; This is a continued line
                                        u_55_15.smap       \ ; 55 azimuth, 15 altitude
                                        u_54.2_15.smap     \ ; 54.2 azimuth, 15 altitude
                                        u_55.8_15.smap     \ ; 55.8 azimuth, 15 altitude
                                        u_55_14.2.smap     \ ; 55 azimuth, 14.2 altitude
                                        u_55_15.8.smap       ; 55 azimuth, 15.8 altitude
        save_shadowmap                  u_merged.smap
        overwrite                       false
        ; log_errors                    mergeshadows.log
```

8.5.10 Image viewer settings

This application has a short name so one section covers all platforms.

```
[imgview]
        ; This is the name of the image to load.
        main_image                      "sample_image.ppm"

        ; When in blink comparison mode we must specify the blink image
        ; and, ideally, the time between blinks (in milliseconds).
        ; blink_image                   blink_image.ppm
        blink_delay                     200

        ; We could use the full screen instead of just a window.
        ; full_screen                   true

        ; If we DO NOT specify the size of the imgview window either here
        ; or in the defaults then imgview will use the size of the image
        ; being displayed to define the size of the window.
        ; window_size                   512 512

        ; In case the user has defined a specific window size then we can
        ; request that the image is scaled to fill the window (keeping the
        ; aspect ration unchanged of course).
        ; scale_to_fit                  true

        ; We could redirect output for imgview to a separate log file.
        ; log_errors                    imgview.log
```

8.5.11 PANGU Object dumper settings

```
; PANGU object file dumper: settings for all platforms.
[pandump]
        ; First we have to specify the name of the model to examine.
        model                           4_layers.pan

        ; Specify the name of the file to write the information to.
        output_file                     pandump.txt

        ; We don't mind overwriting an existing "output_file" file.
        overwrite                       true

        ; We could redirect output for pandump to a separate log file.
        ; log_errors                    pandump.log
```

8.5.12 PANGU client settings

```
; Remote client application settings for UNIX, WinNT, Win2k and WinXP. For
; Win95, Win98 and WinMe settings see [pangu_~1] section below.
[pangu_client]
        ; In this example the remote server is on the same machine as us
        ; and we always use protocol version 1.00.
        pangu_client.server_name        127.0.0.1 ; localhost
        pangu_client.protocol           1

        ; We could define the server port here but we pick up the server_port
        ; default setting instead.
        ; pangu_client.server_port      10363

        ; The initial camera position can be specified here if necessary.
        ; This setting will override and free_camera setting.
        ; pangu_client.free_camera      0 0 600           180 -90 0
```

```
        ; To read flight file commands and request images from the server
        ; to display the results of those commands we specify a flight file.
        ; pangu_client.flight_file        "example.fli"

        ; If a flight file is processed the images are simply displayed. To
        ; save images to disk the following setting must be true.
        pangu_client.save_frames          false

        ; We can choose to run in full-screen mode if we wish. We don't.
        ; This setting will override and full_screen setting.
        pangu_client.full_screen          false

        ; Choose a sensible window size: overrides window_size.
        pangu_client.window_size          512 512

        ; The window size used by the client may not match the size of the
        ; images generated by the server. This option allows the client to
        ; resize images to fit the window while preserving the aspect ratio.
        pangu_client.scale_to_fit         true

        ; When saving files the first image will be called scr_000.ppm, the
        ; next scr_001.ppm etc. NEVER use \ as a directory separator because
        ; it won't work and may generate file names that contain invisible
        ; characters. NEVER use any other % characters in the format.
        ; Always use / as the directory separator.
        ; This setting overrides the save_format setting.
        pangu_client.save_format          "scr_%03d.ppm" ; (uses C printf() format)

        ; We could redirect output for pangu_client to a separate log file.
        ; log_errors                      pangu_client.log

; Remote client application settings for Win95, Win98, WinMe. For details
; of these settings see the [pangu_client] section above.
[pangu_~1]
        pangu_client.server_name          127.0.0.1 ; localhost
        pangu_client.protocol             1
        ; pangu_client.server_port        10363
        ; pangu_client.free_camera        0 0 600            180 -90 0
        ; pangu_client.flight_file        "example.fli"
        pangu_client.save_frames          false
        pangu_client.full_screen          false
        pangu_client.window_size          512 512
        pangu_client.scale_to_fit         true
        pangu_client.save_format          "scr_%03d.ppm" ; (uses C printf() format)
        ; log_errors                      pangu_client.log
```

8.5.13 PANGU to POV-Ray file converter settings

```
; PANGU object file to POVRAY script converter: settings for all platforms.
[pan2pov]
        ; Specify the model to be converted into a POV-Ray script.
        model                             sample.pan

        ; Specify the name of the POV-Ray script to produce.
        save_povray                       sample.inc

        ; We don't want to overwrite an existing "save_povray" file.
        overwrite                         false

        ; By default the viewer users smooth-shaded polygons so we generate
        ; a smooth-shaded POV-Ray model.
        povray_smooth                     true

        ; Our converter does not support boulders sensibly so disable them.
        do_boulders                       false

        ; We can specify the surface colour for the generated objects. The
        ; lighting settings are generated by the PANGU viewer when the P key
        ; is pressed.
        surface.diffuse.colour            0.900 0.900 0.900

        ; We could redirect output for pan2pov to a separate log file.
        ; log_errors                      pan2pov.log
```

This page has been left blank intentionally.

9. USING POV-RAY TO VISUALISE MODELS

It is not necessary to use POV-Ray to render surface models as a dedicated viewer has been provided which will render the surface models faster. However, for completeness, a legacy application, **povgen**, has been provided which uses POV-Ray to render single resolution surface models.

9.1 INSTALLING POV-RAY

POV-Ray must be installed on your PC to enable the rendering of surface models using PovGen. POV-Ray is a freeware ray-tracing tool and is available for download at http://www.povray.org.

To complete the installation, after POV-Ray and PANGU have been installed, copy the "boulder0.inc", "boulder1.inc" and "boulder2.inc" files from the PANGU "PovrayInclude" directory and put them in the main POV-Ray "include" directory (e.g. "C:\Program Files\POV-Ray for Windows\include"). This completes the installation.

9.2 POVGEN COMMAND REFERENCE

This program is used to create a POV-Ray scene file and then ask POV-Ray to render it.

povgen [-p *file*] [–d *file*] [-h | -i]

Options available are:

- -p, -P Create a [P]OV-Ray scene file and render it. Requires **povgen** parameter file (Section 9.3).
- -d, -D Create a [D]efault **povgen** parameter file. The *file* parameter is used as a prefix for the parameter file.
- -h, -H Display help text and exits **povgen**.
- -i, -I Same as -h/-H

9.3 POVGEN PARAMETERS (POVPARAMS.TXT)

This file is used to define how the surface model is visualised in POV-Ray. These parameters are used to define the POV-Ray script file, which renders the image. The single resolution, DEM surface model and output image directory must be defined in a hard-coded path. The sun is approximated by an array of point light sources whose distance and diameter can be defined. This is relevant if penumbra is to be modelled. The larger the number of point light sources defined then the better quality any visible penumbra will be but the longer it will take an image to be rendered.

The spacecraft position should be defined in relation to the centre of the surface where the y position is defined as a distance vertically upwards from the surface. The roll parameter is currently unused.

If boulders are to be added then the boulder filename must be specified.

This page has been left blank intentionally.

10. COMPILING PANGU FROM C++ SOURCES

The source code for the PANGU tools is not publicly available. However, for those who have a license for the source code, this section describes the directory structure used and provides the necessary compilation instructions. Note that although separate source distributions are provided for Windows and UNIX the source code is identical. The separate distributions allow us to use platform-specific compilation systems (Microsoft Visual Studio under Windows and Make/GNU C++ under UNIX) and allow files to be stored using the native format of the build platform. Since the surface generator tools and the viewer tools are being developed as part of separate projects the sources for these tool sets are distributed separately.

10.1 BUILDING THE SURFACE GENERATOR

The directory structure for the surface generator sources and compilation environment is shown in Figure 10-1.

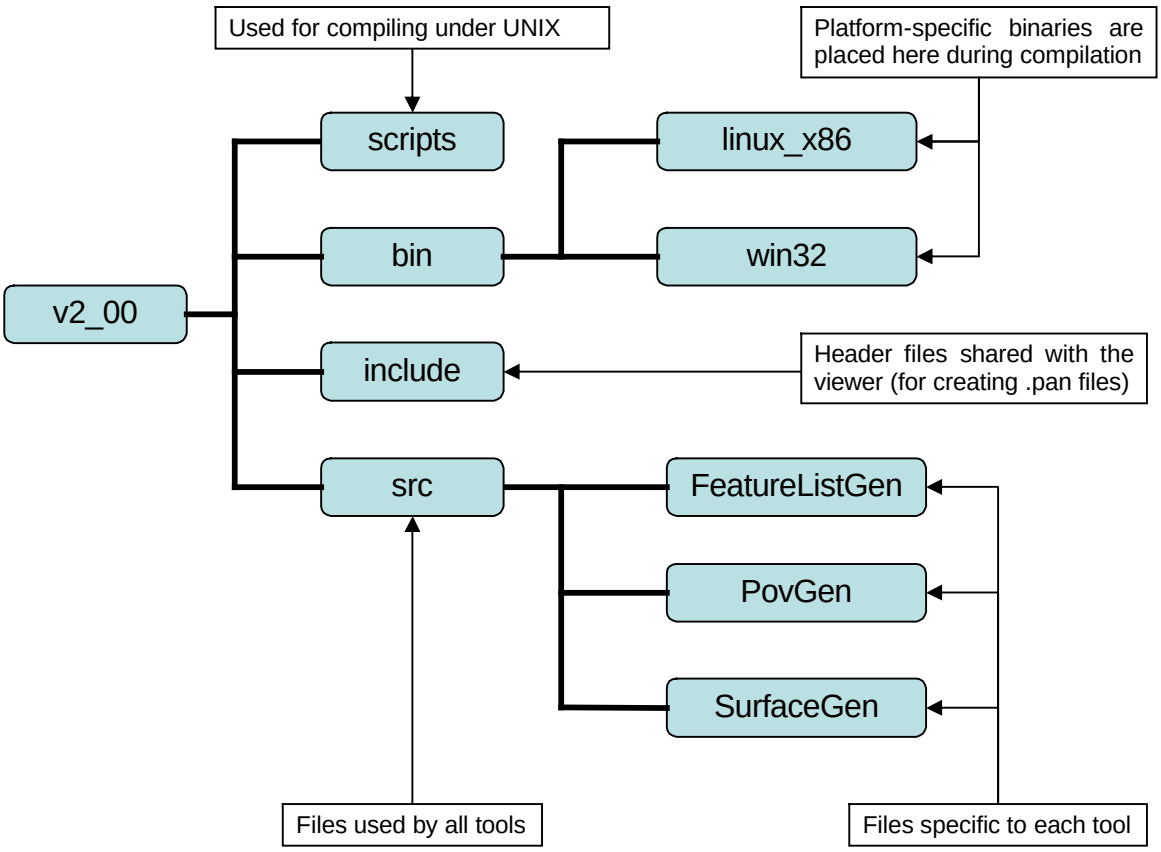

Figure 10-1: surface generator source directory structure

10.1.1 Windows developers

Windows developers are expected to compile the surface generator and related tools using Microsoft Visual C++ 6.0 and have a basic working knowledge of that application. Other C++ compilers ought to work but their use is not supported by the PANGU development team.

The workspace for the surface generator is stored in `src\pangu.dsw` and contains five projects:

- _BuildEverything: a dummy project used to build all the others in one step.
- Pangu: a library of common objects used by the other three projects.
- SurfaceGen: the tool for creating surface DEMs and PANGU object files
- FeatureListGen: the tool for creating lists of craters and boulders
- PovGen: a legacy tool which uses POV-Ray to visualise PANGU models

The steps for building the three executables **surfacegen**, **featurelistgen** and **povgen** are:

- Open `src\pangu.dsw` using Microsoft Visual C++
- Set the active configuration to "_BuildEverything - Win32 Release" (Build → Set Active Configuration)
- Remove any temporary files (Build → Clean)
- Compile everything (Build → Build or press F7)

The executables will be placed in the `bin\win32` directory.

10.1.2 UNIX developers

UNIX developers are expected to compile the surface generator and related tools using GNU C++ 2.96 on RedHat 7.x. The C++ standard template library will be needed so install the `libstdc++-devel` package if necessary. The build process is managed by a set of Makefiles with the top-level Makefile in the `src` directory. Each tool has its own set of Makefiles but these are invoked from the top-level when necessary.

The commands for building the three executables **surfacegen**, **featurelistgen** and **povgen** are:

```
cd src
make clean
make install
```

The executables will be placed in the `bin/linux_x86` directory.

10.2 BUILDING THE VIEWER AND RELATED TOOLS

The directory structure for the **viewer** sources and compilation environment is shown in Figure 10-2.

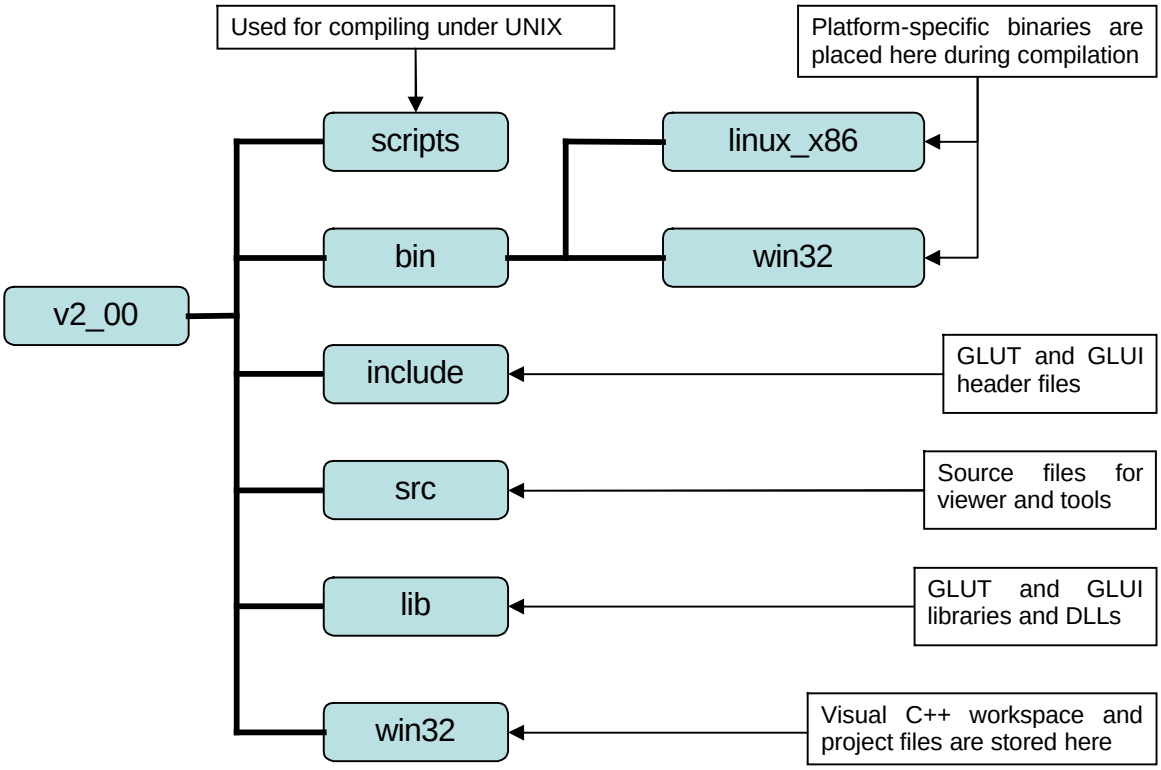

Figure 10-2: viewer and tools source directory structure

Note that the sources for all the tools are stored in the `src` directory. The application name is used as the basis of the top-level source file e.g. `mkshadows.cpp` for **mkshadows**.

10.2.1 Windows developers

Windows developers are expected to compile the **viewer** and related tools using Microsoft Visual C++ 6.0 and have a basic working knowledge of that application. Other C++ compilers ought to work but their use is not supported by the PANGU development team.

The workspace for the **viewer** and tools is stored in `win32\win32.dsw` and contains eleven projects. Ten of the projects correspond to the **viewer** or tool of the same name. The remaining project is called _BuildEverything: this is a dummy project which is used to build all the other projects in one step.

The steps for building the **viewer** and tools executables are similar to those for the surface generator (Section 10.1.1). Note that the warnings about the `String` macro being redefined in `choice_optparser.h` can be ignored along with any "discarded library" warning messages.

- Copy `lib\glui.lib` to the `lib` directory of your Microsoft Visual C++ installation. By default this is the directory "Program Files\Microsoft Visual Studio\VC98\lib"
- Copy `lib\glut32.lib` into the `lib` directory of Microsoft Visual C++ if necessary (just as for `glui.lib`)
- Open `win\pangu.dsw` using Microsoft Visual C++
- Set the active configuration to "_BuildEverything - Win32 Release" (Build → Set Active Configuration)
- Remove any temporary files (Build → Clean)
- Compile everything (Build → Build or press F7)

The executables will be placed in the `bin\win32` directory.

10.2.2 UNIX developers

UNIX developers are expected to compile the viewer and related tools using GNU C++ 2.96 on RedHat 7.x. The C++ standard template library will be needed so install the `libstdc++-devel` package if necessary. The build process is managed by a set of Makefiles with the top-level Makefile in the `src` directory. Each tool has its own set of Makefiles but these are invoked from the top-level when necessary.

The commands for building the executables are:

```
cd src
make clean
make install
```

The executables will be placed in the `bin/linux_x86` directory. If there are link errors pertaining to GLUI please copy `lib/libglui.so` into `/usr/lib` and ensure that it is world readable (`chmod 644 /usr/lib/libglui.so`).

10.3 NOTE ON MAKEFILES FOR UNIX DEVELOPERS

The GNU C++ compiler does not perform any source/object dependency analysis when producing executables. To get around this problem we have developed a set of Makefiles which automatically compute the necessary dependency information before invoking the compiler. The result is that the `src` directory and each of the tool directories use five Makefiles of which four are distributed with the sources (the other is created on demand):

`Makefile`
: This is the top-level Makefile which is read by `make`(1). It contains rules for building each tool along with rules for cleaning intermediate files and source-code maintenance. When a top-level rule is invoked this Makefile will cause the C and C++ dependencies to be written to `Makefile.depends` using compiler-specific rules from `Makefile.arch`. The same top-level rule from `Makefile.common` is then invoked.

`Makefile.arch`
: This file contains compiler and architecture-specific definitions for the compilation process. It is read by `Makefile` and `Makefile.common`. Currently this is a soft link to `Makefile.gcc`.

`Makefile.gcc`
: This file contains GCC and G++ definitions for x86-based Linux systems.

`Makefile.common`

This file controls the build process. The main rule creates object files for the sources in `src` before invoking the main rule for each tool in `FeatureListGen/Makefile`, `SurfaceGen/Makefile` and `PovGen/Makefile`.

`Makefile.depends`

This file contains source/object dependency information. It is recreated every time `make`(1) is used.

10.4 FURTHER NOTES FOR UNIX DEVELOPERS

During external testing it was discovered that the support scripts will not work if they are unpacked using DOS or Windows text format. If compilation under UNIX stops with the error “bad interpreter: No such file or directory” then the files in the scripts directory have been unpacked incorrectly. Converting the files into UNIX format is not as easy as it may seem if conversion utilities such as **unix2dos** are not available. The PANGU `to_unix.sh` script could be used except that this is likely to be corrupted too! One solution is to use VIM: to ensure that `clobber.sh` is in UNIX text format type the following commands precisely:

```
vi clobber.sh
:set fileformat=unix
:x
```

The last command causes the file to be re-saved and then quits the editor. Repeat for each file in the scripts directory.

11. CHANGES TO PANGU

In this section we describe the main changes introduced between different versions of PANGU and show their use.

11.1 VERSION 1.50

The major additions in this version of PANGU with respect to version 1.02 are:

- curved surface support
- flexible crater profile equations
- mesh subdivision
- bi-resolution meshes
- view culling
- resolution selection
- depth rescaling
- depth buffer optimisation
- different sky backgrounds
- improved flight file support (new commands)
- improved network support
- **imgview** can now render a sequence of .ppm and .pgm files as a movie
- area light support for **mkshadows**

Details of each of these additions are described in detail below.

11.1.1 Curved surfaces

Previous versions of PANGU were used to model areas of terrain in which the surface curvature was negligible and could be ignored. This version of PANGU allows users to generate models of sections of spherical bodies with a user defined radius of curvature. To generate a curved model representative of the surface of Mercury, edit or create a `PlanetProperties.txt` file with the following three lines:

```
Identifier = PANGU: Planet Properties File
Curved surface = 1
Radius of curvature = 2.44e+06
```

When generating a .pan model with the `-l` option of **surfacegen** the `-p PlanetProperties.txt` option must be specified before the `-l` option. Otherwise a default flat model will be produced. Note that the `-p` option is currently ignored by the surface generator when producing DEMs: these are always flat:

```
..\..\bin\surfacegen -p PlanetProperties.txt -l layers.txt
```

11.1.2 Flexible crater profile equations

The maximum crater depth and the maximum crater rim height are defined by the equation kD^c. In this version of PANGU users can specify the values of "k" and "c" in the crater model parameter file (`CraterModel.txt`). The "Crater depth constant" defines the value of "k" for the maximum crater depth equation while "Crater power constant" defines the value of "c". Likewise "Crater rim height constant" and "Crater rim power Constant" define the values of "k" and "c" for the maximum crater rim height equation. Users are advised to check their crater model parameter files before use to ensure that these parameters are defined: earlier versions of these files omit these parameters and will cause invalid and degenerate models to be generated which the **viewer** cannot display. See Section 5.2.6.22 for details.

11.1.3 Flat bottomed craters

The crater model has been extended to simulate crater filling in by flattening out the bottom of craters to simulate some Martian craters which erode differently to craters on weatherlesss bodies. This feature can be turned on or off in the crater model parameters file. To create a mixture of Lunar and flat bottomed craters you must add craters in two stages using a different crater model parameters file for each stage. A flatAgeFactor parameter controls the amount of

crater fill in. The normalised amount of the crater fill-in is determined from the crater depth decay and the crater flat age factor. See sections 5.2.6.24 and 5.2.6.25 for more details.

11.1.4 Mesh subdivision

To significantly improve the speed of mkshadows and to assist the viewer the current version of PANGU produces polygon meshes which are recursively divided into smaller pieces. Thus instead of producing a single mesh of perhaps 1025 by 1025 vertices **surfacegen** now produces a group of four objects representing each of the 513 by 513 vertex quadrants of the original mesh. Each of these four objects will be subdivided into quadrants until an internal limit is reached (currently 8 by 8 vertices). Although this means that model files contain many more objects, the hierarchical system enables **mkshadows** to quickly ignore large areas of the model which cannot shadow a given vertex. For example, the **mkshadows** can show that three of the top four quadrants do not cast a shadow onto the current vertex then there is an immediate saving of 75% of the original computation time required by previous versions of PANGU.

11.1.5 Bi-resolution meshes

As described in other parts of this document, PANGU models usually consists of hierarchies of two or more layers, each layer having a resolution twice that of the enclosing layer. In this version of PANGU **surfacegen** generates two versions for the centre of each layer: a high resolution version consisting of the group of higher resolution layers and a low resolution version consisting of a single mesh at the same resolution of the outer edges of the layer. If the **viewer** determines that the triangles in the high-resolution mesh are too small to be rendered accurately then it can use the low resolution mesh instead.

11.1.6 View culling

With the introduction of model subdivision (Section 11.1.2) the viewer performance has increased with its ability to ignore large parts of a model which can significantly reduce the number of polygons which need to be rendered. In extreme cases viewpoints close to the surface of a model with large pitch angles can be rendered at more than 60 frames per second. If the **viewer** can prove that a large section of a model cannot be seen from the current view point then it does not have to render that part of the model. Previous versions of the **viewer** had to render the entire model.

11.1.7 Resolution selection

The introduction of bi-resolution meshes (Section 11.1.5) enables the viewer to determine the optimum resolution for each layer of the model. Users can control the resolution switch using the **viewer.resolution_limit** INI option in `pangu.ini` or with the `-resolution` command line option. The value for these options is the angular size of the mesh in pixels: a value of 0.5 means that if the high resolution triangles become smaller than 0.5 pixels then the **viewer** will use the low resolution version of the mesh (where triangles will be approximately 1 pixel in size). Since the resolution switch occurs on a layer basis the **viewer** may still render some triangles that are much smaller than the resolution limit. This will occur when the viewpoint is close to the surface where some parts of the layer will be close to the camera and ought to be rendered at a higher resolution than the parts of the layer in the distance.

11.1.8 Depth rescaling

This is a technique to enable the viewer to render objects which are at large distances from the camera. The PANGU **viewer**, like many rendering systems, employs a depth-buffer to determine whether part of the model lies in front of other parts of the model. This buffer has a finite resolution (normally 16 bits) and it requires limits to be placed on the closest and furthest object that can be rendered: these are known as the near and far clipping planes and correspond to the front and back of the depth buffer respectively. For modern PCs the ratio of the distance to the far plane to the distance of the near plane should not exceed a value of approximately 1000 although models with coarse meshes may be able to support much larger ratios. Unfortunately PANGU needs to be able to render models containing geometry which varies in resolution from a few centimetres up to hundreds of metres so the ratio needs to be nearer one million to avoid parts of the model vanishing in unexpected ways when viewed from distances of a few tens of kilometres.

One solution to this problem is to use depth rescaling. If the distance to the far clipping plane is d then objects between the near plane and distance $d/2$ will be rendered normally. Objects at distances between $d/2$ and 8 will be rescaled

using a power law so that they will appear to be positioned at distances between *d*/2 and *d* respectively. The rescaling ensures that the 3D/2D perspective projections are unchanged.

With this technique users can select the optimal near and far plane distances for their graphics system (which might position the far plane as close as 1000 units from the camera) and still be able to render accurately objects that are many times further away (provided that distant objects are relatively large).

Depth rescaling imposes a significant performance penalty of the **viewer** frame rate (perhaps reducing it by a factor of two) but may significantly improve the quality of images obtained from large distances away from the model. To enable depth rescaling set the **viewer.rescale_depth** INI option in `pangu.ini` to true or use the `-rescale` command line option.

Users are currently advised to try depth buffer optimisation (Section 11.1.9) first however.

11.1.9 Depth buffer optimisation

This is a relatively new feature which has been added to the **viewer** and has not been exhaustively tested yet so users are requested to use it with caution. However, initial tests have shown that it is very promising and may replace depth rescaling completely in future versions of PANGU.

With this technique the **viewer** renders the first scene using the near and far clipping planes defined by the user. If the position of the near plane could be moved significantly closer to the model (perhaps because the viewpoint is far from the model) or if the far plane could be moved significantly closer to the camera (perhaps because the model isn't very large) then these adjustments will be made to the clipping planes and the scene will be redrawn. Each time a scene is redrawn the position of the clipping planes will be compared with their optimum positions within the user defined limits. Occasionally this will cause a scene to be rendered a second time to obtain the best view which will reduce the **viewer** frame rate slightly.

Since the user-defined clipping plane distances are used to limit the search for optimal positions the user is advised to set the near plane distance to a small value such as 1 (or a smaller positive non-zero value) and the far plane distance to a large value such as $1x10^{20}$ (`1e20`). The **viewer.near_plane** and **viewer.far_plane** INI options in `pangu.ini` or the `-znear` and `-zfar` command line options control these values.

To enable depth buffer optimisation set the **viewer.optimise_depth** INI option or use the `-optimise_depth` command line option. It is advisable to disable depth rescaling when this option is enabled.

11.1.10 Sky backgrounds

Previous versions of PANGU always rendered models against a black background. In PANGU 1.50 support has been added for five new types of sky background: painted/textured star fields, raw star fields, uniform red sky, uniform green sky and uniform blue sky. The different backgrounds can be accessed once the **viewer** has started by pressing the F5 key to cycle through the available options.

11.1.10.1 Painted sky backgrounds

The painted sky background can be accessed by setting the **viewer.sky** INI option to `painted` and by defining the star field texture with the **viewer.starmap** option or by using the `-sky painted` and `-starmap` command line options. The texture scale factor will be used to replicate the texture to cover the entire celestial sphere and the texture is redrawn according to the current camera attitude. Unfortunately the current version of the **viewer** will ignore any camera roll when rendering this texture. Additionally the angular speed of features in the texture map varies across the field of view: features near the edge of the field of view appear to move at a different speed to corresponding parts of the planet surface being rendered. Another drawback of this option is that large textures must be used: for a 30 degree field of view rendered at 1024 by 1024 pixels at 2048 by 2048 texture may be required which is beyond the capabilities of older graphics systems. However, for still images this option can produce stunning results and could be used to simulate atmospheric effects or views from planets in other parts of the galaxy.

Sample star field textures can be found in the `textures` directory distributed with PANGU with names such as `stars_2048x2048.ppm` indicating a 2048 square texture. These were generated with the **pnmforge** PBM+ utility.

11.1.10.2 Raw sky backgrounds

The raw sky background consists of unprocessed stars which ought to occupy a single pixel of the final image. The position and intensity of each star is determined by the star catalogue specified using the **viewer.star_catalogue** INI option or the `-star_catalogue` command line option. Further details can be found in Section 6.1.6.

11.1.10.3 Coloured sky backgrounds

The final sky background type is a uniform colour filling one of the three channels red, green or blue. These are intended to allow post-processing and image analysis algorithms to accurately separate the sky part of an image from the planet surface part of an image. Since PANGU surface images are often grey-scale (the pixel value in each of the three channels is the same) it is trivial to separate the sky from the model. For example, if an image is obtained with a raw sky background then an identical image with a coloured sky could be obtained and used to generate a mask.

11.1.11 Improved flight file support

Added new commands to the flight file system: **ambient_colour**, **boulder_colour**, **boulder_view**, **field_of_view**, **print**, **quaternion**, **quit**, **reset_frame_count**, **save**, **save_as**, **save_fmt**, **sky_type**, **sun_colour**, **surface_view**, **surface_colour** and **verify_sun**. See Section 6.1.4 for a full listing of all the commands.

11.1.12 Improved network support

The previous PANGU network protocol (known as protocol version 0.00) only supported camera positions and attitudes that could be represented using long integers. The latest protocol (version 1.00) has been redesigned to enable camera positions and attitudes to be specified using floating-point values, back-projection from pixel coordinates in images to 3D model coordinates as well as limited control over the way images are rendered (*e.g.* setting the field of view width and the sky rendering method). More details of this can be found in Section 7.3.2.

11.1.13 Movie-mode for imgview

The **imgview** program can now be used to display a sequence of PPM or PGM images one after the other as a simple animation. Every image must be the same size and the number and size of images only depends on the amount of memory available to imgview. To enable this mode specify the `-movie` command line option: every other filename on the command line will be used as a frame of the animation in the order specified.

11.1.14 Area light support for mkshadows

The **mkshadows** tool has been updated to enable area light sources to be simulated automatically. This can be used to avoid creating and merging multiple shadow maps. See Section 3.3.8 for more details.

11.2 VERSION 2.00

The major additions in this version of PANGU with respect to version 1.50 are:

- display of the action associated with keys when they are pressed
- display of keys and mouse functions to console
- the term altitude has been replaced by elevation
- progress indicators for `mkshadows` and `mktexture`
- display of model-view origin, user-defined landing site and surface coordinate origin
- support for user-defined distance units and scale factor for viewer output
- **pangu_client** can now read and execute .fli files to control the viewer remotely
- automatic enabling/disabling of surface texture and detail based on camera altitude
- support for NPAL and Melosh crater equations
- improved boulder burial modelling
- extra flight file command
- area light INI option for **mkshadows**
- improved `pangu.ini` files

- text format DEM generation
- remote client back-projection bug fixes

Details of each of these additions are described in detail below.

11.2.1 Action key feedback

If the `-report_keys` command line option has been specified or if the **viewer.report_keys** INI option is set to true then whenever a key is pressed the viewer will display the action associated with the key before performing the action. This enables users to confirm that they pressed the correct key and to help them to use the **viewer**.

11.2.2 Display of keys to console

A new button has been added to the control panel on the right hand side of the **viewer** window labelled "Commands". Pressing this button will cause a list of all keys understood by the **viewer** and their meaning to be written to the console output stream (known as standard output). This list is always written to the viewer error stream when the program starts up but under Windows this is normally redirected to a log file such as `PANGU.log`. Note that if the **viewer** has been launched without a command window then pressing this key will appear to have no effect: the output will be sent, in effect, to an invisible window.

11.2.3 Altitude/Elevation Terminology

Prior to PANGU 1.50 the term altitude was used to refer to the angle of an object relative to the horizontal plane. This was based on the astronomical altitude-azimuth system of mounting telescopes. Since this term may be confused with the notion of physical altitude relative to a planet surface it has been replaced with the term elevation. Thus the model view camera uses azimuth and elevation angles and the Sun position is measured in the same way.

11.2.4 Progress indicates for mkshadows/mktexture

The process of generating texture maps may take several minutes on a fast computer during which time the **mktexture** application appears to be doing nothing. Likewise the **mkshadows** application may run for days before finishing. To allow users to judge the progress of these applications a text-based progress indicator has been added.

For **mktexture** the number of 3x3 blocks is used as the measure of progress. The input DEM is divided into a set of 3x3 blocks; this is multiplied by the number of times the Sun is sampled (area light support) and by the number of screen tiles required. The number of screen tiles is approximately the number of samples in the input DEM divided by area of the **mktexture** window in pixels: a 2048x2048 DEM with a 512x512 mktexture window needs 16 tiles.

For **mkshadows** the number of vertices in the model is used as the measure of progress.

These progress indicators can be enabled or disabled via the `-[no]progress` command line option or via the **mktexture.show_progress** or **mkshadows.show_progress** INI options.

11.2.5 Model-view origin, landing site position and coordinate origin

To allow users to identify the current model-view origin (target), the origin of the surface coordinate system and the position of a user-defined landing site, the viewer has been extended with special decal and box objects. A decal is a textured square of user-defined dimensions which follows the shape of the surface on which it is laid.

The surface coordinate origin decal marker may be enabled or disabled via the `-[no]show_origin` command line option or by the **viewer.show_origin** INI option. The size, shape and texture properties of the origin decal are defined by the `-origin` command line option or by the **viewer.origin** INI option.

Likewise the model-view box marker is controlled by the `-[no]show_target` and `-target_marker` command line options and by the **viewer.show_target** and **viewer.target** INI options. This marker is only visible when in model-view mode.

Finally the user-defined landing-site decal marker is enabled/disabled by the `-[no]show_landing_site` command line option or the **viewer.show_landing_site** INI option. The position, size, shape and texture properties of the landing site marker are defined by the `-landing_site` command line option or the **viewer.landing_site** INI option.

11.2.6 Distance scale factor and units

Previously all distance units reported by the viewer were measured in metres. With PANGU 2.00 the user can define a scale factor to apply to all distances displayed by the viewer and an associated label for the units. For example, a scale factor of 0.001 could be used with a label of "km" to force the viewer to display all distances in kilometres. Note that the viewer always uses metres as the input units. The scale factor is defined by the `-output_scale` command line option or the **viewer.output_scale** INI option; the label for distance units is defined by the `-output_units` command line option or the **viewer.output_units** INI option. These default to 1 and "m" respectively.

11.2.7 Flight file support in pangu_client

The **pangu_client** application has been updated to support viewer .fli flight files. This enables the flight file support of the viewer to be used remotely: the images obtained from **pangu_client** in this mode will be exactly the same as those obtained by the **viewer** in movie mode.

Flight files (see Section 6.1.4) are normally used to specify the position and orientation of the camera at different points along a trajectory. The viewer (or **pangu_client**) can be instructed to save images at each of these points. Additionally the **viewer** properties such as sky rendering mode, Sun colour, field-of-view width *etc.* may be specified.

The `-flight` command line option or the **pangu_client.flight_file** INI option of **pangu_client** is used to specify the input flight file. The `-savefmt` command line option or the **pangu_client.save_format** INI option is used to specify the name of each saved frame (see Section 6.1.4). The `-save` command line option or the **pangu_client.save_frames** INI option is used to instruct **pangu_client** to save each frame (rather than simply displaying it).

Note that the **viewer** must be running in server mode before **pangu_client** can be used.

For example, if the file `sample.fli` contains a set of camera positions and attitudes then the command

```
pangu_client -save -flight sample.fli -savefmt 'frames/f_%04d.ppm'
```

will send each camera position and attitude from `sample.fli` to the viewer and save the corresponding images into the files `frames/f_0000.ppm`, `frames/f_0001.ppm`, etc. Without the `-save` option the images would not be saved unless **pangu_client.save_frames** was set to true.

11.2.8 Automatic surface texture and surface detail enabling

The user can now instruct the viewer to enable surface texture or surface detail whenever the camera altitude drops below a specific height. The `-trigger_texture` command line option or **viewer.trigger_texture** INI option specifies the camera altitude above which surface texture is disabled and below which surface texture is enabled. The `-trigger_detail` command line option or the **viewer.trigger_detail** INI option specifies the camera altitude above which surface detail is disabled and below which surface detail is enabled. The trigger altitude for surface texture must be greater than the trigger altitude for surface detail. If one or both trigger altitudes are unspecified then that texture mode will not be automatically triggered.

For example, to enable surface texture as soon as the camera drops below an altitude of 500 m and to switch to surface detail as soon as the camera drops below 50 m the following command may be used:

```
viewer -trigger_texture 500 -trigger_detail 50
```

Above 500 m texture will not be used. Alternatively to only enable surface detail below 100 m and disable all texture above that altitude the following command can be used:

```
viewer -trigger_detail 50
```

Note that when texture triggering is used it is not possible to manually enable or disable surface texture or detail.

11.2.9 NPAL and Melosh crater equations

In addition to allowing the user to specify the coefficients and exponents in the crater profile equations the user can now specify a scale factor. The NPAL profile equations are defined in units of metres and this notation has been adopted by PANGU 2.00; the Melosh profile equations used by older versions of PANGU are defined in units of kilometres. To resolve this situation the `CraterModel.txt` parameter file has been extended with an extra parameter defining the number of metres in the units used for the profile equations. The default is 1 (NPAL equations) and so if Melosh equations are used this parameter must be set to 1000 (1 km). See Section 5.2.6.23 for details.

11.2.10 Improved boulder burial modelling

Boulder modelling has been improved by interpreting the boulder burial measure as a fraction of the boulder size rather than a physical depth. This means that it is possible to ensure that all boulders, for example, are at least 50% buried. This eliminates the problem of tall narrow boulders appeared to defy gravity. Users may wish to alter the boulder depth distribution files (normally called `BoulderDepthDist.txt` and `BoulderCRDepthDist.txt`).

11.2.11 Extra flight file command

The **aspect_ratio** flight file command has been added which allows the pixel aspect ratio to be defined. See Section 6.1.4 for a full listing of all the flight file commands.

11.2.12 Area light INI option for mkshadows

The sun.sample_method INI option allows users to select point-sources or ring-sampled disks. See Section 3.3.8.4.

11.2.13 Improved pangu.ini files

The `testmodel` and `lib` directories contain an updated `pangu.ini` file with every option listed along with default settings and complete documentation for PANGU 2.00. See Section 8.5 for a complete description.

11.2.14 Text format DEM generation

To enable users to validate the back-projection facilities of PANGU, **surfacegen** has been updated to generate a semicolon-separated text file of DEM coordinates in addition to the `.pan` file. The DEM file is created by specifying the `-Y` command line option followed by the name of the output file before the `-L` or `-C` option of **surfacegen**. The DEM file is only created when a `.pan` file is created so this option must be used in conjunction with `-L` or `-C`.

The first line of the DEM file is a label for the data columns (X; Y; Z;); the remaining lines contain three semicolon-separated values representing the x and y LDS coordinates of points on the surface followed by the elevation at those points. Each point in the file corresponds to a vertex of the model when viewed in wireframe mode in the **viewer**. Note that model re-centering (**viewer.recenter**) must be disabled in the **viewer** to ensure that back-projection values match those in the DEM file.

11.2.15 Remote client back-projection bug fixes

Science Systems reported that quaternion rotations in PANGU 1.50 were not behaving in the way that they were documented and that coordinate look-ups by remote clients were returning incorrect results. These problems were due to four bugs which have all been fixed in PANGU 2.00:

1) the wrong internal camera was used for **LookupPoint** and **LookupPoints** requests
2) the y-coordinate of **LookupPoint** and **LookupPoints** requests had an error of one pixel
3) yaw and roll quaternions were swapped in **SetViewpointByQuaternion**
4) incorrect handling of camera position in **SetViewpointByQuaternion**

Astrium subsequently reported that the camera frame was not aligned correctly with the NPAL LDS landing site frame. The unit quaternion ought to ensure that the camera frame is identical to the LDS: since the camera looks along

the z-axis of the camera frame the unit quaternion must generate a view looking along the NPAL LDS z-axis *i.e.* directly up at the zenith. This problem has been fixed in PANGU 2.00.

11.3 VERSION 2.10

The major additions in this version of PANGU with respect to version 2.00 are:

- asteroid model generation
- whole-planet model generation for long-distance views

11.4 VERSION 2.40

The major additions in this version of PANGU with respect to version 2.10 are:

- updated crater models to support modelling of Martian surfaces
- addition of sand dune modelling
- TCP/IP support for LIDAR sensor simulation

11.4.1 Sand Dune Modelling

The ability to model sand dunes has been added to PANGU. Surfacegen is used to expand a surface hierarchy with a sand dune parameter file specified. This enables the stochastic generation of dunefields. Dunes can be created simply based on a sand supply and wind regime. Alternatively a further set of parameter files can be used to define the statistical variation of dune metrics for the dune field.

11.5 VERSION 2.50

The major additions in this version of PANGU with respect to version 2.40 are:

- memory management system for the **viewer** and **mkshadows**
- support for generating and rendering terrain using the ROAM algorithm

These changes are described in more detail below:

11.5.1 Memory management

The **viewer** and **mkshadows** have been extended with a memory management system which is designed to reduce the amount of RAM used to store the PANGU model being rendered or shadow mapped. In addition to optimisations of the runtime representation of objects to reduce wastage, mesh objects have been extended to support lazy loading and caching. These benefits can be seen by comparing the performance of PANGU 2.40 with PANGU 2.50 when loading and rendering large models. These changes are described in detail in Section 3.4.2.

11.5.2 ROAM Terrain

The ROAM algorithm is a technique for generating “optimal” triangle meshes which can be rendered using OpenGL or other systems. The meshes are optimal with respect a chosen quality metric for the ROAM mesh tessellation. The use of this for creating and viewing PANGU models is described in detail in Section 3.4.3.

11.6 VERSION 2.60

The major addition in this version of PANGU with respect to version 2.50 is the addition of RADAR support to the PANGU network protocol. See Section 3.4.5 and **GetRadarResponse**/**RadarResponse** in Section 7.3.2 for more details.

11.7 VERSION 2.65

This version of PANGU adds fog/dust cloud support. See Section 3.4.6 for more details.

11.8 VERSION 2.70

This version of PANGU extends fog/dust cloud support with the addition of dust/fog inside a sphere and dust/fog inside a general volume defined by a PANGU `.pan` file. See Section 3.4.6 for more details.

Additionally, the image-saving facility has been extended to allow the saved images to be larger than the **viewer** window in which scenes are rendered. The **viewer.save_scale** integer INI option and the `-save_scale` command line option both define the size of saved images relative to the size of the **viewer** window. This enables images larger than the size of the screen of the host computer to be generated.

Three self-test facilities were also added to allow the lighting model and back-projection equations of the **viewer** to be validated. These facilities are described in Section 3.4.7.

Finally an experimental option to enable the use of vertex arrays (an OpenGL rendering facility) has been provided via the **viewer.use_vertex_arrays** INI option and `-vertex_arrays` command line argument. When enabled this option might increase rendering speed by a factor of two. However, it may cause boulder colours to be rendered incorrectly so this option should not be enabled when images are required for scientific or validation purposes.