API Documentation (DRAFT)

ossimPlanet

Prepared by David Lucas - dlucas@radiantblue.com

Table of Contents

ossimPlanetPointModel.h

```
#include <ossimPlanet/ossimPlanetPointModel.h>
```

The point model class is located in
ossimPlanet/include/ossimPlanet/ossimPlanetPointModel.h.

Description:

This will be the root class to handle all point-based models. Typically this may pertain to buildings or other 3-D point models. This will also own an ossimLsrSpaceTransform that allows one to talk to the MatrixTransform in a common Euler orientation, latitude longitude altitude placement, and scale settings. The Lsr stands for Local Space Reference and mirrors the concepts in the ossim core engine.

Caveat: If the setMatrix is called on a MatrixTransform, which is a base of ossimPlanetLsrSpaceTransform, then it will not sync the Euler and latitude longitude altitude parameters until the next call to computeBound. Note setMatrix will dirty the bound. So if you want immediate sync to occur then you must manually say getBound immediately after a setMatrix. This should force a re-compute.

Public Member Functions:

update

Description:

For now we will do nothing for the update. This is reserved for future background operations on a node to update itself. It will be up to the node's responsibility to

update only its dirty components. Note this is not synched to the Update Visitor but this virtual method is an interface to an UpdateOperation.

Syntax:

```
virtual void update(){}
```

traverse

Description:

It will keep in sync during the update stage the node this point model is controlling with an LsrSpace transform.

Syntax:

```
virtual void traverse(osg::NodeVisitor& nv);
```

Parameter:

```
osg::NodeVisitor& nv
```

nv – node visitor base for all visitors.

setNode

Description:

This will be the point model controlled by this class. During the update traversal it will be added to the ossimPlanetLsrSpaceTransform.

Syntax:

```
void setNode(osg::Node* node);
```

Parameter:

```
osg::Node* node
```

node - This node is assumed to be relative values from the LsrSpaceTransform.

lsrSpace (constant)

Description:

Returns a constant reference to a Local Space Reference for this point model.

Syntax:

```
const ossimPlanetLsrSpaceTransform* lsrSpace()const{return
theLsrSpaceTransform.get();}
```

lsrSpaceTransform

Description:

Returns a reference to a Local Space Reference for this point model.

Syntax:

```
ossimPlanetLsrSpaceTransform* lsrSpace(){return
theLsrSpaceTransform.get();}
```

copyLsrSpaceParameters

Description:

This will copy the LSR parameters.

Syntax:

```
void copyLsrSpaceParameters(const ossimPlanetLsrSpaceTransform& lsr);
```

Parameter:

```
(const ossimPlanetLsrSpaceTransform& lsr);
```

lsr - The Local Space Reference to copy the parameters from.

setLayer

Description:

Overrides set layer so we can pass the model immediately to the Local Space Reference transform.

Syntax:

```
virtual void setLayer(ossimPlanetLayer* layer);
```

Parameter:

```
ossimPlanetLayer* layer
```

layer - The ossimPlanetLayer we are part of.

computeBound

Syntax:

```
virtual osg::BoundingSphere computeBound()const
   {
      if(theLsrSpaceTransform.valid())
      {
         return theLsrSpaceTransform->computeBound();
      }
      return osg::BoundingSphere(osg::Vec3d(0.0,0.0,0.0), -1);
   }
```

Chapter Summary

```
// Create a new annotation layer.
viewer->planet()->addChild(new ossimPlanetAnnotationLayer);

// Create a new point model object.
osg::ref_ptr<ossimPlanetPointModel> thePointModel1 = new
ossimPlanetPointModel;

// Add the point model to the annotation layer.
```

```
viewer->addAnnotation(thePointModel1.get());

// Set the node to point model.  Node is some local 3-D model that you
wish to position.
thePointModel1->setNode(node);

// Set the latitude, longitude, and altitude of the point model.
thePointModel1->lsrSpace()->setLatLonAltitude(osg::Vec3d(28.2395, -
80.6076, 50.9824));
// The point model is set at 28.2395 degrees latitude, -80.6076 degrees
longitude, and 50.9824 meters altitude above the ellipsoid.  Note that
this call assumes ellipsoidal height.  Use the mean sea level call for
heights above the geoidal egm grid.

Refer to the ossimPlanetLsrSpaceTransform class for more information.
```


ossimPlanetLsrSpaceTransform.h

```
#include <ossimPlanet/ossimPlanetLsrSpaceTransform.h>
```

The lsr space transform class is located in *ossimPlanet/include/ossimPlanet/ossimPlanetLsrSpaceTransform.h*.

Description:

ossimPlanetLsrSpaceTransform allows one to talk to osg::Transform in the form of Euler angles and lat lon altitude where altitude is in meters. The Lsr stands for Local Space Reference. This manages a local space axis at the given <lat,lon,altitude> point.

Public Member Functions:

ossimPlanetLsrSpaceTransform

Description:

Constructs an LsrSpace with a given geo reference model and orientation type.

Syntax:

```
ossimPlanetLsrSpaceTransform(ossimPlanetGeoRefModel* model=0);
```

Parameter:

```
ossimPlanetGeoRefModel* model=0
```

model - The Geo Reference Model to use.

ossimPlanetLsrSpaceTransform (constant)

Description:

Copy constructor.

Syntax:

```
ossimPlanetLsrSpaceTransform(const ossimPlanetLsrSpaceTransform&
lsrTransform,const osg::CopyOp& copyop=osg::CopyOp::SHALLOW_COPY);
```

setModel

Description:

Allows one to change the geo reference model. Will apply the new model to the coordinates and dirty the bound.

Syntax:

```
void setModel(ossimPlanetGeoRefModel* model);
```

Parameter:

```
ossimPlanetGeoRefModel* model
```

model - The model to use to generates the lsrMatrix.

copyParametersOnly

Description:

Will copy the parameters only form one transform to the next.

Syntax:

```
void copyParametersOnly(const ossimPlanetLsrSpaceTransform& src)
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      dirtyBound();
      if(!theModel.valid())
      {
         theModel = src.theModel;
      }
      setHeadingPitchRoll(src.headingPitchRoll());
      setLatLonAltitude(src.latLonAltitude());
      setScale(src.scale());
      theOrientationMode = src.theOrientationMode;
   }
```

Parameter:

```
const ossimPlanetLsrSpaceTransform& src
```

src - This is the source to copy from.

ossimPlanetGeoRefModel (constant)

Description:

Returns the pointer to the active model used by the transform.

Syntax:

```
const ossimPlanetGeoRefModel* model()const
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theModel.get();
   }
```

ossimPlanetGeoRefModel

Description:

Returns the pointer to the active model used by the transform.

Syntax:

```
ossimPlanetGeoRefModel* model()
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theModel.get();
   }
```

setMatrix

Description:

This is the local to world transform matrix. It will extract out the translation, scale, and orientation from the past in matrix and set the Lsr parameters.

Syntax:

```
void setMatrix(const osg::Matrix& m);
```

Parameter:

```
const osg::Matrix& m
```

m - The matrix to copy and extract the Lsr space parameters.

lat

Description:

Returns the latitude position in degrees.

Syntax:

```
double lat()const
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theLatLonAltitude[0];
   }
```

Example:

```
std::cout << "lat: " << thePointModel1->lsrSpace()->lat() << std::endl;

Result:
lat: 28.2395
```

lon

Description:

Returns the longitude position in degrees.

Syntax:

```
double lon()const
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theLatLonAltitude[1];
   }
```

Example:

```
std::cout << "lon: " << thePointModel1->lsrSpace()->lon() << std::endl;

Result:
lon: -80.6076
```

altitude

Description:

Returns the altitude in meters.

Syntax:

```
double altitude()const
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theLatLonAltitude[2];
   }
```

Example:

```
std::cout << "altitude: " << thePointModel1->lsrSpace()->altitude() <<
std::endl;

Result:
altitude: 50.9824
```

latLonAltitude

Description:

Returns latLonAltitude as a vector.

Syntax:

```
const osg::Vec3d& latLonAltitude()const
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theLatLonAltitude;
   }
```

Example:

```
std::cout << "latLonAltitude: " << thePointModel1->lsrSpace()-
>latLonAltitude() << std::endl;

Result:
latLonAltitude: 28.2395 -80.6076 50.9824
```

x

Description:

Returns the model X coordinate of the model that was formed from the inverse of the lat lon altitude.

Syntax:

```
double x()const
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theXYZ[0];
   }
```

Example:

```
std::cout << "x: " << thePointModel1->lsrSpace()->x() << std::endl;

Result:
x: 0
```

y

Description:

Returns the model Y coordinate of the model that was formed from the inverse of the lat lon altitude.

Syntax:

```
double y()const
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theXYZ[1];
   }
```

Example:

```
std::cout << "y: " << thePointModel1->lsrSpace()->y() << std::endl;

Result:
y: 0
```

z

Description:

Returns the model Z coordinate of the model that was formed from the inverse of the lat lon altitude.

Syntax:

```
double z()const
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theXYZ[2];
   }
```

Example:

```
std::cout << "z: " << thePointModel1->lsrSpace()->z() << std::endl;

Result:
z: 0
```

xyz

Description:

Returns the model XYZ coordinate of the model that was formed from the inverse of the lat lon altitude.

Syntax:

```
const osg::Vec3d& xyz()const
```

```
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theXYZ;
   }
```

Example:

```
std::cout << "xyz: " << thePointModel1->lsrSpace()->xyz() << std::endl;

Result:
xyz: 0 0 0
```

heading

Description:

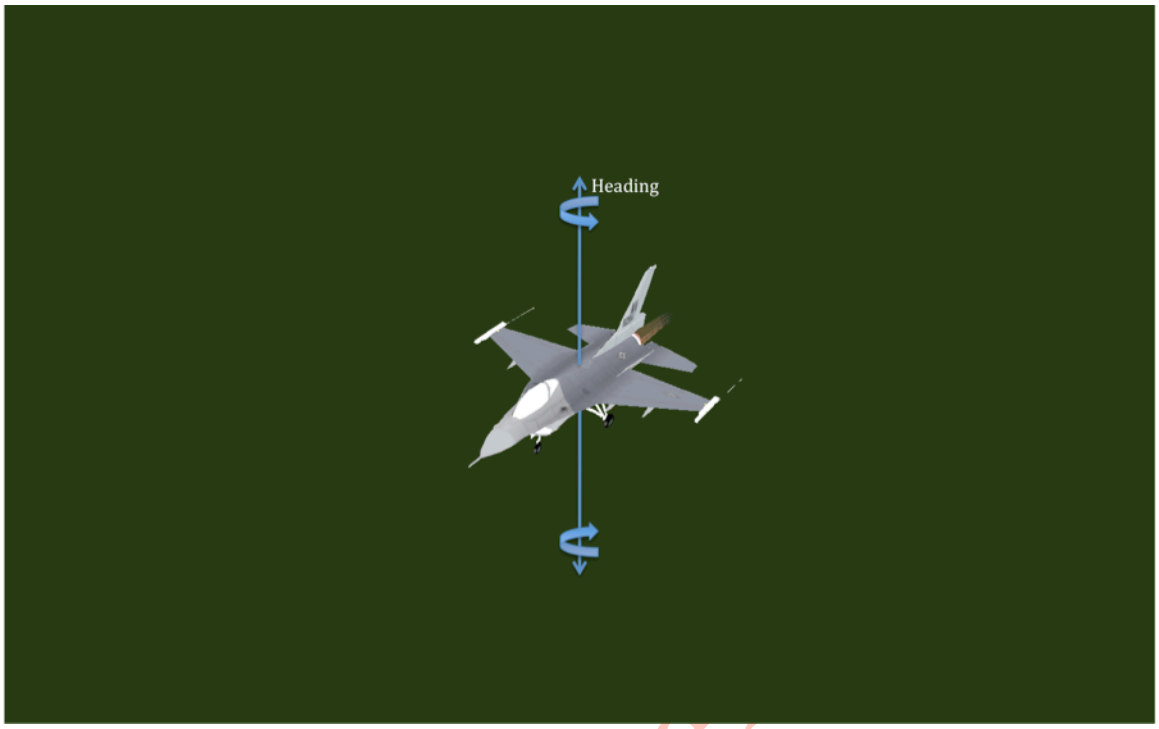
Returns the heading orientation in degrees. Indicates the rotation about the nadir-axis.

Syntax:

```
double heading()const
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theHpr[0];
   }
```

Example:

```
std::cout << "heading: " << thePointModel1->lsrSpace()->heading() <<
std::endl;

Result:
heading: 200
```



pitch

Description:

Returns the pitch orientation in degrees. Indicates the rotation about the x-axis.

Syntax:

```
double pitch()const
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theHpr[1];
   }
```

Example:

```
std::cout << "pitch: " << thePointModel1->lsrSpace()->pitch() <<
std::endl;

Result:
```

```
pitch: 20
```

roll

Description:

Returns the roll orientation in degrees. Indicates the rotation about the y-axis.

Syntax:

```
double roll()const
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theHpr[2];
   }
```

Example:

```
std::cout << "roll: " << thePointModel1->lsrSpace()->roll() <<
std::endl;
```

```
Result:
roll: 0
```

headingPitchRoll

Description:

Returns all values for heading pitch and roll.

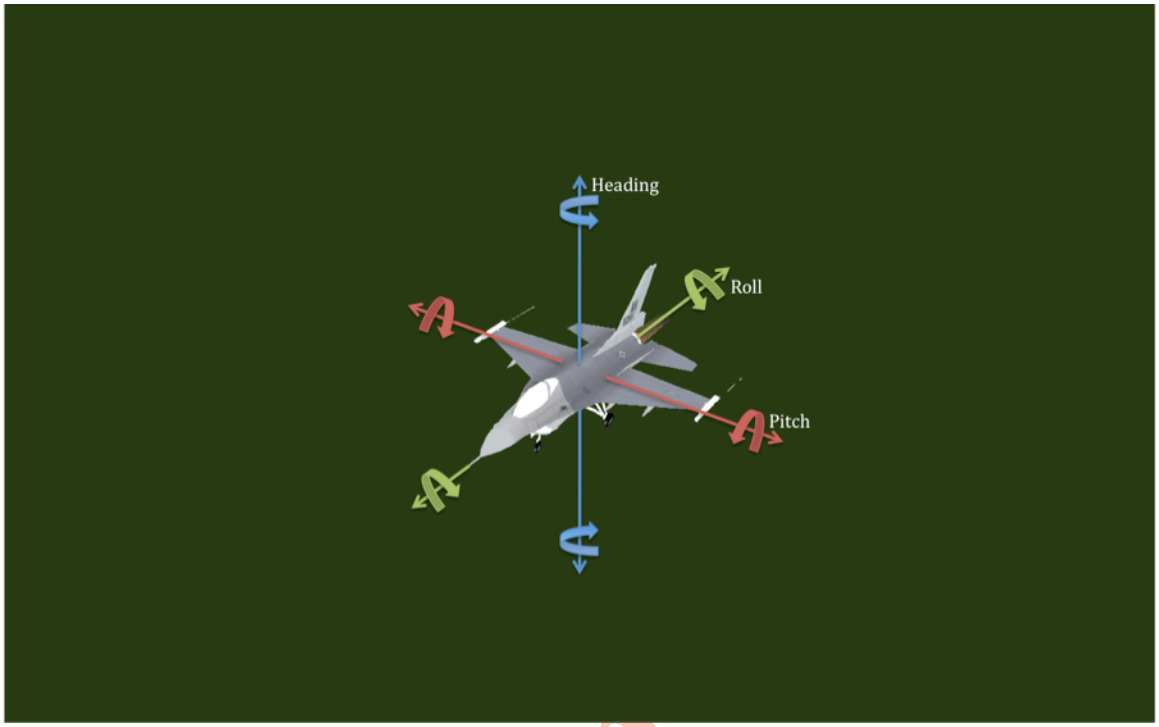
Syntax:

```
const osg::Vec3d& headingPitchRoll()const{return theHpr;}
```

Example:

```
std::cout << "headingPitchRoll: " << thePointModel1->lsrSpace()-
>headingPitchRoll() << std::endl;

Result:
```

```
headingPitchRoll: 200 20 0
```



scalex

Description:

Returns the x scale factor.

Syntax:

```
double scalex()const
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theScale[0];
   }
```

Example:

```
std::cout << "scalex: " << thePointModel1->lsrSpace()->scalex() <<
std::endl;
```

```
Result:
scalex: 1
```

scaley

Description:

Returns the y scale factor.

Syntax:

```
double scaley()const
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theScale[1];
   }
```

Example:

```
std::cout << "scaley: " << thePointModel1->lsrSpace()->scaley() <<
std::endl;

Result:
scaley: 1
```

scalez

Description:

Returns the z scale factor.

Syntax:

```
double scalez()const
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theScale[2];
```

```
    }
```

Example:

```
std::cout << "scalez: " << thePointModel1->lsrSpace()->scalez() <<
std::endl;

Result:
scalez: 1
```

scale

Description:

Returns the scale for x,y,z as a vector.

Syntax:

```
const osg::Vec3d& scale()const
    {
        OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
        return theScale;
    }
```

Example:

```
std::cout << "scale: " << thePointModel1->lsrSpace()->scale() <<
std::endl;

Result:
scale: 1 1 1
```

setHeadingPitchRoll

Description:

Allows one to set the heading, pitch, and roll orientation.

Syntax:

```
void setHeadingPitchRoll(const osg::Vec3d& hpr);
```

Parameter:

```
const osg::Vec3d& hpr
```

hpr – contains the heading pitch and roll in that order for the orientation.

Examples:

```
thePointModel1->lsrSpace()->setHeadingPitchRoll(osg::Vec3d(200,20,0));

Result:
200 Degree Heading
20 Degree Pitch
0 Degree Roll
```

```
Set Heading Orientation

pointModel->lsrSpace()->setHeadingPitchRoll(osg::Vec3d(0,0,0));
Result:
0 Degree Heading


pointModel->lsrSpace()->setHeadingPitchRoll(osg::Vec3d(90,0,0));
Result:
90 Degree Heading


pointModel->lsrSpace()->setHeadingPitchRoll(osg::Vec3d(180,0,0));
Result:
180 Degree Heading


pointModel->lsrSpace()->setHeadingPitchRoll(osg::Vec3d(270,0,0));
Result:
270 Degree Heading


pointModel->lsrSpace()->setHeadingPitchRoll(osg::Vec3d(360,0,0));
Result:
360 Degree Heading
```

Set Pitch Orientation

```
pointModel->lsrSpace()->setHeadingPitchRoll(osg::Vec3d(0,0,0));
```
Result:
0 Degree Pitch

```
pointModel->lsrSpace()->setHeadingPitchRoll(osg::Vec3d(0,90,0));
```
Result:
90 Degree Pitch

```
pointModel->lsrSpace()->setHeadingPitchRoll(osg::Vec3d(0,180,0));
```
Result:
180 Degree Pitch

```
pointModel->lsrSpace()->setHeadingPitchRoll(osg::Vec3d(0,270,0));
```
Result:
270 Degree Pitch

```
pointModel->lsrSpace()->setHeadingPitchRoll(osg::Vec3d(0,360,0));
```
Result:
360 Degree Pitch

Set Roll Orientation

```
pointModel->lsrSpace()->setHeadingPitchRoll(osg::Vec3d(0,0,0));
```
Result:
0 Degree Roll

```
pointModel->lsrSpace()->setHeadingPitchRoll(osg::Vec3d(0,0,90));
```
Result:
90 Degree Roll

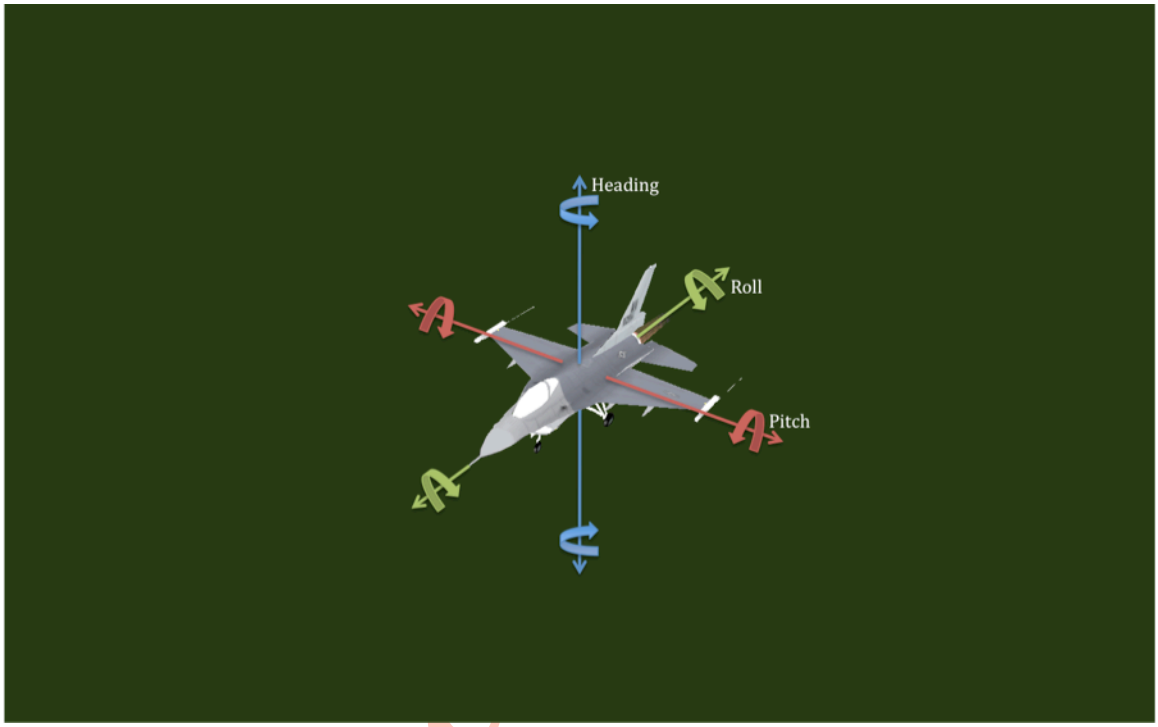
```
pointModel->lsrSpace()->setHeadingPitchRoll(osg::Vec3d(0,0,180));
```
Result:
180 Degree Roll

```
pointModel->lsrSpace()->setHeadingPitchRoll(osg::Vec3d(0,0,270));
```
Result:
270 Degree Roll

```
pointModel->lsrSpace()->setHeadingPitchRoll(osg::Vec3d(0,0,360));
```

Result:

360 Degree Roll



setLatLonAltitude (relative to ellipsoid)

Description:

Allows one to set the latitude, longitude, and altitude position relative to the ellipsoid.

Syntax:

```
void setLatLonAltitude(const osg::Vec3d& value);
```

Parameter:

```
const osg::Vec3d& value
```

value - contains the position lat lon and altitude where lat and lon are in degrees and altitude is in meters relative to the ellipsoid.

Example:

```
thePointModel1->lsrSpace()->setLatLonAltitude(osg::Vec3d(28.2395, -
80.6076, 50.9824));

Result:
28.2395 Latitude
-80.6076 Longitude
50.9824 Altitude
```

setLatLonAltitudeMeanSeaLevel (relative to geoid)

Description:

Allows one to set the latitude, longitude, and altitude position relative to the current geoid contained in the ossimPlanetGeoRefModel.

Syntax:

```
void setLatLonAltitudeMeanSeaLevel(const osg::Vec3d& value);
```

Parameter:

```
const osg::Vec3d& value
```

value - contains the position lat lon altitude where lat lon are in degrees and altitude is in meters relative to the current geoid contained in the ossimPlanetGeoRefModel.

Example:

```
thePointModel1->lsrSpace()-
>setLatLonAltitudeMeanSeaLevel(osg::Vec3d(28.2395, -80.6076, 50.9824));

Result:
28.2395 Latitude
-80.6076 Longitude
50.9824 Altitude
```

setScale

Description:

Allows one to set the x-scale, y-scale, z-scale components.

Syntax:

```
void setScale(const osg::Vec3d& value);
```

Parameter:

```
const osg::Vec3d& value
```

value - contains the values for the x-scale, y-scale, z-scale components.

Examples:

```
thePointModel1->lsrSpace()->setScale(osg::Vec3d(0.5,0.5,0.5));
```

Result:
Half Scale (.5x)

```
```

```
thePointModel1->lsrSpace()->setScale(osg::Vec3d(1,1,1));

Result:
True Scale (1x)
Default Scaling
```

```
thePointModel1->lsrSpace()->setScale(osg::Vec3d(2,2,2));

Result:
Double Scale (2x)
```

setXYZ

Description:

Allows one to set the model XYZ coordinate of the model that was formed from the inverse of the latitude longitude altitude.

Syntax:

```
void setXYZ(const osg::Vec3d& xyz);
```

Parameter:

```
const osg::Vec3d& xyz
```

xyz - the x,y,z values resulting from the conversion of lat, lon, altitude using the ossimPlanetGeoRefModel.

computeLocalToWorldMatrix

Description:

Override the base class Transform localToWorldMatrix.

Syntax:

```
virtual bool computeLocalToWorldMatrix(osg::Matrix&
matrix,osg::NodeVisitor* nv) const;
```

computeWorldToLocalMatrix

Description:

Override the base class world to local transforms.

Syntax:

```
virtual bool computeWorldToLocalMatrix(osg::Matrix&
matrix,osg::NodeVisitor* nv) const;
```

traverse

Description:

Override the traverse to make sure that the model pointer is set and if the redraw flag is set then pass it up.

Syntax:

```
virtual void traverse(osg::NodeVisitor& nv);
```

Chapter Summary

```
// Set the latitude, longitude, and altitude of the point model.
thePointModel1->lsrSpace()->setLatLonAltitude(osg::Vec3d(28.2395, -
80.6076, 50.9824));
// The point model is set at 28.2395 degrees latitude, -80.6076 degrees
```

```
longitude, and 50.9824 meters altitude above the ellipsoid.  Note that
this call assumes ellipsoidal height.  Use the mean sea level call for
heights above the geoidal egm grid.

Result:
28.2395 Latitude
-80.6076 Longitude
50.9824 Altitude

// Set the orientation of the point model.
thePointModel1->lsrSpace()->setHeadingPitchRoll(osg::Vec3d(200,20,0));
// The point model is set at 200 degrees heading, 20 degrees pitch, and
0 degrees roll.

Result:
200 Degree Heading
20 Degree Pitch
0 Degree Roll

// Set the scale of the point model.
thePointModel1->lsrSpace()->setScale(osg::Vec3d(1,1,1));
// The point model is set at 1,1,1 scaling which is also the default
scaling.


Result:
1, 1, 1
True Scale (1x)
Default Scaling
```

Refer to the ossimPlanetPointModel class for more information.

ossimPlanetTerrain.h

```
#include <ossimPlanet/ossimPlanetTerrain.h>
```

The terrain class is located in *ossimPlanet/include/ossimPlanet/ossimPlanetTerrain.h*.

Description:

ossimPlanetTerrain allows one to control mesh density and topographic elevation sources.

Public Member Functions

addElevation

Description:

Allows one to add elevation data to the terrain layer. The system is able to parse DTED, SRTM and general formats for the construction of elevation meshes.

Syntax:

```
bool addElevation(const ossimFilename& file, bool sortFlag=false);
```

Parameters:

```
const ossimFilename& file
```

file - The path to an elevation directory.

```
bool sortFlag=false
```

sortFlag - The flag to enable or disable sorting by GSD.

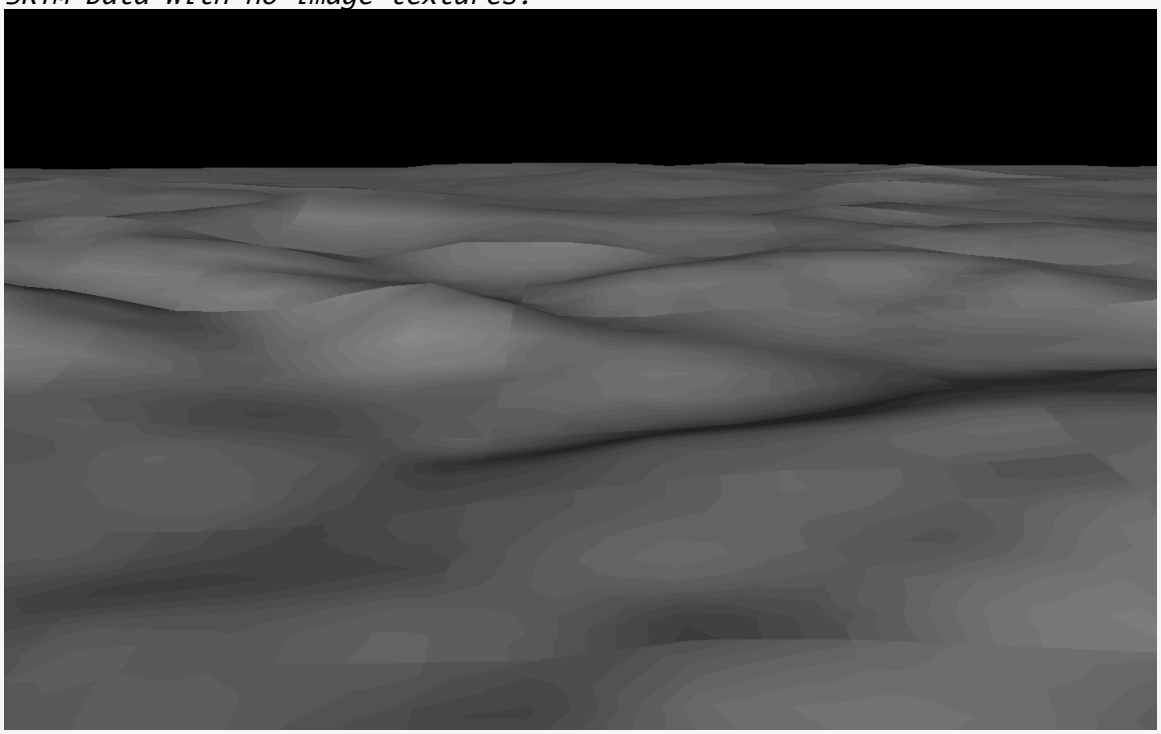
Note: The elevation directory is not completely recursed. The first few files searched in the directory will determine the elevation database type. Planet has detectors for DTED, SRTM, and general raster. General rasters are handled differently. Currently we take the first general raster and use it as the elevation data.

Post spacing is assumed on the first elevation file found in the directory. After the assumption is made the entire database is set to have the same post spacing. DTED and SRTM levels cannot be mixed in the same elevation database. If having DTED and SRTM elevation data is desired, then two separate elevation database will be needed. The database itself is sorted to other registered databases.

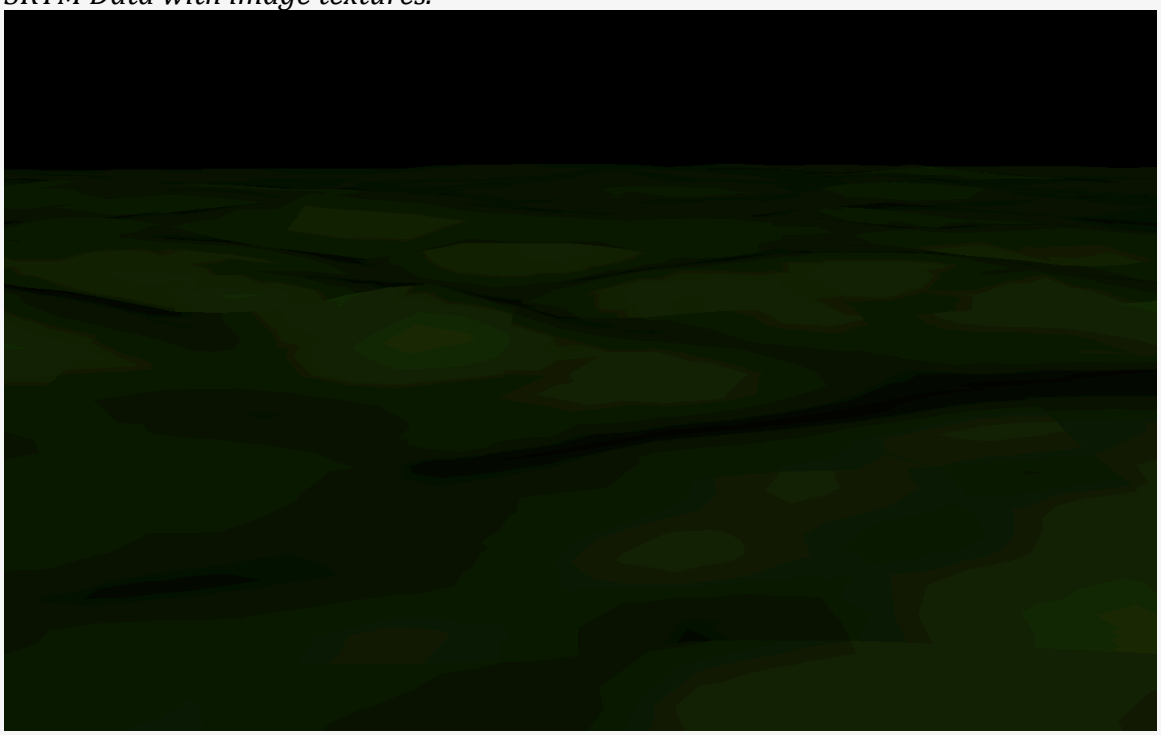
Example:

```
viewer->terrainLayer()-
>addElevation("/data/ossimplanettest/srtm30plus/" 1);

Result:
Elevation in /data/ossimplanettest/strm30plus/ will be added and sorted
by GSD.
```

SRTM Data with no image textures.



SRTM Data with image textures.



addElevation

Syntax:

```
bool addElevation(osg::ref_ptr<ossimPlanetElevationDatabase> database,
bool sortFlag=false);
```

setNumberOfTextureLayers

Description:

Allows one to set the number of texture layers to the planet. Typically these will be imagery or map overlays to the elevation mesh. Each texture can be a mosaic of input images. The term texture layer here refers to GL texture layer. For a single GL texture layer we can have a mosaic of N number of images. This is typically used for GL shaders attached to the terrain.

Syntax:

```
void setNumberOfTextureLayers(ossim_uint32 size);
```

Parameter:

```
void setNumberOfTextureLayers(ossim_uint32 size);
```

size - The number of texture layers to set.

Examples:

```
terrain->setNumberOfTextureLayers(2);
```

numberOfTextureLayers

Description:

Returns the number of texture layers from the terrain.

Syntax:

```
ossim_uint32 numberOfTextureLayers()const;
```

Examples:

```
std::cout << "numberOfTextureLayers = " << terrain-
>numberOfTextureLayers() << std::endl;
```

Result:
numberOfTextureLayers: 2

refreshImageLayers

Description:

Refreshes image layers from the terrain.

Syntax:

```
void refreshImageLayers();
```

Example:

```
viewer->terrainLayer()->refreshImageLayers();
```

refreshElevationLayers

Description:

Refreshes elevation layers from the terrain.

Syntax:

```
void refreshElevationLayers();
```

Example:

```
viewer->terrainLayer()->refreshElevationLayers();
```

refreshImageAndElevationLayers

Description:

Refreshes image and elevation layers from the terrain.

Syntax:

```
void refreshImageAndElevationLayers();
```

Example:

```
viewer->terrainLayer()->refreshImageAndElevationLayers();
```

resetImageLayers

Description:

Resets image layers from the terrain.

Syntax:

```
void resetImageLayers();
```

Example:

```
viewer->terrainLayer()->resetImageLayers();
```

resetGraph

Description:

Resets the graph.

Syntax:

```
void resetGraph();
```

setCullAmountType

Description:

Allows one to set the cull amount type. Culling is a way to remove objects from the graph. Culling is a means of reducing the processing and storage requirements on the system. The higher the cull amount, the fewer terrain objects in the graph. Objects are typically pruned or culled from the graph when they are no longer in the field of view.

Syntax:

```
enum CullAmountType
   {
      NO_CULL  = 0,
      LOW_CULL,
      MEDIUM_LOW_CULL,
      MEDIUM_CULL,
      MEDIUM_HIGH_CULL,
      HIGH_CULL
   };

virtual void setCullAmountType(CullAmountType cullAmount);
```

Parameter:

```
CullAmountType cullAmount
```

cullAmount – The cull amount to set.

See CullAmountType Enumeration for more details.

Examples:

```
viewer->terrainLayer()->setCullAmountType(ossimPlanetTerrain::NO_CULL);

Result:
No Cull
```

```
viewer->terrainLayer()-
>setCullAmountType(ossimPlanetTerrain::LOW_CULL);

Result:
Low Cull
```

```
viewer->terrainLayer()-
>setCullAmountType(ossimPlanetTerrain::MEDIUM_LOW_CULL);

Result:
Medium Low Cull
```

```
viewer->terrainLayer()-
>setCullAmountType(ossimPlanetTerrain::MEDIUM_CULL);

Result:
Medium Cull
```

```
viewer->terrainLayer()-
>setCullAmountType(ossimPlanetTerrain::MEDIUM_HIGH_CULL);

Result:
Medium High Cull
```

```
viewer->terrainLayer()-
>setCullAmountType(ossimPlanetTerrain::HIGH_CULL);

Result:
High Cull
```

setTextureTileSize

Description:

Allows one to set the texture width and height to the planet. Increasing the texture size will increase the fidelity of the texture at greater distances. Note that higher textures sizes will consume much more memory. One can also go through setTextureDensityType to set the texture width and height using an enumeration.

Syntax:

```
virtual void setTextureTileSize(ossim_uint32 width, ossim_uint32
height);
```

Parameters:

```
ossim_uint32 width
```

width - The texture width value to set.

```
ossim_uint32 height
```

height - The texture height value to set.

Examples:

```
// Set the texture tile size to 1024 by 1024
terrain->setTextureTileSize(1024, 1024);
```

textureTileWidth

Description:

Returns the texture tile width.

Syntax:

```
ossim_uint32 textureTileWidth()const;
```

Examples:

```
std::cout << "textureTileWidth = " << terrain->textureTileWidth() <<
std::endl;

Result:
// LOW_TEXTURE_DENSITY:
textureTileWidth = 64

// MEDIUM_LOW_TEXTURE_DENSITY:
textureTileWidth = 128

// MEDIUM_TEXTURE_DENSITY:
textureTileWidth = 256

// MEDIUM_HIGH_TEXTURE_DENSITY:
textureTileWidth = 512
```

```
// HIGH_TEXTURE_DENSITY:
textureTileWidth = 1024
```

textureTileHeight

Description:

Returns the texture tile height.

Syntax:

```
ossim_uint32 textureTileHeight()const;
```

Examples:

```
std::cout << "textureTileHeight = " << terrain->textureTileHeight() <<
std::endl;

Result:
// LOW_TEXTURE_DENSITY:
textureTileHeight = 64

// MEDIUM_LOW_TEXTURE_DENSITY:
textureTileHeight = 128

// MEDIUM_TEXTURE_DENSITY:
textureTileHeight = 256

// MEDIUM_HIGH_TEXTURE_DENSITY:
textureTileHeight = 512

// HIGH_TEXTURE_DENSITY:
textureTileHeight = 1024
```

setElevationTileSize

Description:

Allows one to set the elevation width and height to the planet. Setting the elevation width and height adjusts the post spacing for each patch. Increasing the elevation size will increase the fidelity of the terrain at greater distances. Note that higher

elevation sizes will consume much more memory. For faster systems with greater ram you can increase the elevation density to get a better horizon. One can also go through setElevationDensityType to set the elevation width and height using an enumeration.

Syntax:

```
virtual void setElevationTileSize(ossim_uint32 width, ossim_uint32
height);
```

Parameters:

```
ossim_uint32 width
```

width - The elevation width value to set.

```
ossim_uint32 height
```

height - The elevation height value to set.

Example:

```
// Set the elevation tile size to 129 by 129
terrain->setElevationTileSize(129, 129);
```

elevationTileWidth

Description:

Returns the elevation tile width value.

```
ossim_uint32 elevationTileWidth()const;
```

Examples:

```
std::cout << "elevationTileWidth = " << terrain->elevationTileWidth()
<< std::endl;

Result:
// LOW_ELEVATION_DENSITY:
elevationTileWidth = 9
```

```
// MEDIUM_LOW_ELEVATION_DENSITY:
elevationTileWidth = 17

// MEDIUM_ELEVATION_DENSITY:
elevationTileWidth = 33

// MEDIUM_HIGH_ELEVATION_DENSITY:
elevationTileWidth = 65

// HIGH_ELEVATION_DENSITY:
elevationTileWidth = 129
```

elevationTileHeight

Description:

Returns the elevation tile width value.

```
ossim_uint32 elevationTileHeight()const;
```

Examples:

```
std::cout << "elevationTileHeight = " << terrain->elevationTileHeight()
<< std::endl;

Result:
// LOW_ELEVATION_DENSITY:
elevationTileHeight = 9

// MEDIUM_LOW_ELEVATION_DENSITY:
elevationTileHeight = 17

// MEDIUM_ELEVATION_DENSITY:
elevationTileHeight = 33

// MEDIUM_HIGH_ELEVATION_DENSITY:
elevationTileHeight = 65

// HIGH_ELEVATION_DENSITY:
elevationTileHeight = 129
```

setElevationDensityType

Description:

Allows one to set the elevation density type to the terrain.

Syntax:

```
enum ElevationDensityType
   {
      LOW_ELEVATION_DENSITY = 0,
      MEDIUM_LOW_ELEVATION_DENSITY,
      MEDIUM_ELEVATION_DENSITY,
      MEDIUM_HIGH_ELEVATION_DENSITY,
      HIGH_ELEVATION_DENSITY
   };

virtual void setElevationDensityType(ElevationDensityType type);
```

Parameter:

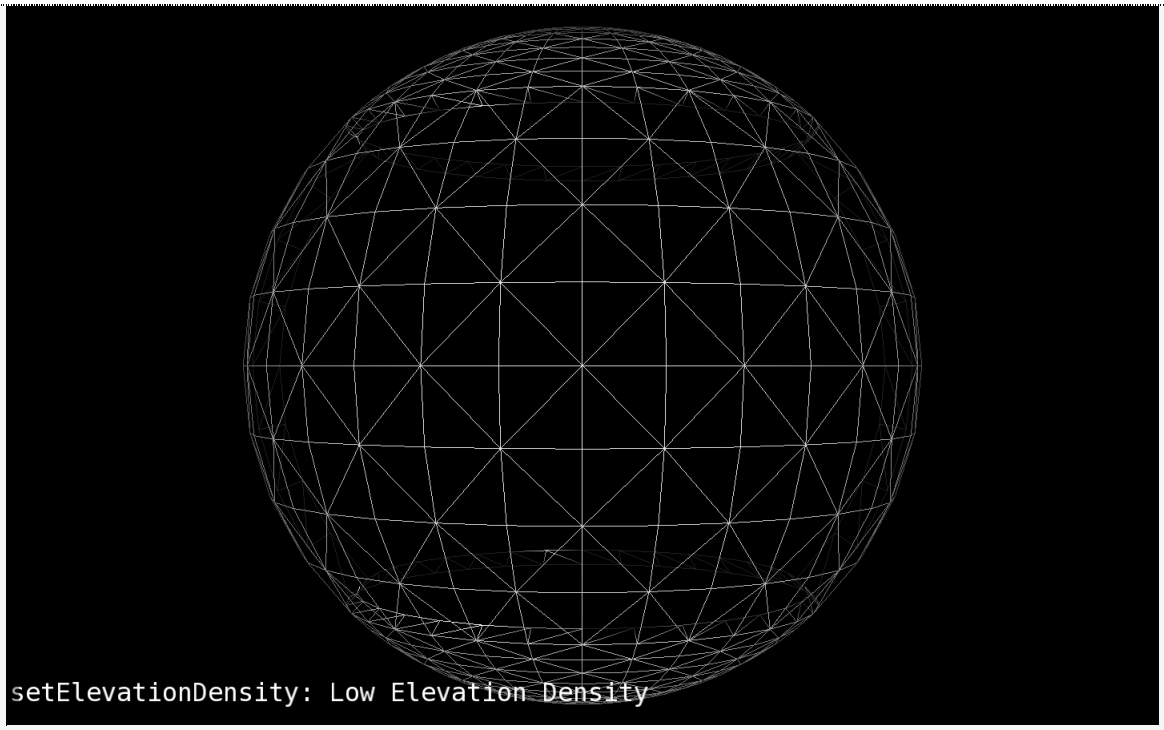
```
ElevationDensityType type
```

type – The elevation density type to set.

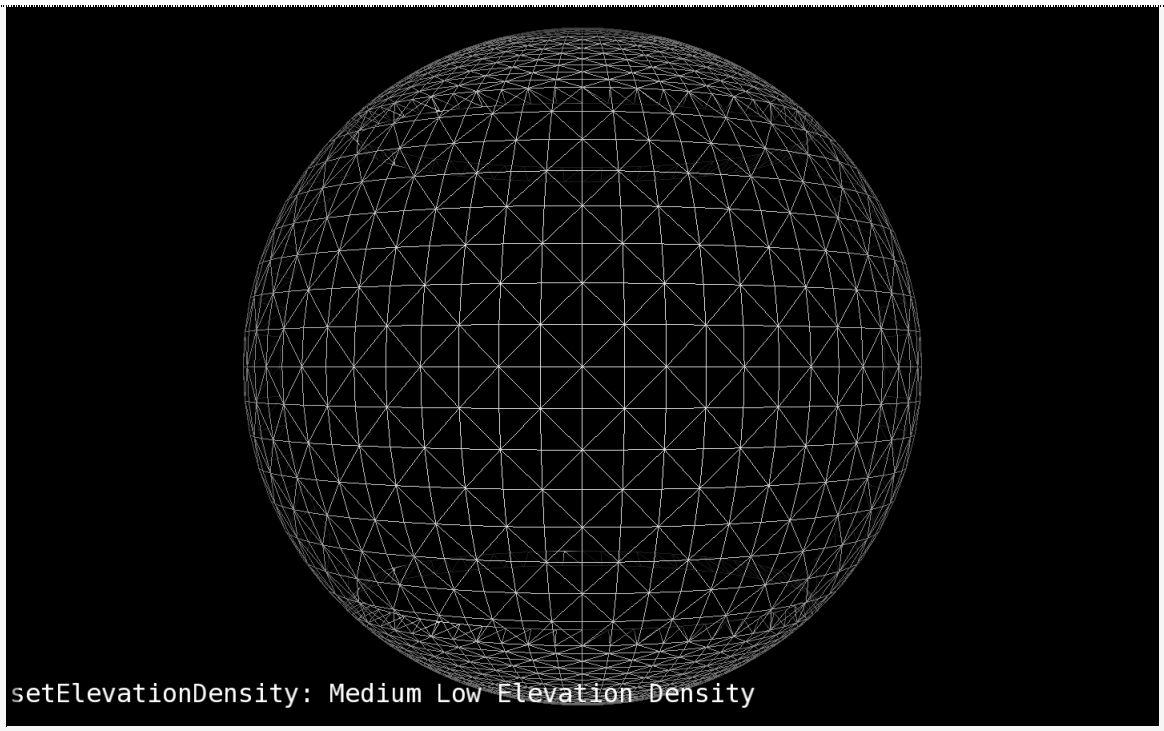
See ElevationDensityType enumeration for more details.

Examples:

```
viewer->terrainLayer()-
>setElevationDensityType(ossimPlanetTerrain::LOW_ELEVATION_DENSITY);

viewer->terrainLayer()->refreshElevationLayers();

Result:
Low Elevation Density
```

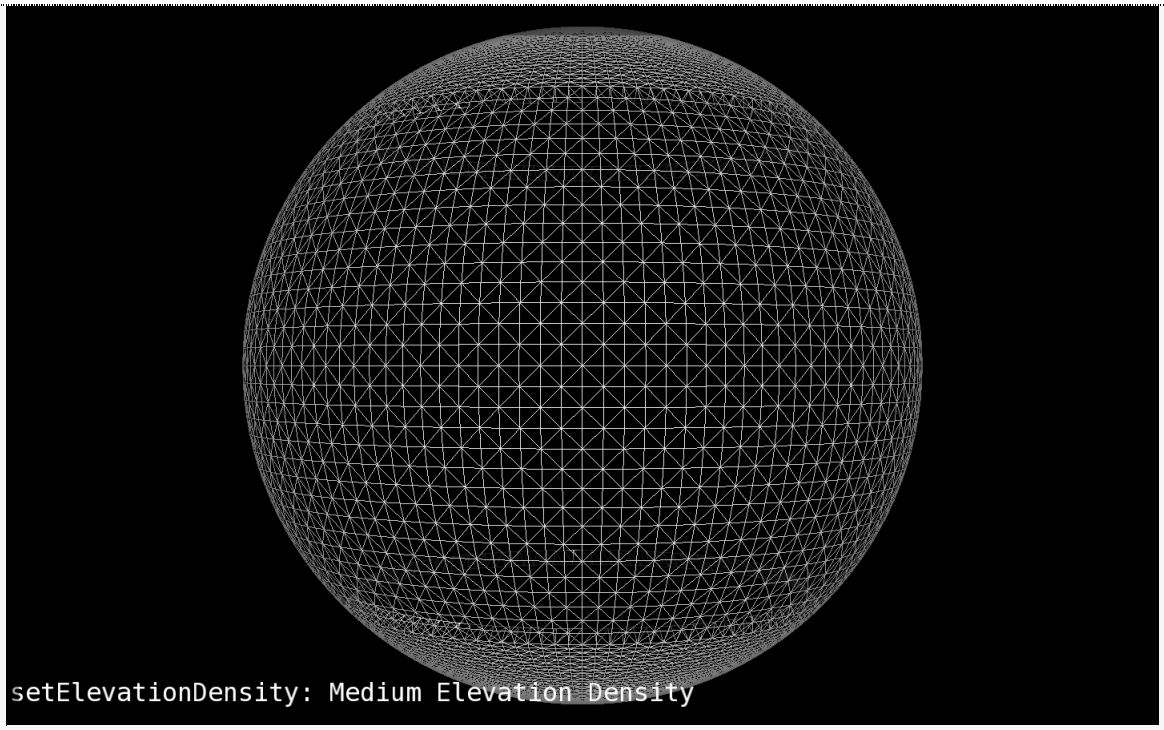



```
viewer->terrainLayer()-
>setElevationDensityType(ossimPlanetTerrain::MEDIUM_LOW_ELEVATION_DENSI
TY);

viewer->terrainLayer()->refreshElevationLayers();

Result:
Medium Low Elevation Density
```

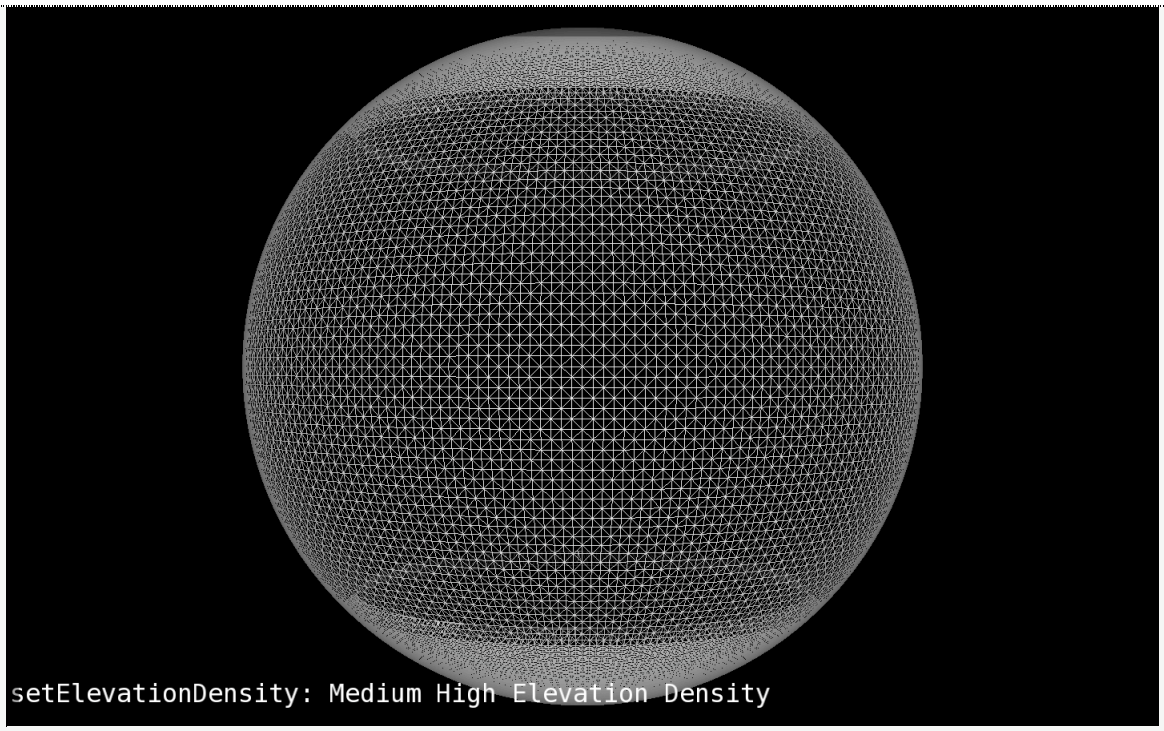



```
viewer->terrainLayer()-
>setElevationDensityType(ossimPlanetTerrain::MEDIUM_ELEVATION_DENSITY);

viewer->terrainLayer()->refreshElevationLayers();
```

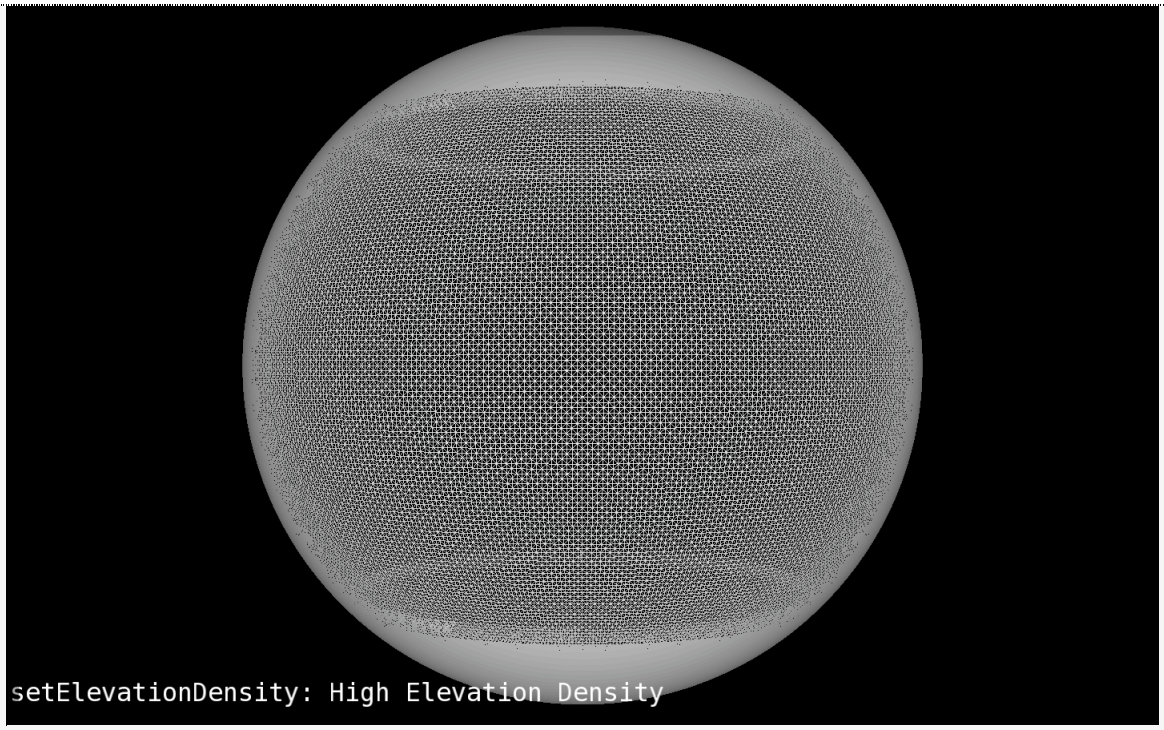
Result:
Medium Elevation Density




```
viewer->terrainLayer()-
>setElevationDensityType(ossimPlanetTerrain::MEDIUM_HIGH_ELEVATION_DENS
ITY);

viewer->terrainLayer()->refreshElevationLayers();

Result:
Medium High Elevation Density
```




```
viewer->terrainLayer()-
>setElevationDensityType(ossimPlanetTerrain::HIGH_ELEVATION_DENSITY);

viewer->terrainLayer()->refreshElevationLayers();

Result:
High Elevation Density
```



setTextureDensityType

Description:

Allows one to set the texture density type to the terrain.

Syntax:

```
enum TextureDensityType
   {
      LOW_TEXTURE_DENSITY = 0,
      MEDIUM_LOW_TEXTURE_DENSITY,
      MEDIUM_TEXTURE_DENSITY,
      MEDIUM_HIGH_TEXTURE_DENSITY,
      HIGH_TEXTURE_DENSITY
   };

virtual void setTextureDensityType(TextureDensityType type);
```

Parameter:

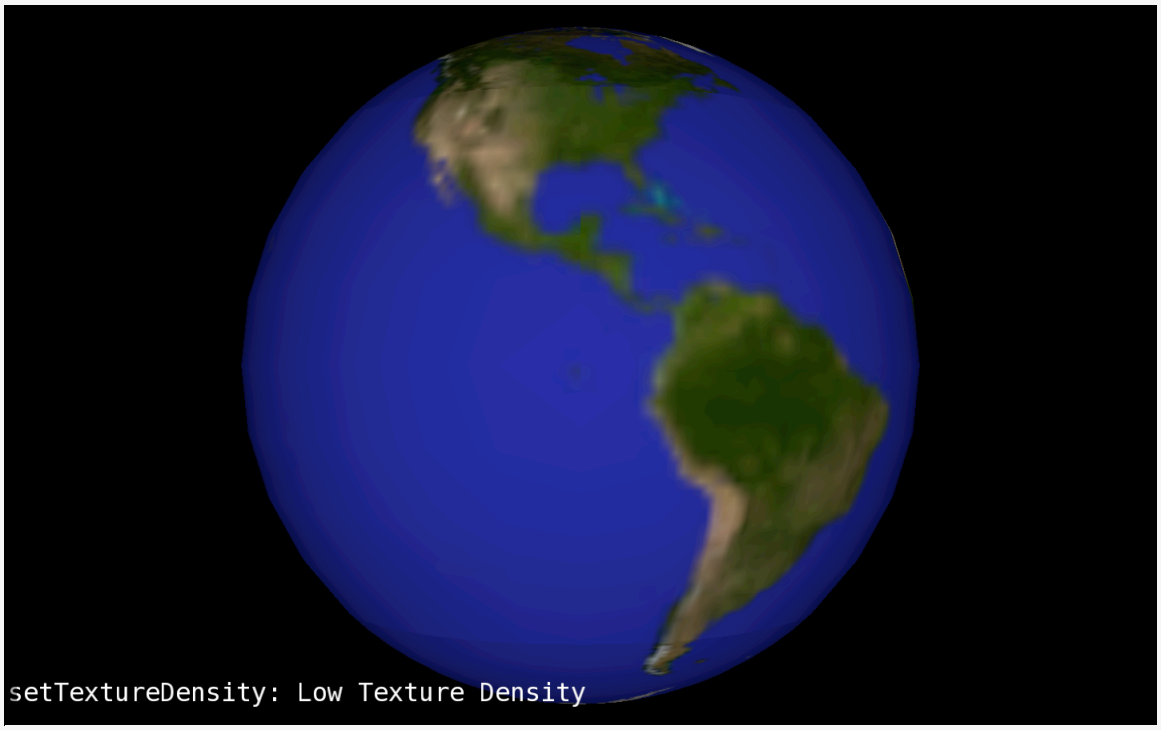
```
TextureDensityType type
```

type – The texture density type to set.

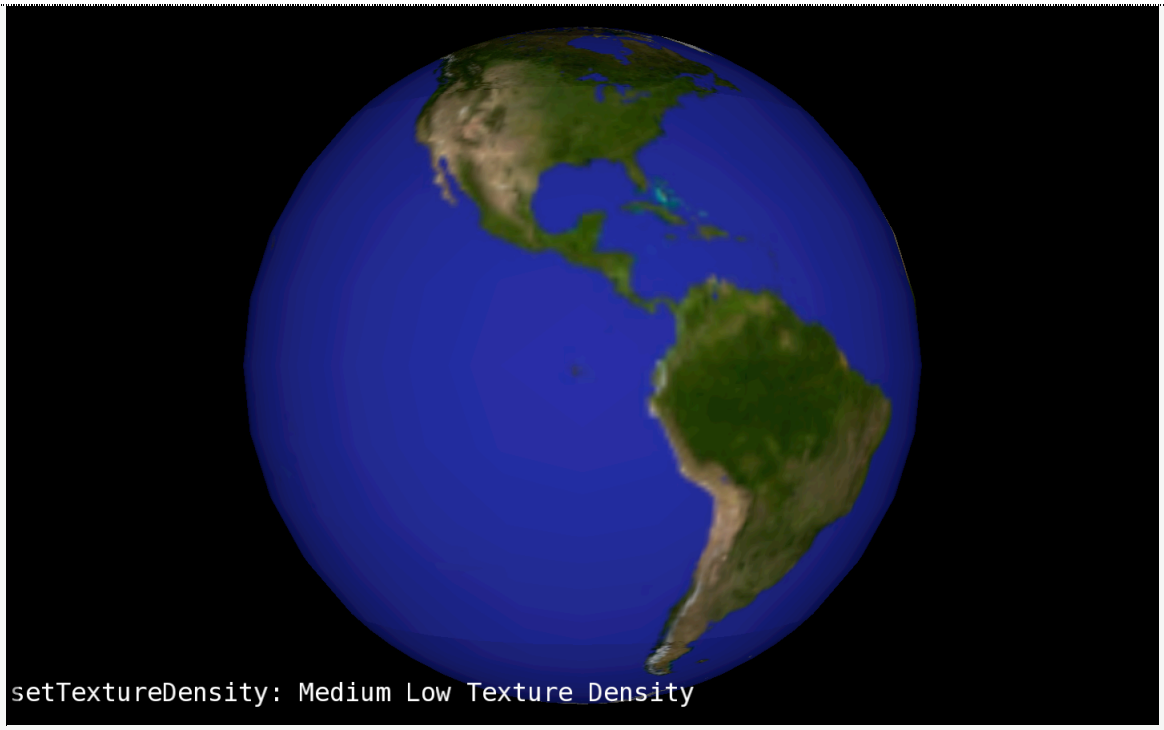
See TextureDensityType enumeration for more details.

Examples:

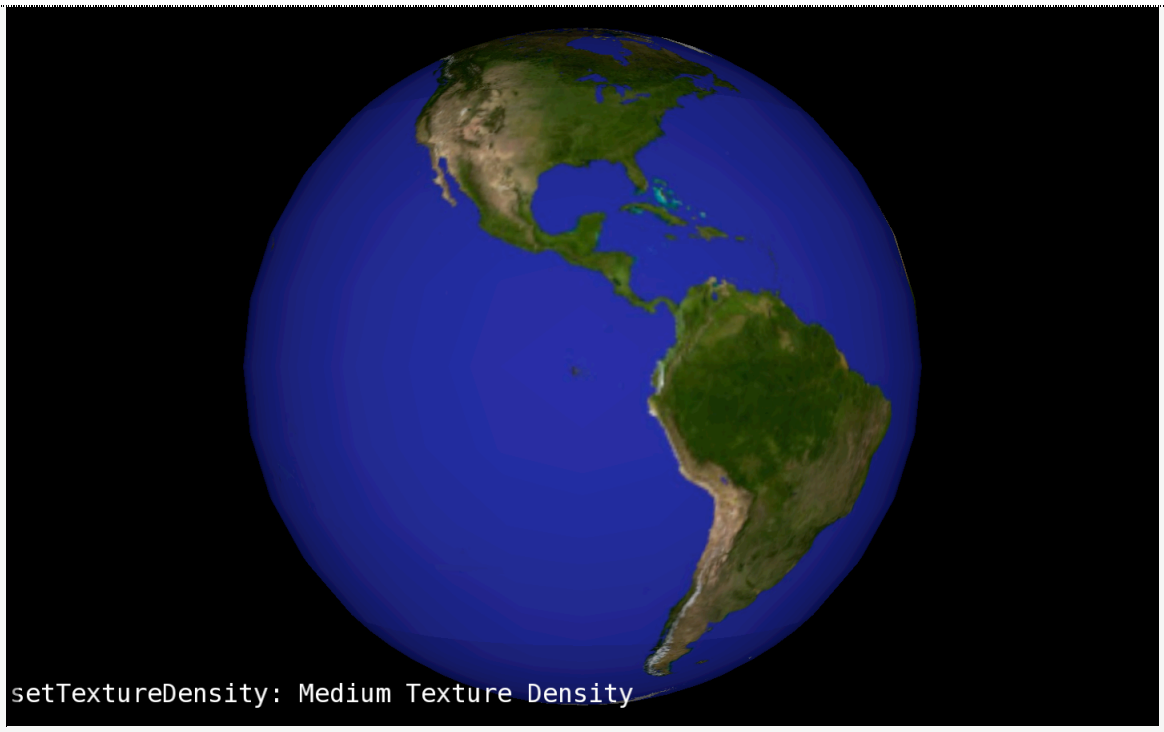
```
viewer->terrainLayer()-
>setTextureDensityType(ossimPlanetTerrain::LOW_TEXTURE_DENSITY);

viewer->terrainLayer()->refreshImageLayers();

Result:
Low Texture Density
```




```
viewer->terrainLayer()-
>setTextureDensityType(ossimPlanetTerrain::MEDIUM_LOW_TEXTURE_DENSITY);

viewer->terrainLayer()->refreshImageLayers();

Result:
Medium Low Texture Density
```

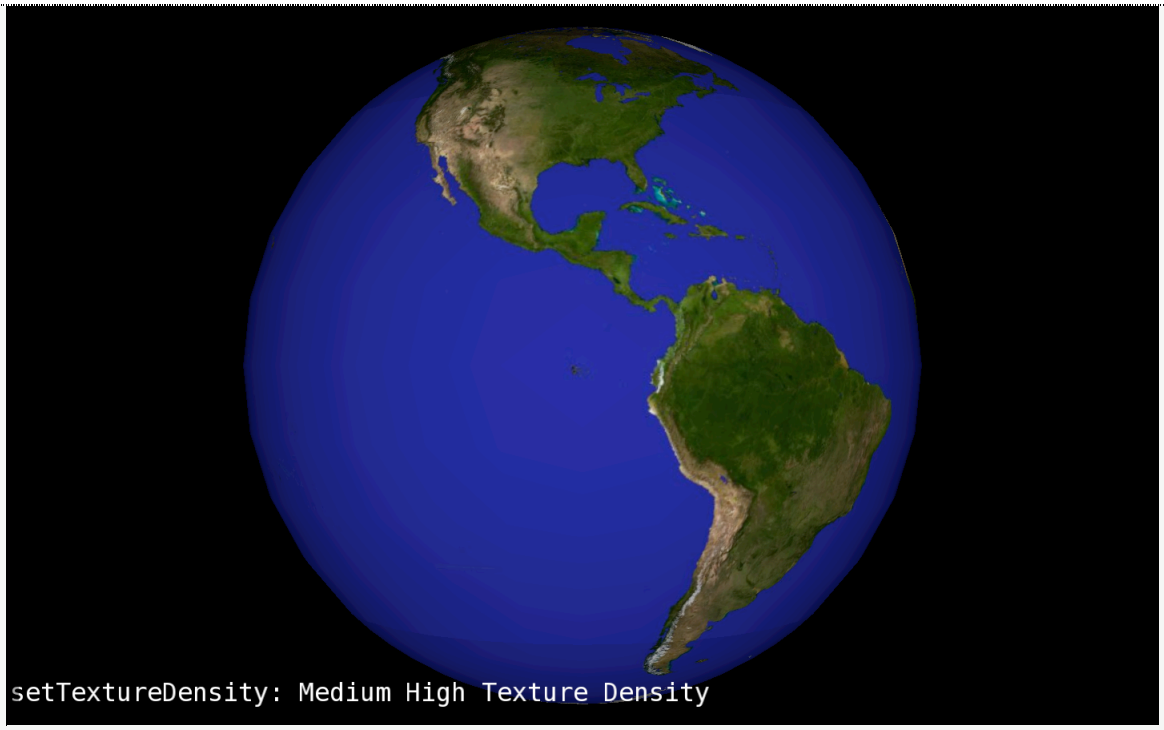



```
viewer->terrainLayer()-
>setTextureDensityType(ossimPlanetTerrain::MEDIUM_TEXTURE_DENSITY);

viewer->terrainLayer()->refreshImageLayers();

Result:
Medium Texture Density
```

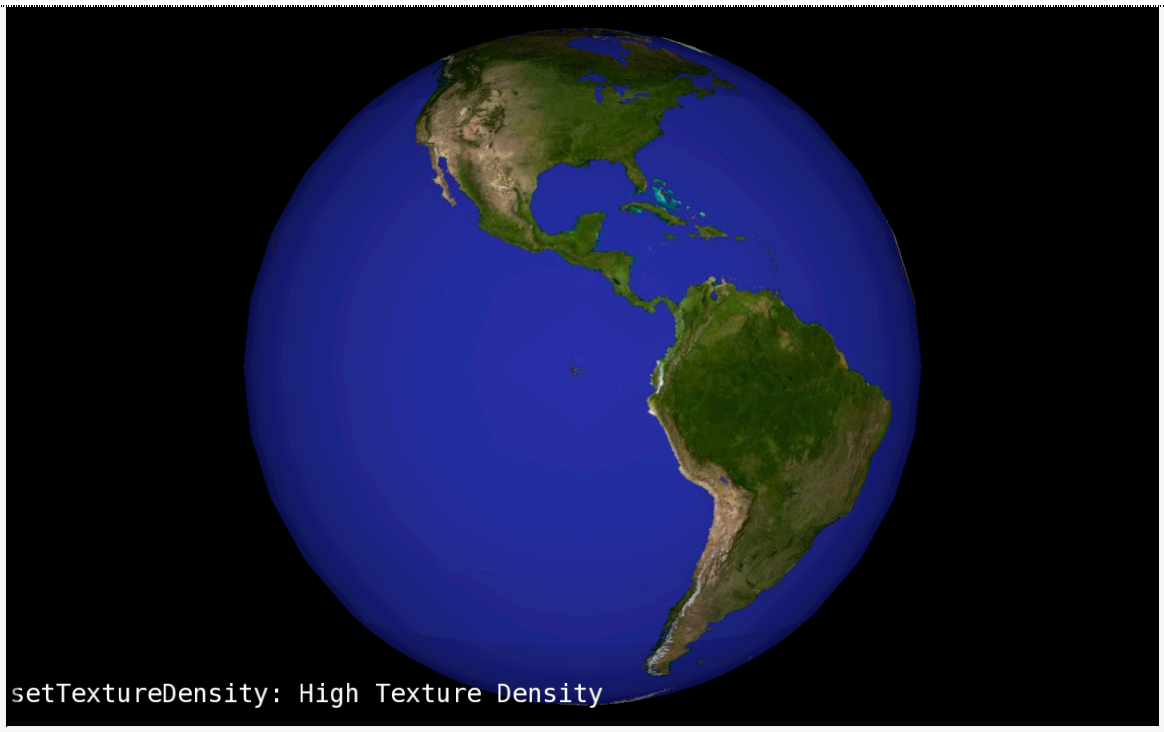



```
viewer->terrainLayer()-
>setTextureDensityType(ossimPlanetTerrain::MEDIUM_HIGH_TEXTURE_DENSITY)
;

viewer->terrainLayer()->refreshImageLayers();
```

Result:
Medium High Texture Density




```
viewer->terrainLayer()-
>setTextureDensityType(ossimPlanetTerrain::HIGH_TEXTURE_DENSITY);

viewer->terrainLayer()->refreshImageLayers();

Result:
High Texture Density
```



setSplitMergeSpeedType

Description:

Split merge speed determines how fast or basically how soon the patch should split. Currently our algorithm is based solely on distance, which means if you set the split merge speed type to a LOW_SPEED setting you have to move your eye closer to a patch before it is further refined. If you set the speed to a higher speed that low it progressively refines the graph sooner. So the eye can be further away and still refine without having to be close so the higher the speed the more detail you see from further away.

Syntax:

```
enum SplitMergeSpeedType
   {
      LOW_SPEED = 0,
      MEDIUM_LOW_SPEED,
      MEDIUM_SPEED,
      MEDIUM_HIGH_SPEED,
      HIGH_SPEED
   };
```

```
virtual void setSplitMergeSpeedType(SplitMergeSpeedType type);
```

Parameter:

```
SplitMergeSpeedType type
```

type – The split merge speed type to set.

See SplitMergeSpeedType enumeration for more details.

Examples:

```
viewer->terrainLayer()-
>setSplitMergeSpeedType(ossimPlanetTerrain::LOW_SPEED);

Result:
Low Speed
```

```
viewer->terrainLayer()-
>setSplitMergeSpeedType(ossimPlanetTerrain::MEDIUM_LOW_SPEED);

Result:
Medium Low Speed
```

```
viewer->terrainLayer()-
>setSplitMergeSpeedType(ossimPlanetTerrain::MEDIUM_SPEED);

Result:
Medium Speed
```

```
viewer->terrainLayer()-
>setSplitMergeSpeedType(ossimPlanetTerrain::MEDIUM_HIGH_SPEED);

Result:
Medium High Speed
```

```
viewer->terrainLayer()-
>setSplitMergeSpeedType(ossimPlanetTerrain::HIGH_SPEED);

Result:
High Speed
```

setElevationExageration

Description:

Allows one to set elevation exaggeration to the planet. This is a multiplicative factor applied to the elevation values. Zero would result in a flat surface, 1.0 would reflect actual elevation value, above 1.0 would exaggerate the elevation.

Syntax:

```
virtual void setElevationExaggeration(ossim_float64 exaggeration);
```

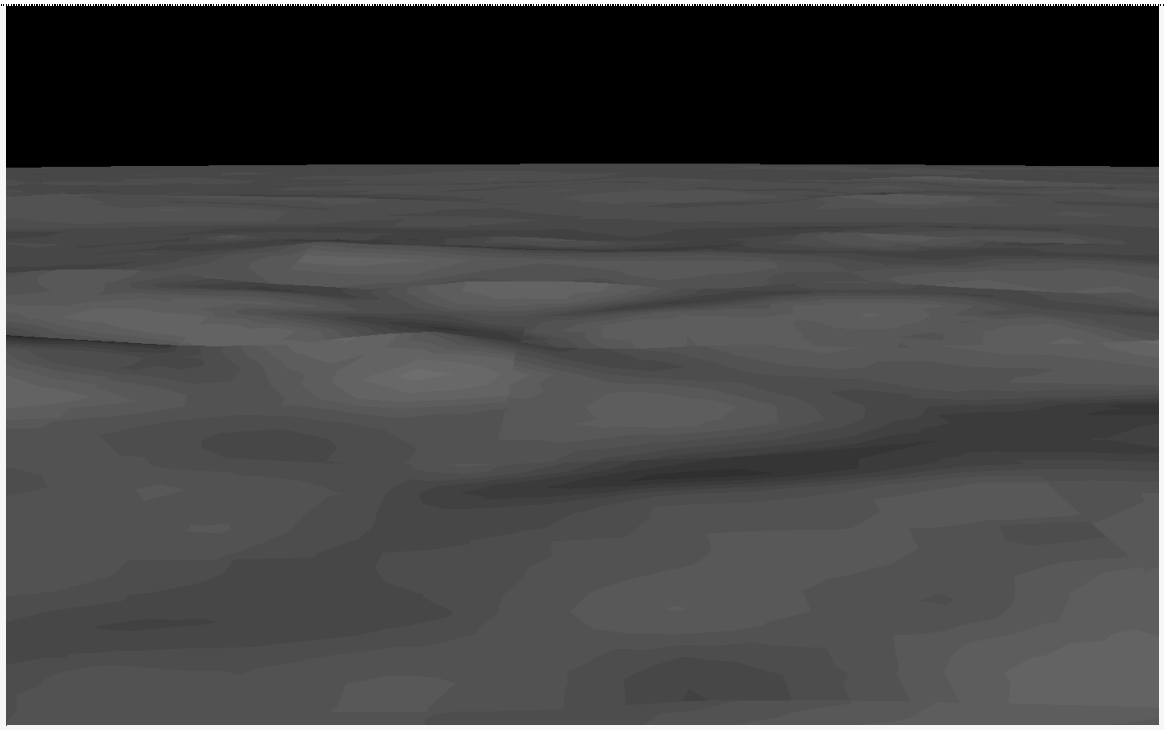
Parameter:

```
ossim_float64 exaggeration
```

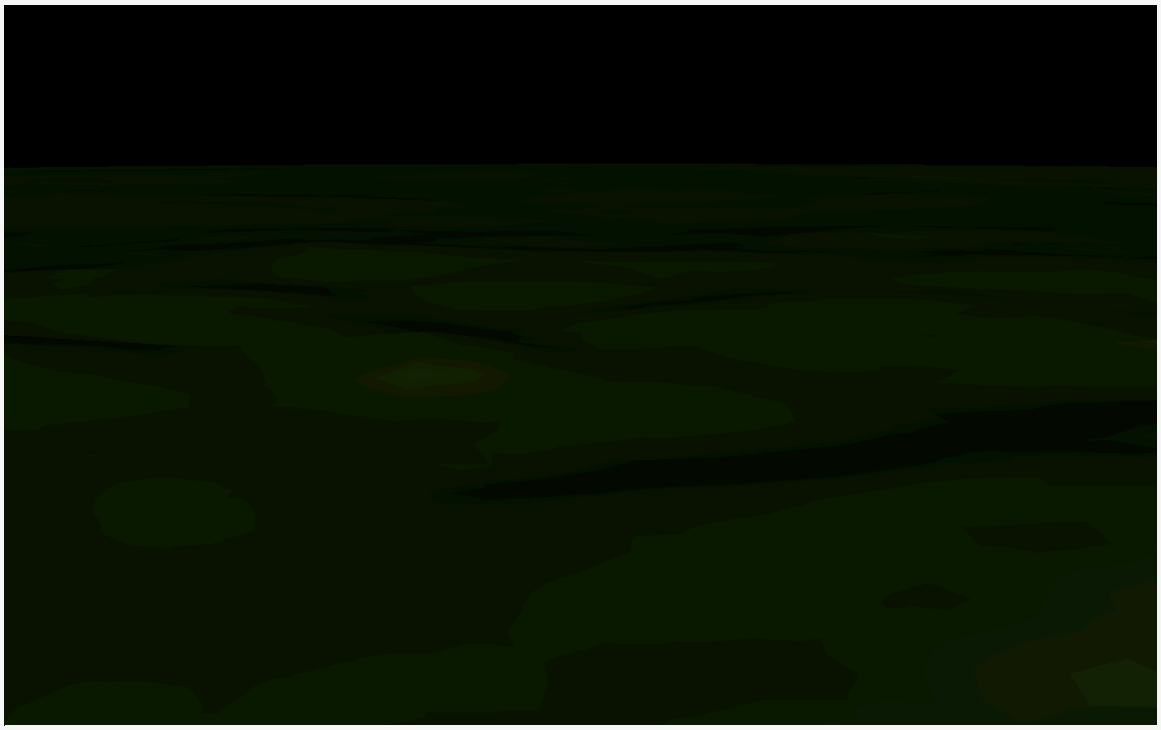
exaggeration - The exaggeration to be set.

Examples:

```
terrain->setElevationExageration(.5);
viewer->terrainLayer()->refreshElevationLayers();

Result:
.5x

Without textures.
```



With textures.

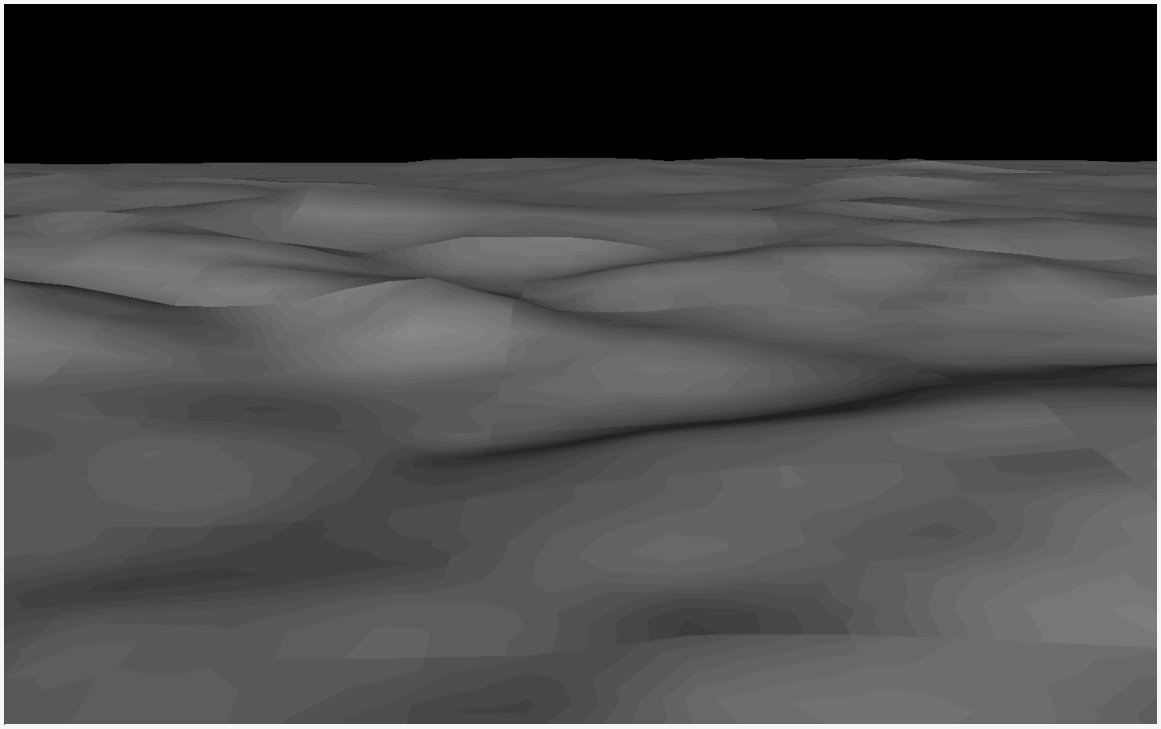


```
terrain->setElevationExageration(1);
```

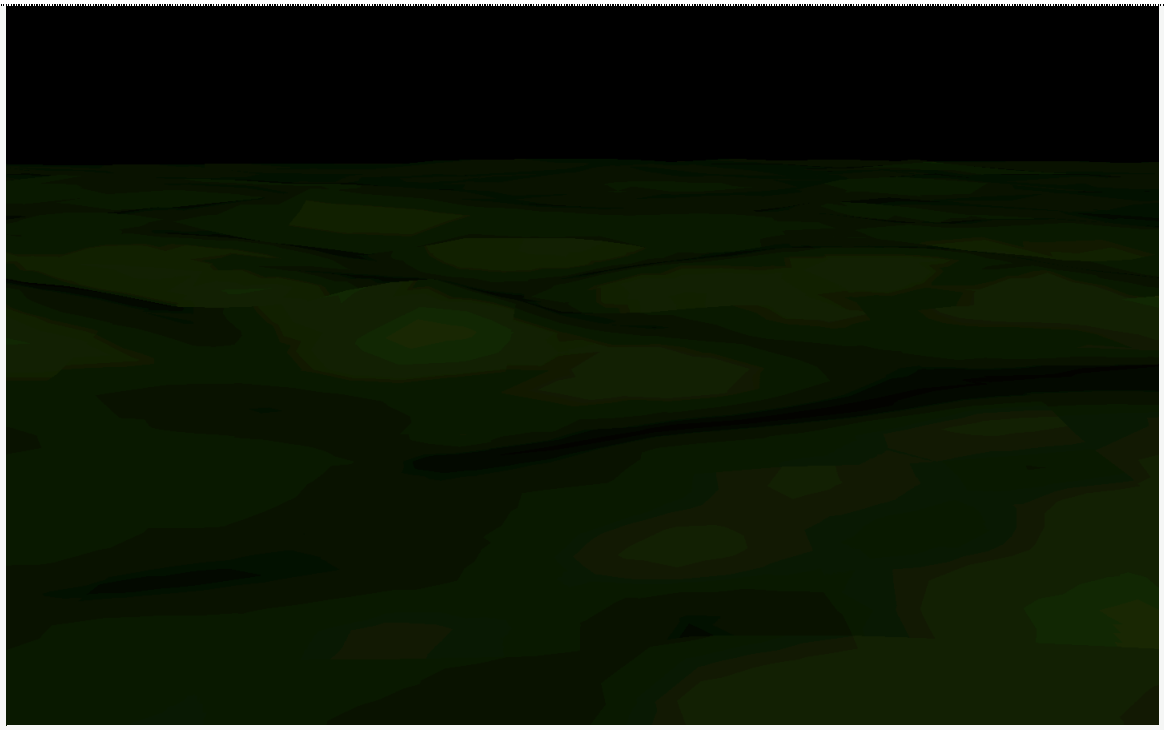
```
viewer->terrainLayer()->refreshElevationLayers();
```

Result:
1x

Without textures.



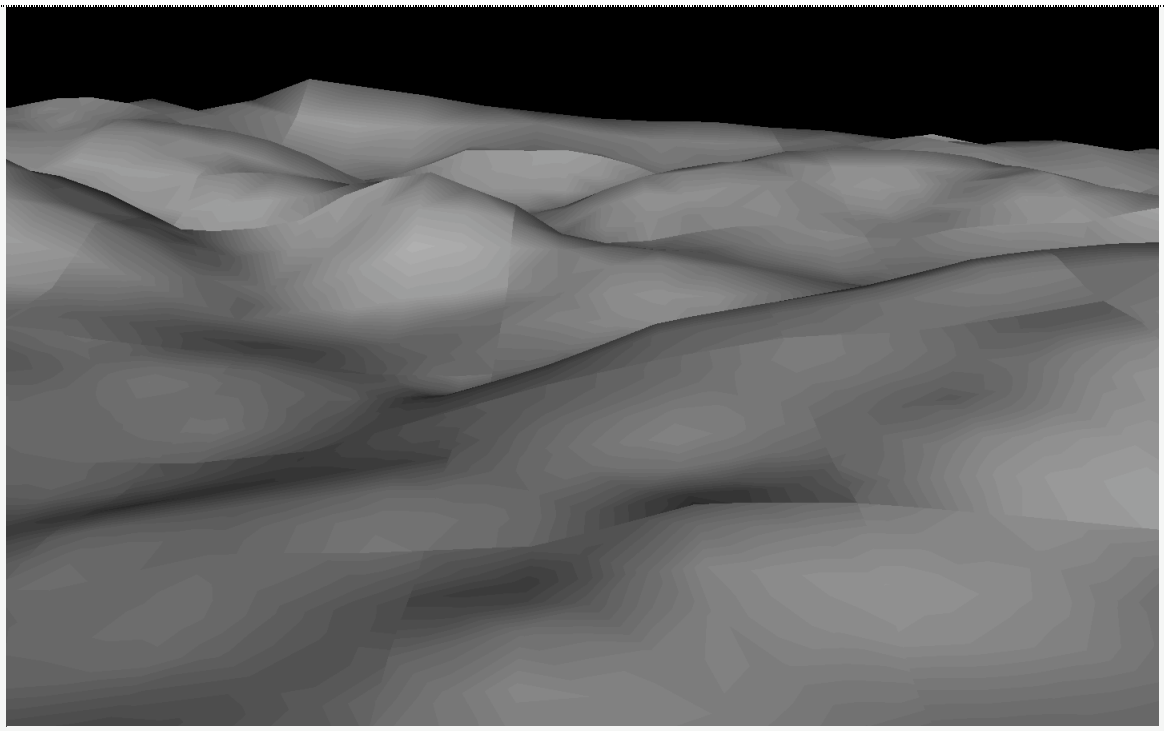
With textures.



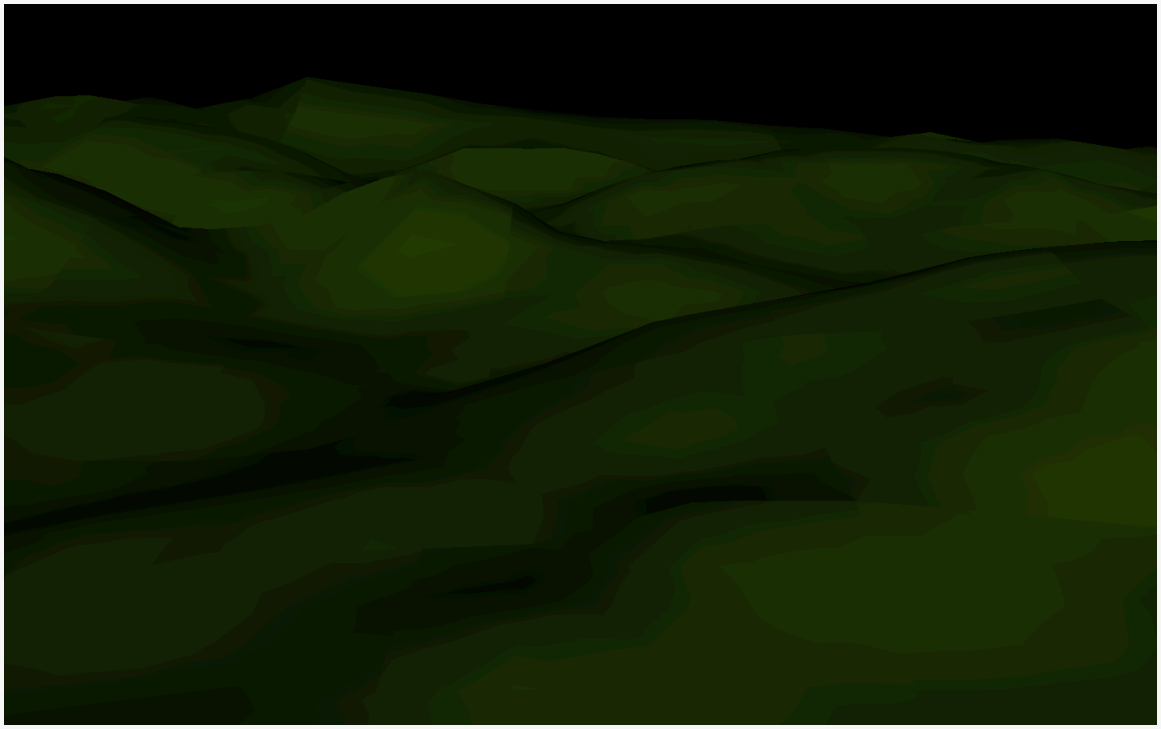
```
terrain->setElevationExageration(2);
viewer->terrainLayer()->refreshElevationLayers();
```

Result:
2x

Without textures.



With textures.



elevationExaggeration

Description:

Returns the elevation exaggeration factor.

Syntax:

```
ossim_float64 elevationExaggeration()const;
```

Examples:

```
viewer->terrainLayer()->setElevationExaggeration(.5);
viewer->terrainLayer()->refreshElevationLayers();

std::cout << "elevationExaggeration: " << viewer->terrainLayer()-
>elevationExaggeration() << std::endl;

Result:
elevationExaggeration: .5
```

```
viewer->terrainLayer()->setElevationExaggeration(1);
viewer->terrainLayer()->refreshElevationLayers();

std::cout << "elevationExaggeration: " << viewer->terrainLayer()-
>elevationExaggeration() << std::endl;

Result:
elevationExaggeration: 1
```

```
viewer->terrainLayer()->setElevationExaggeration(2);
viewer->terrainLayer()->refreshElevationLayers();

std::cout << "elevationExaggeration: " << viewer->terrainLayer()-
>elevationExaggeration() << std::endl;

Result:
elevationExaggeration: 2
```

initElevation

Description:

Initializes elevation in the terrain layer.

If you do not add elevation explicitly with the addElevation methods then one can call initElevation and it will use all registered elevation databases from the ossimCore engine to initialize and setup the elevation in the planet.

It is important to note that at this time any elevation added through ossimPlanet is not added to the ossimCore engine. If there is any sensor modeling or a need to do elevation lookups then the values will not be present for these types of operations.

Syntax:

```
void initElevation();
```

Example:

```
terrain->initElevation();
```

ossimPlanetGrid.h

```
#include <ossimPlanet/ossimPlanetGrid.h>
```

The grid class is located in *ossimPlanet/include/ossimPlanet/ossimPlanetGrid.h*.

Description:

The ossimPlanetGrid class allows one access to set the polar cap locations.

Public Member Functions

setCapLocation

Description:

Allows one to set the cap location. The default cap location is 45 degrees latitude (LOW_CAP).

Syntax:

```
enum CapLocation
   {
      LOW_CAP = 0, // 45 degrees latitude
      MEDIUM_LOW_CAP, // 67.5 degrees latitude
      MEDIUM_CAP, // 78.75 degrees latitude
      MEDIUM_HIGH_CAP, // 84.375 degrees latitude
      HIGH_CAP // 87.1875 degrees latitude
   };

virtual void setCapLocation(CapLocation location);
```

Parameter:

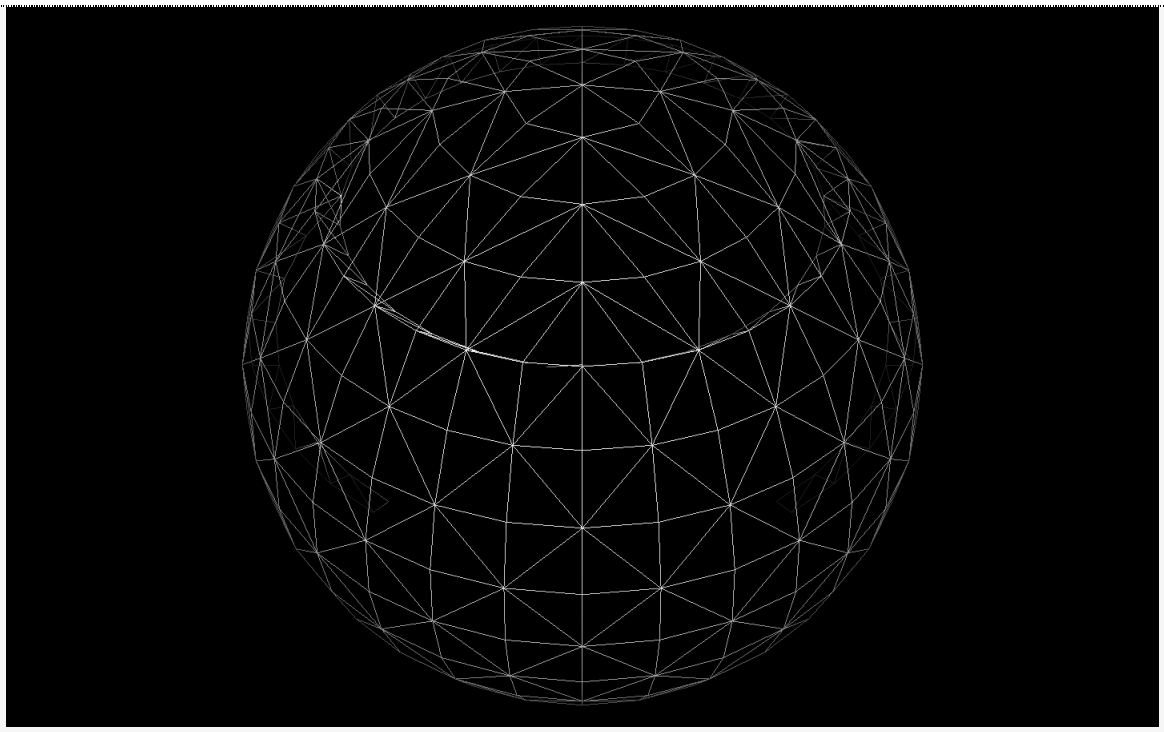
```
CapLocation location
```

location - The cap location to set.

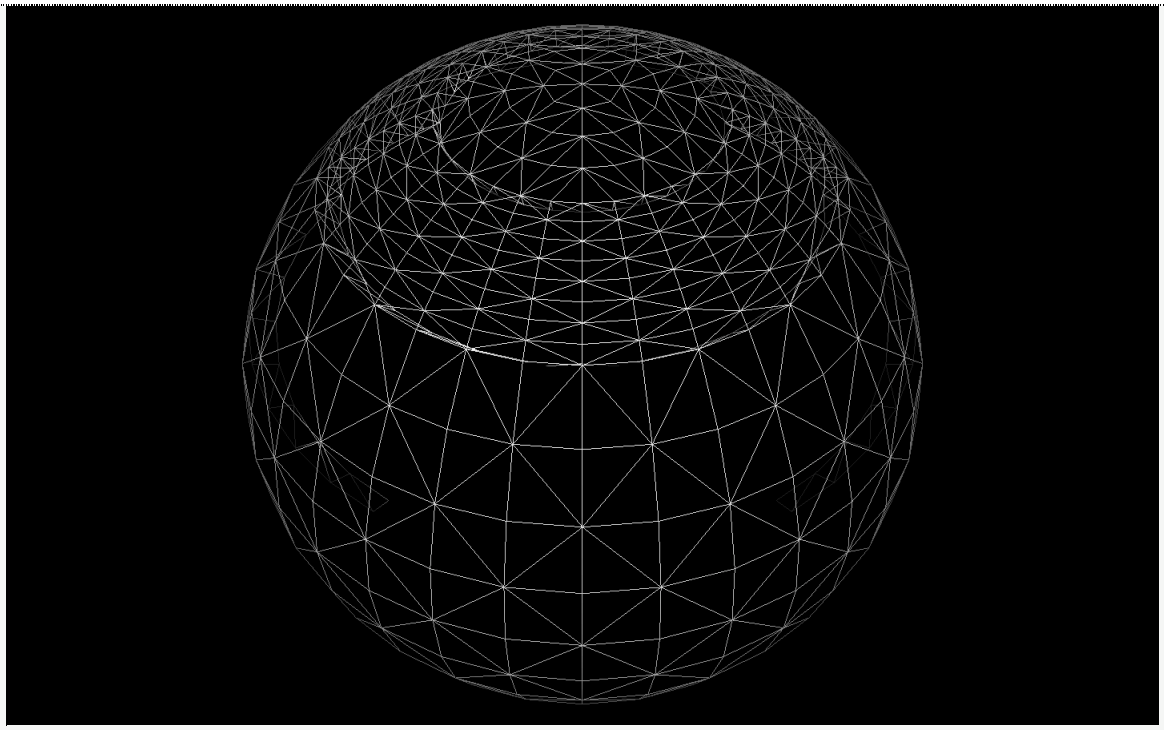
See CapLocation enumeration for more details.

Examples:

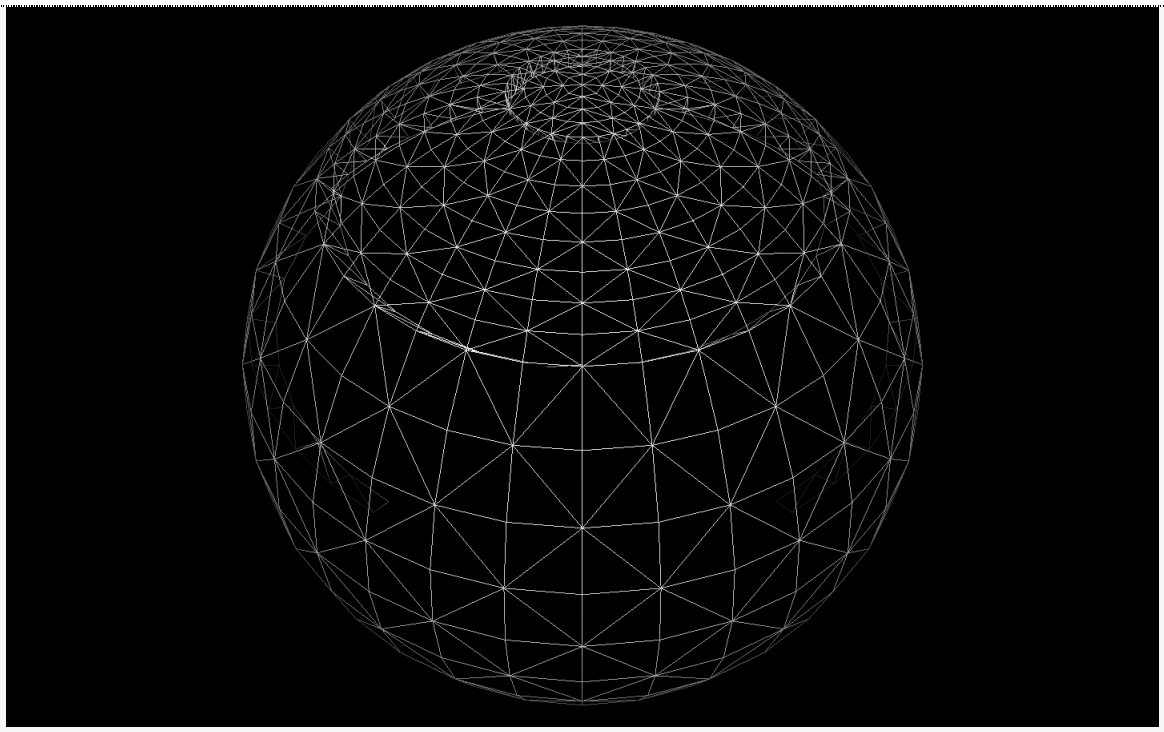
```
ossimPlanetAdjustableCubeGrid* grid = new
ossimPlanetAdjustableCubeGrid(ossimPlanetAdjustableCubeGrid::LOW_CAP);

viewer->terrainLayer()->setGrid(grid);

Result:
Low Cap Location (Default Cap Location)
45 Degrees Latitude
```



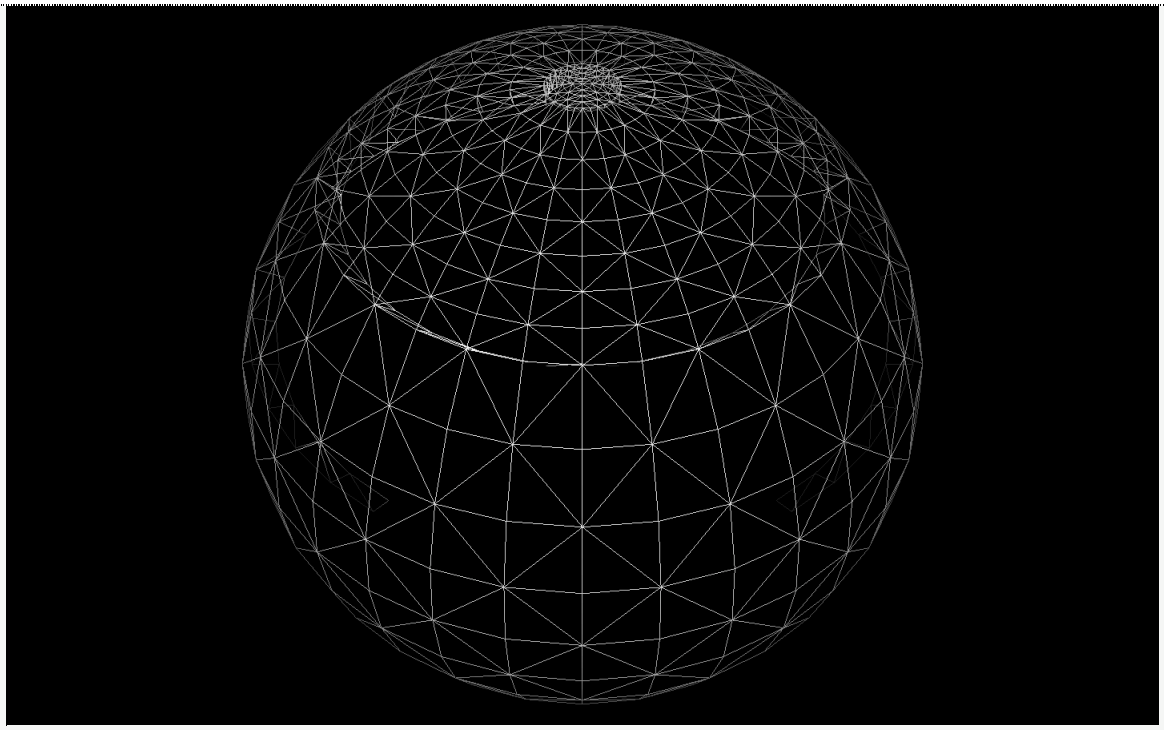
```
ossimPlanetAdjustableCubeGrid* grid = new
ossimPlanetAdjustableCubeGrid(ossimPlanetAdjustableCubeGrid::MEDIUM_LOW
_CAP);

viewer->terrainLayer()->setGrid(grid);

Result:
Medium Low Cap Location
67.5 Degrees Latitude
```



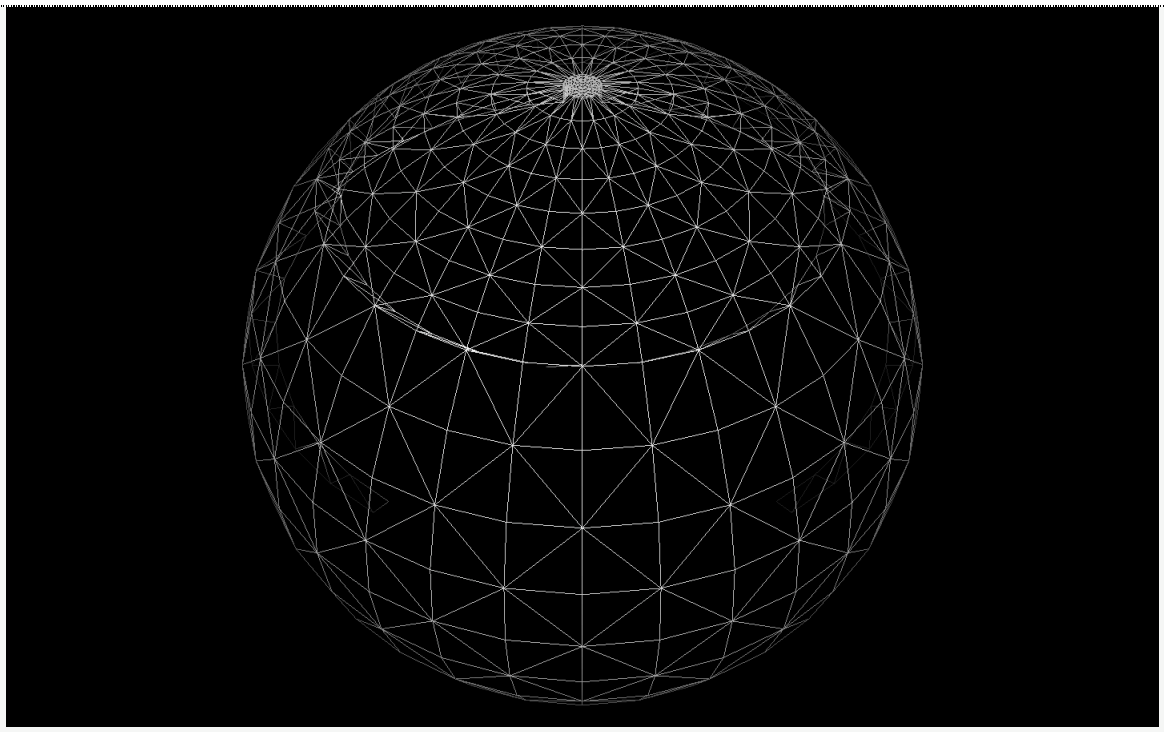
```
ossimPlanetAdjustableCubeGrid* grid = new
ossimPlanetAdjustableCubeGrid(ossimPlanetAdjustableCubeGrid::MEDIUM_CAP
);

viewer->terrainLayer()->setGrid(grid);

Result:
Medium Cap Location
78.75 Degrees Latitude
```



```
ossimPlanetAdjustableCubeGrid* grid = new
ossimPlanetAdjustableCubeGrid(ossimPlanetAdjustableCubeGrid::MEDIUM_HIG
H_CAP);

viewer->terrainLayer()->setGrid(grid);

Result:
Medium High Cap Location
84.375 Degrees Latitude
```



```
ossimPlanetAdjustableCubeGrid* grid = new
ossimPlanetAdjustableCubeGrid(ossimPlanetAdjustableCubeGrid::HIGH_CAP);

viewer->terrainLayer()->setGrid(grid);

Result:
High Cap Location
87.1875 Degrees Latitude
```



ossimPlanetCloudLayer.h

```
#include <ossimPlanet/ossimPlanetCloudLayer.h>
```

The planet cloud layer class is located in *ossimPlanet/include/ossimPlanet/ossimPlanetCloudLayer.h*.

Description:

The ossimPlanetCloudLayer class allows one access to control the cloud model.

Public Member Functions:

updateTexture

Syntax:

```
void updateTexture(osg::Image* cloudTexture);
```

Parameter:

```
osg::Image* cloudTexture
```

updateTexture

Syntax:

```
void updateTexture(ossim_int64 seed = 0,
                   ossim_int32 coverage = 20,
                   ossim_float64 sharpness = .95);
```

Parameters:

```
ossim_int64 seed = 0,
```

```
ossim_int32 coverage = 20
```

```
ossim_float64 sharpness = .95
```

Example:

```
ossim_int32 cloudCoverage = 20;
ossim_float64 cloudSharpness = .95;

cloud->updateTexture(time(0), cloudCoverage, cloudSharpness);
```

computeMesh

Syntax:

```
void computeMesh(double patchAltitude,
                 ossim_uint32 patchWidth=9,
                 ossim_uint32 patchHeight=9,
                 ossim_uint32 level=3);
```

Parameters:

```
double patchAltitude
```

```
ossim_uint32 patchWidth=9
```

```
ossim_uint32 patchHeight=9
```

```
ossim_uint32 level=3
```

Example:

```
ossim_float64 cloudAltitude = 20000;

cloud->computeMesh(cloudAltitude, 32,32,1);
```

setHeading

Description:

Allows one to set the heading of the clouds in degrees.

Syntax:

```
void setHeading(ossim_float64 heading)
   {
      theHeading = heading;
   }
```

Parameter:

```
ossim_float64 heading
```

heading - The heading to set in degrees.

setSpeedPerHour

Description:

The speed is in units per hour.

Syntax:

```
void setSpeedPerHour(ossim_float64 speed, ossimUnitType
unit=OSSIM_METERS)
   {
      // get it into units per second.
      //
      theSpeed = ossimUnitConversionTool(speed,
unit).getMeters()/3600.0;
   }
```

Parameters:

```
ossim_float64 speed
```

speed - The speed to set in units per hour.

```
ossimUnitType unit=OSSIM_METERS
```

Example:

```
cloud->setSpeedPerHour(60.0, OSSIM_MILES);
```

setSpeedPerSecond

Description:

The speed is in units per seconds.

Syntax:

```
void setSpeedPerSecond(ossim_float64 speed, ossimUnitType
unit=OSSIM_METERS)
   {
      // get it into units per second.
      //
      theSpeed = ossimUnitConversionTool(speed, unit).getMeters();
   }
```

Parameters:

```
ossim_float64 speed
```
speed - The speed to set in units per second.

```
ossimUnitType unit=OSSIM_METERS
```

setScale

Description:

Allows one to set the cloud texture scale.

Syntax:

```
void setScale(ossim_float64 scale)
   {
      theTextureScale[0] = scale;
      theTextureScale[1] = scale;
      theTextureScale[2] = 1.0;
      updateMetersPerPixelCoverage();
   }
```

Parameter:

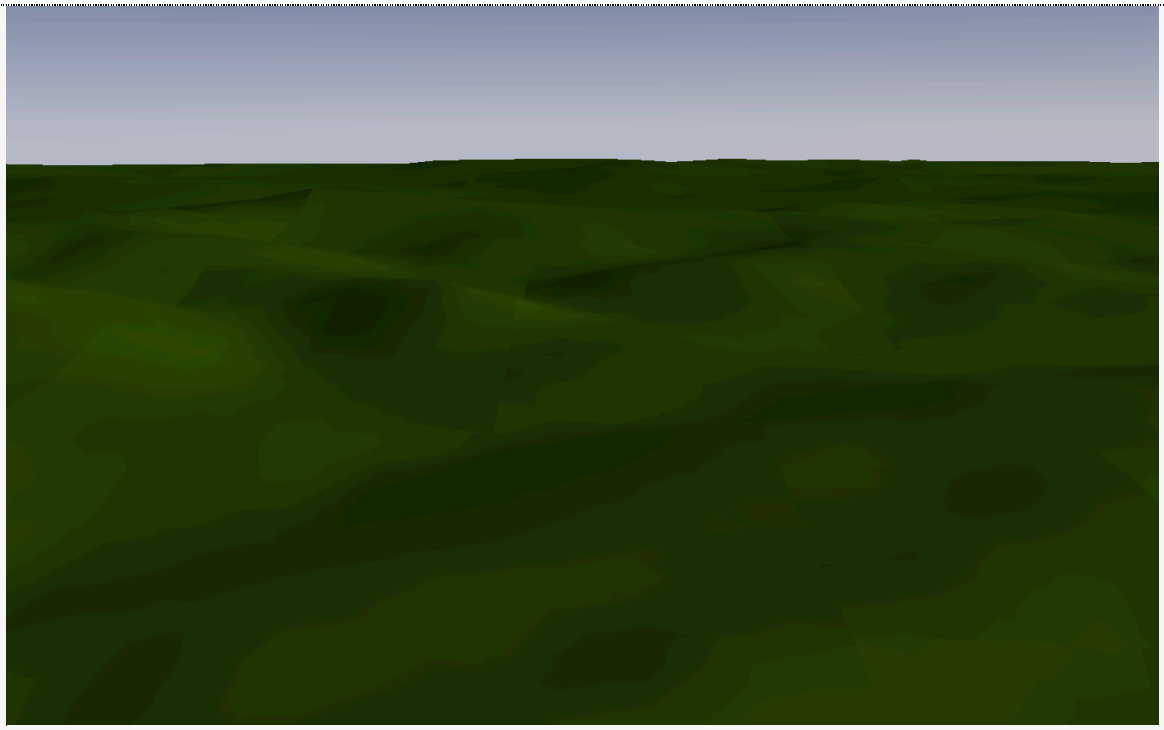
```
ossim_float64 scale
```
scale - The cloud texture scale to set.

Example:

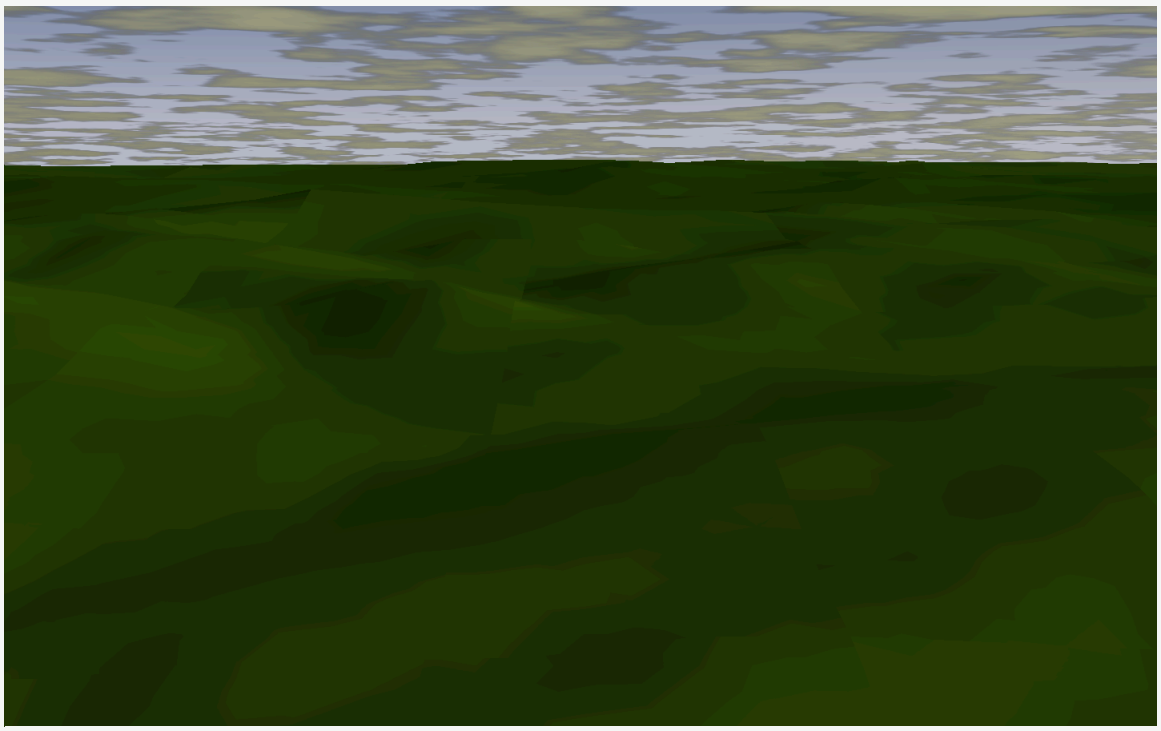
```
cloud->setScale(4);
```

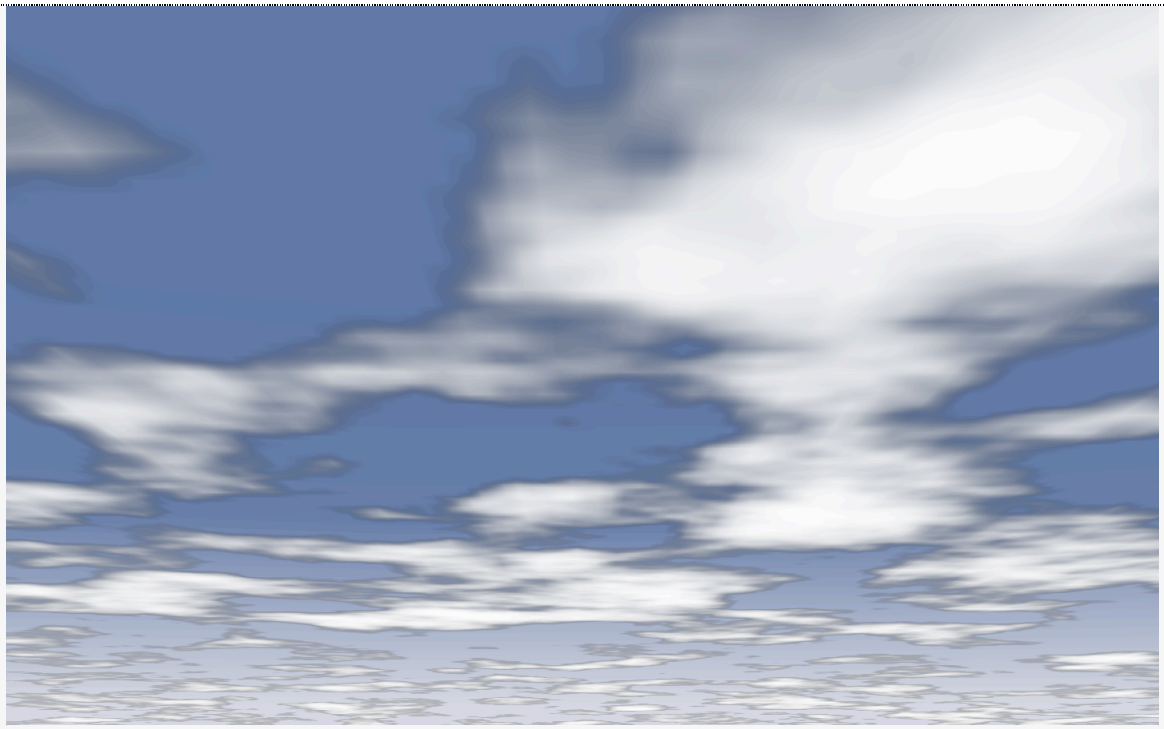

Chapter Summary

Ephemeris model with no clouds.



Ephemeris model with clouds.





```
// Create a new cloud object.
osg::ref_ptr<ossimPlanetCloudLayer> cloud = new ossimPlanetCloudLayer;

// Add cloud to planet.
viewer->planet()->addChild(cloud.get());

// Update the texture.
ossim_int32 cloudCoverage = 20;
ossim_float64 cloudSharpness = .95;
cloud->updateTexture(time(0), cloudCoverage, cloudSharpness);

// Compute the mesh.
ossim_float64 cloudAltitude = 20000;
cloud->computeMesh(cloudAltitude, 32,32,1);

// Set the speed per hour.
cloud->setSpeedPerHour(60.0, OSSIM_MILES);

// Set the scale.
cloud->setScale(4);

Refer to the ossimPlanetEphemeris class for more information.
```

ossimPlanetEphemeris.h

```
#include <ossimPlanet/ossimPlanetEphemeris.h>
```

The ephemeris class is located in *ossimPlanet/include/ossimPlanet/ossimPlanetEphemeris.h*.

Description:

The ossimPlanetEphemeris class allows one access to control the ephemeris model.

Public Member Functions

eyePositionXyz

Description:

Returns the x, y, and z position of the eye.

Syntax:

```
osg::Vec3d eyePositionXyz()const;
```

Example:

```
std::cout << "eyePositionXyz: " << viewer->ephemeris()-
>eyePositionXyz() << std::endl;

Result:
eyePositionXyz: 0 0 0
```

eyePositionLatLonHeight

Description:

Returns the latitude, longitude, and height position of the eye.

Syntax:

```
osg::Vec3d eyePositionLatLonHeight()const;
```

Example:

```
std::cout << "eyePositionLatLonHeight: " << viewer->ephemeris()-
>eyePositionLatLonHeight() << std::endl;

Result:
eyePositionLatLonHeight: 0 0 0
```

setMembers

Description:

Allows one to set what the current active members are. See the enumeration Members for more detail. This is an "ored" bit vector of Members to use.

Syntax:

```
enum Members
   {
      NO_MEMBERS    = 0,
      SUN_LIGHT     = 1,  // Enable the default Sun lighting
      MOON_LIGHT    = 2,  // Enable the default moon lighting
      AMBIENT_LIGHT = 4,
      SUN           = 8,
      MOON          = 16,
      SKY           = 32,
      FOG           = 64,
      ALL_MEMBERS   =
SKY|SUN_LIGHT|MOON_LIGHT|SUN|MOON|AMBIENT_LIGHT|FOG
   };

void setMembers(ossim_uint64 membersBitMap);
```

Parameter:

```
ossim_uint64 membersBitMap
```

memebrsBitMap - The members to set in the ephemeris model.

See Members enumeration for more details.

Example:

```
viewer->ephemeris()->setMembers(121);
```

members

Description:

The members bitmap describing the current members active in the ephemeris model. See the enumeration Members for more details.

Syntax:

```
ossim_uint64 members()const;
```

Example:

```
std::cout << "members: " << viewer->ephemeris()->members() <<
std::endl;

Result:
members: 121
```

setRoot

Syntax:

```
void setRoot(osg::Group* group);
```

Parameter:

```
osg::Group* group
```

group - The root graph to use for light source settings.

setMoonLightIndex

Description:

Allows one to set the openGL moonlight source index.

Syntax:

```
void setMoonLightIndex(ossim_uint32 idxNumber);
```

Parameter:

```
ossim_uint32 idxNumber
```

idxNumber - The openGL moonlight source index to set.

Example:

```
viewer->ephemeris()->setMoonLightIndex(1);
```

moonLightIndex

Description:

Returns the openGL moonlight source index.

Syntax:

```
ossim_uint32 moonLightIndex()const;
```

Example:

```
std::cout << "moonLightIndex: " << viewer->ephemeris()-
>moonLightIndex() << std::endl;

Result:
moonLightIndex: 1
```

setMoonLightCallback

Syntax:

```
void setMoonLightCallback(LightingCallback* callback);
```

moonPositionXyz

Description:

Returns the x, y, and z position of the moon.

Syntax:

```
osg::Vec3d moonPositionXyz()const;
```

Example:

```
std::cout << "moonPositionXyz: " << viewer->ephemeris()-
>moonPositionXyz() << std::endl;

Result:
moonPositionXyz: -34.4603 51.2539 -13.7927
```

moonPositionLatLonHeight

Description:

Returns the latitude, longitude, and height position of the moon.

Syntax:

```
osg::Vec3d moonPositionLatLonHeight()const;
```

Example:

```
std::cout << "moonPositionLatLonHeight: " << viewer->ephemeris()-
>moonPositionLatLonHeight() << std::endl;

Result:
moonPositionLatLonHeight: -12.5789 123.093 3.97246e+08
```

setSunLightIndex

Description:

Allows one to set the openGL sunlight source index.

Syntax:

```
void setSunLightIndex(ossim_uint32 idxNumber);
```

Parameter:

```
ossim_uint32 idxNumber
```

idxNumber - The openGL sunlight source index to set.

Example:

```
viewer->ephemeris()->setSunLightIndex(0);
```

sunLightIndex

Description:

Returns the openGL sunlight source index.

Syntax:

```
ossim_uint32 sunLightIndex()const;
```

Example:

```
std::cout << "sunLightIndex: " << viewer->ephemeris()->sunLightIndex()
<< std::endl;

Result:
sunLightIndex: 0
```

setSunLightCallback

Syntax:

```
void setSunLightCallback(LightingCallback* callback);
```

sunPositionXyz

Description:

Returns the x, y, and z position of the sun.

Syntax:

```
osg::Vec3d sunPositionXyz()const;
```

Example:

```
std::cout << “sunPositionXyz: " << viewer->ephemeris()-
>sunPositionXyz() << std::endl;

Result:
sunPositionXyz: 9841.69 -29571.3 6767.53
```

sunPositionLatLonHeight

Description:

Returns the latitude, longitude, and height position of the sun.

Syntax:

```
osg::Vec3d sunPositionLatLonHeight()const;
```

Example:

```
std::cout << "sunPositionLatLonHeight: " << viewer->ephemeris()-
>sunPositionLatLonHeight() << std::endl;

Result:
```

```
sunPositionLatLonHeight: 16.5286 294.996 1.51713e+11
```

setAutoUpdateSunColorFlag

Description:

Allows one to set the auto update for sun color calculations.

Syntax:

```
void setAutoUpdateSunColorFlag(bool flag);
```

Parameter:

```
bool flag
```

flag - The flag to set for auto update of sun color calculations.

setGlobalAmbientLightIndex

Description:

Allows one to set the openGL global ambient lighting source index.

Syntax:

```
void setGlobalAmbientLightIndex(ossim_uint32 idx);
```

Parameter:

```
ossim_uint32 idx
```

idx - The openGL ambient global lighting source index to set.

Example:

```
viewer->ephemeris()->setGlobalAmbientLightIndex(2);
```

globalAmbientLightIndex

Description:

Returns the openGL global ambient lighting source index.

Syntax:

```
ossim_uint32 globalAmbientLightIndex()const;
```

Example:

```
std::cout << "globalAmbientLightIndex: " << viewer->ephemeris()-
>globalAmbientLightIndex() << std::endl;

Result:
globalAmbientLightIndex: 2
```

setGlobalAmbientLight

Description:

Allows one to set the ambient lighting color value. The ambient light color is a normalized r, g, b value and range from 0.1.

Syntax:

```
void setGlobalAmbientLight(const osg::Vec3d& ambient);
```

Parameter:

```
const osg::Vec3d& ambient
```

ambient - The ambient light color value to set.

Example:

```
viewer->ephemeris()->setGlobalAmbientLight(osg::Vec3d(0.1, 0.1, 0.1));
```

setDate

Description:

This will set the date to use. If this is set then the update of the current time is disabled and will use date as fixed shading. When using this method we basically assume that you are driving the date and time. To reset back to auto update, just call setAutoUpdateToCurrentTimeFlag.

Note, for ephemeris calculations we normalize the date on the fly to GMT time so we will always hold onto the original date passed.

Syntax:

```
void setDate(const ossimLocalTm& date);
```

Parameter:

```
const ossimLocalTm& date
```

time - The ossimLocalTm object.

Example:

```
ossimLocalTm date;
date.now();

viewer->ephemeris()->setDate(date);
```


setAutoUpdateToCurrentTimeFlag

Description:

Allows one to let the ephemeris updates continually change based on the current time.

Syntax:

```
void setAutoUpdateToCurrentTimeFlag(bool flag);
```

Parameter:

```
bool flag
```
flag - The flag to set for auto update to current time.

Example:

```
viewer->ephemeris()->setAutoUpdateToCurrentTimeFlag(true);
```

setApplySimulationTimeOffsetFlag

Description:

This will disable the Auto update to current time setting and do a relative displacement of the current date settings to the sim time passed in the osg::NodeVisitor.

Currently the relative simulation time displacement units must be in seconds.

Syntax:

```
void setApplySimulationTimeOffsetFlag(bool flag);
```

Parameter:

```
bool flag
```
flag - The flag to apply simulation offset time.

Example:

```
viewer->ephemeris()->setApplySimulationTimeOffsetFlag(true);
```

setCamera

Description:

This is used by the ossimPlanetViewer to setup a slave camera for the drawing the sun moon and dome. We were having problems with large displacements being in

the same graph as the terrain. The near and far planes were causing the terrain to clipped at low altitudes.

This might go away in the future, but for now I am playing with coupling to the viewer's master camera.

Syntax:

```
void setCamera(osg::Camera* camera);
```

setVisibility

Description:

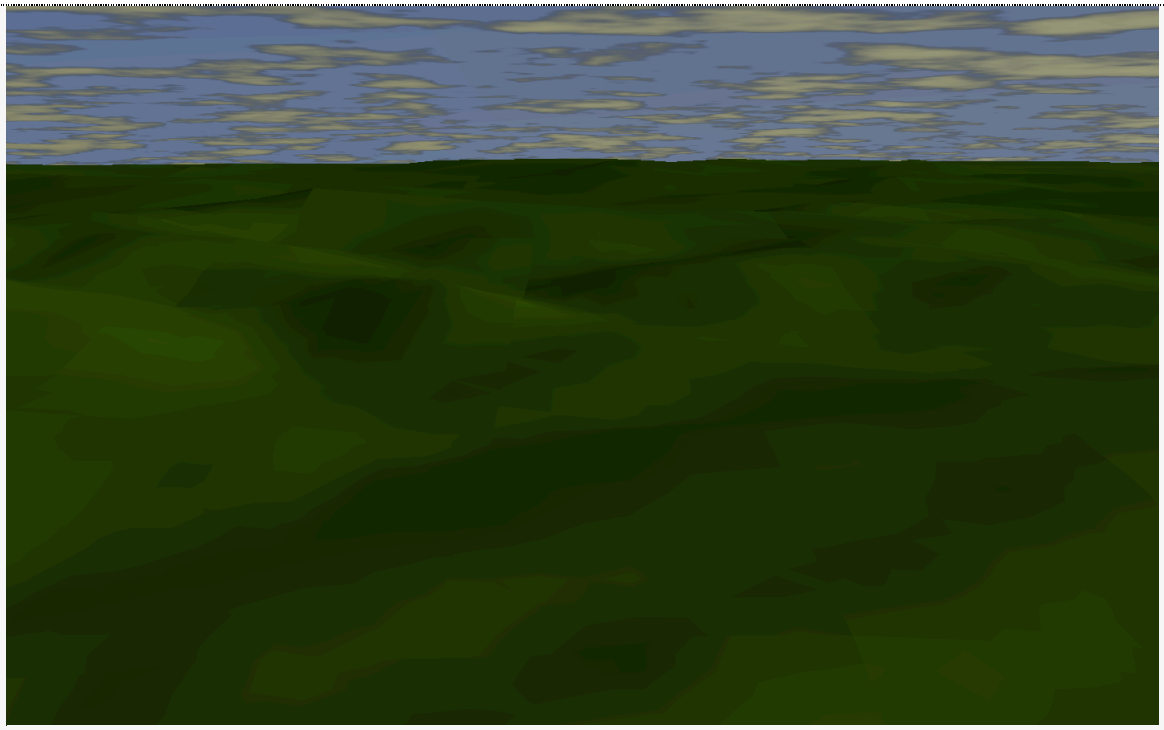
Will use the visibility to set the far plane and the density values of the fog. The visibility is in meters.

Syntax:

```
void setVisibility(ossim_float64 visibility);
```

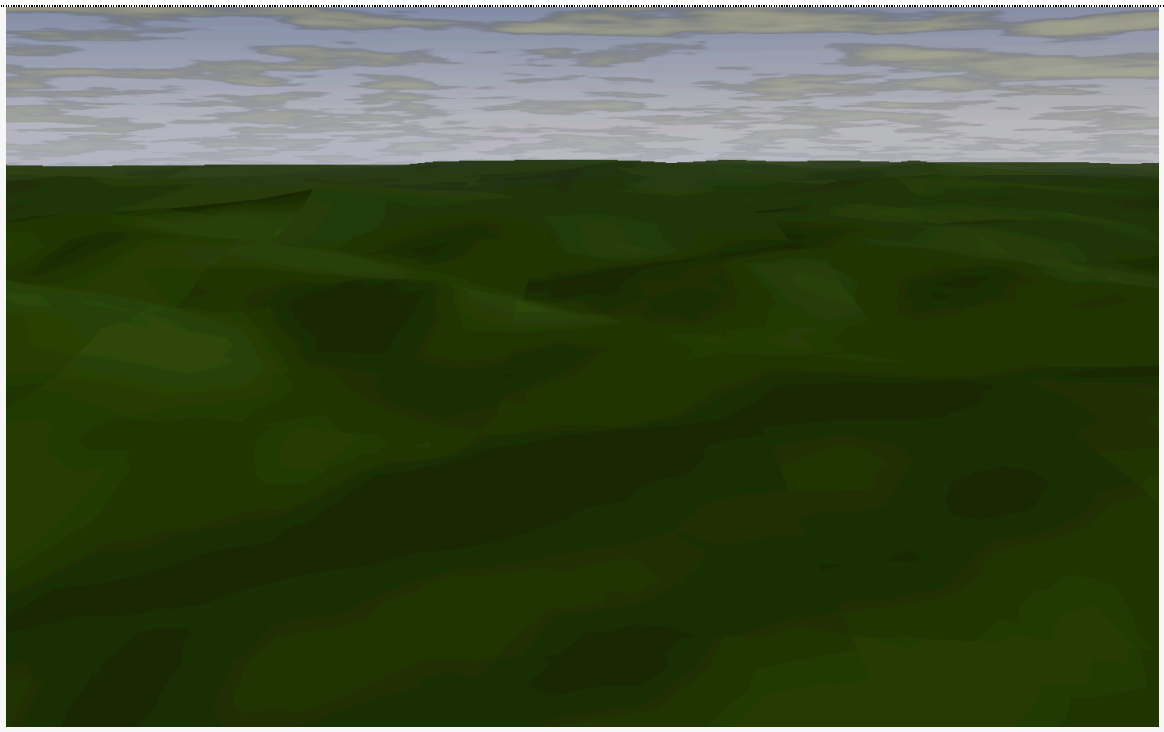
Examples:

```
Clear Day
```



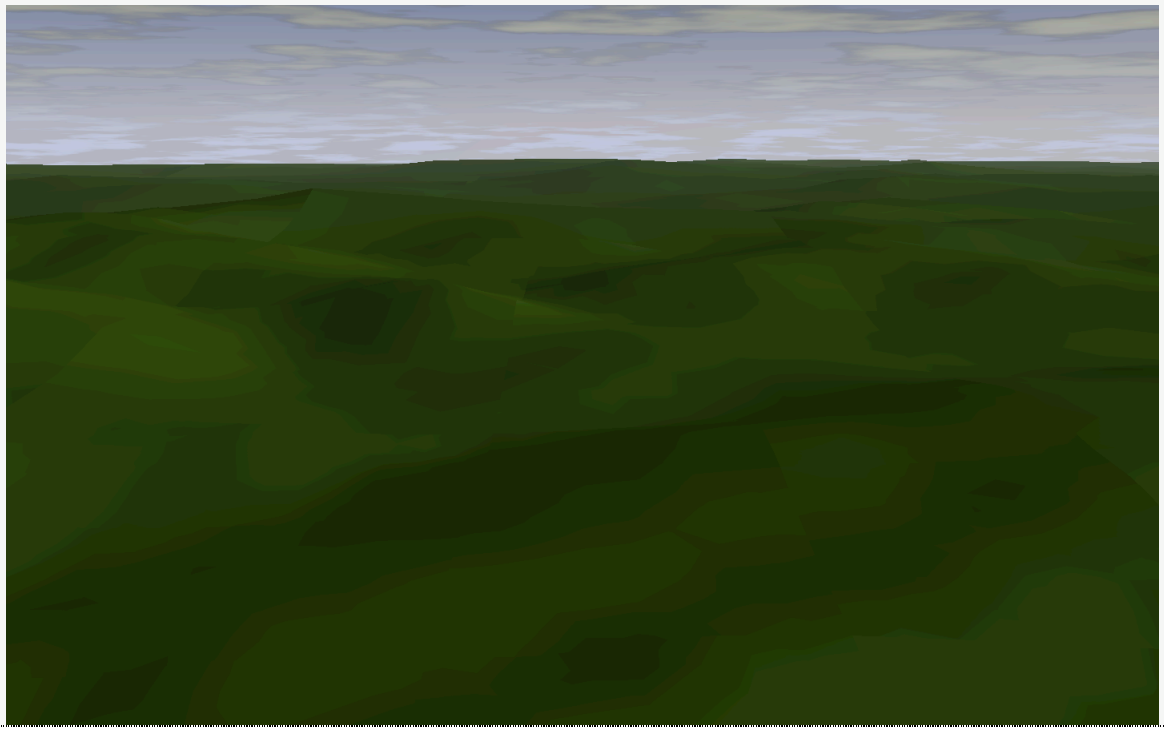
```
viewer->ephemeris()->setFogEnableFlag(true);
viewer->ephemeris()->setVisibility(2000000);
```

```
Very Light Fog
```



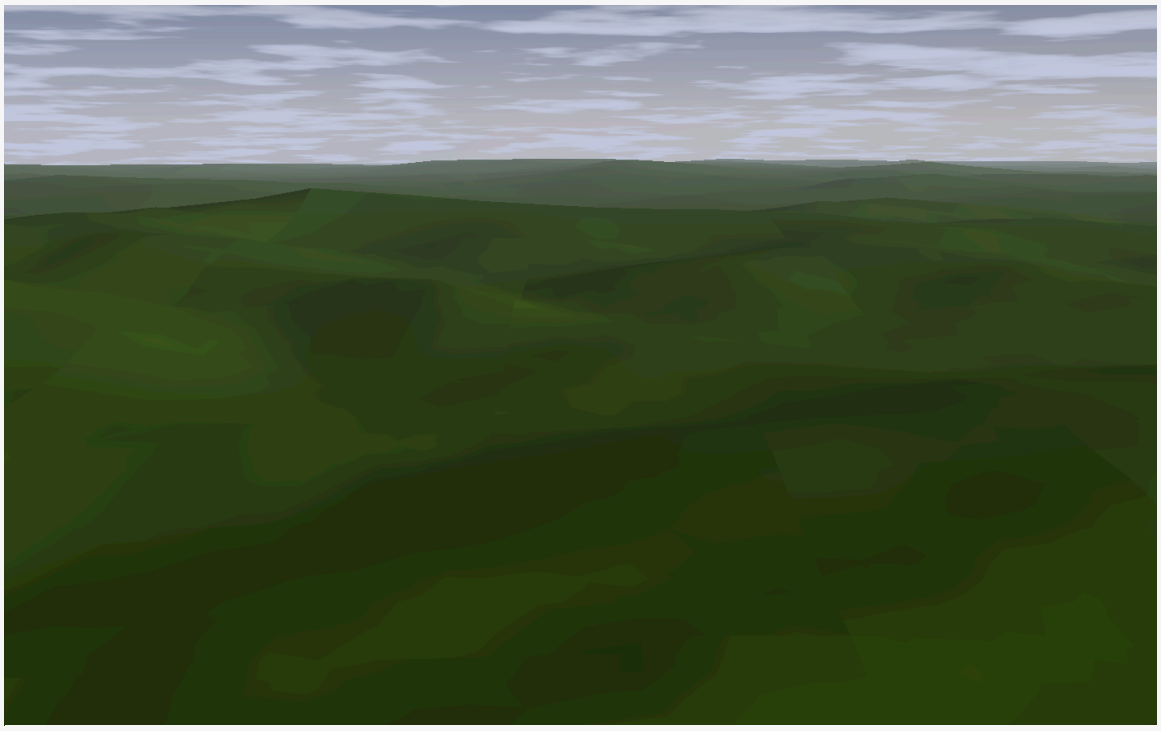
```
viewer->ephemeris()->setFogEnableFlag(true);
viewer->ephemeris()->setVisibility(1000000);
```

Light Fog



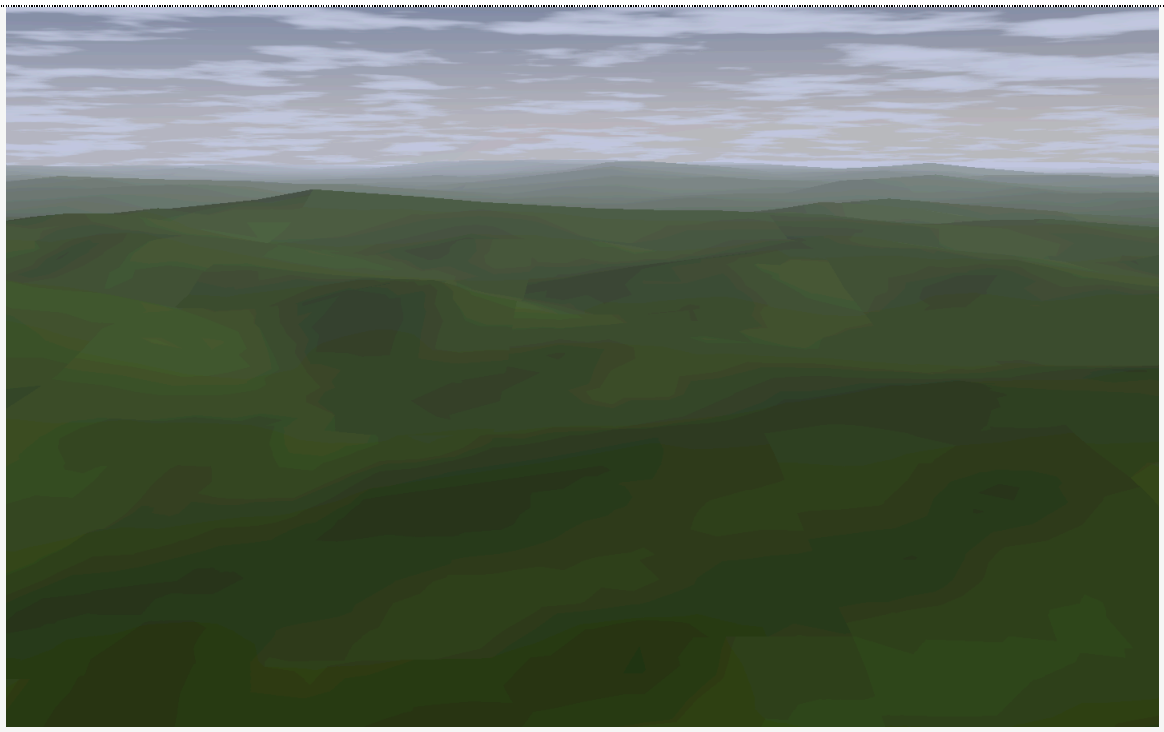
```
viewer->ephemeris()->setFogEnableFlag(true);
viewer->ephemeris()->setVisibility(500000);
```

Medium Fog



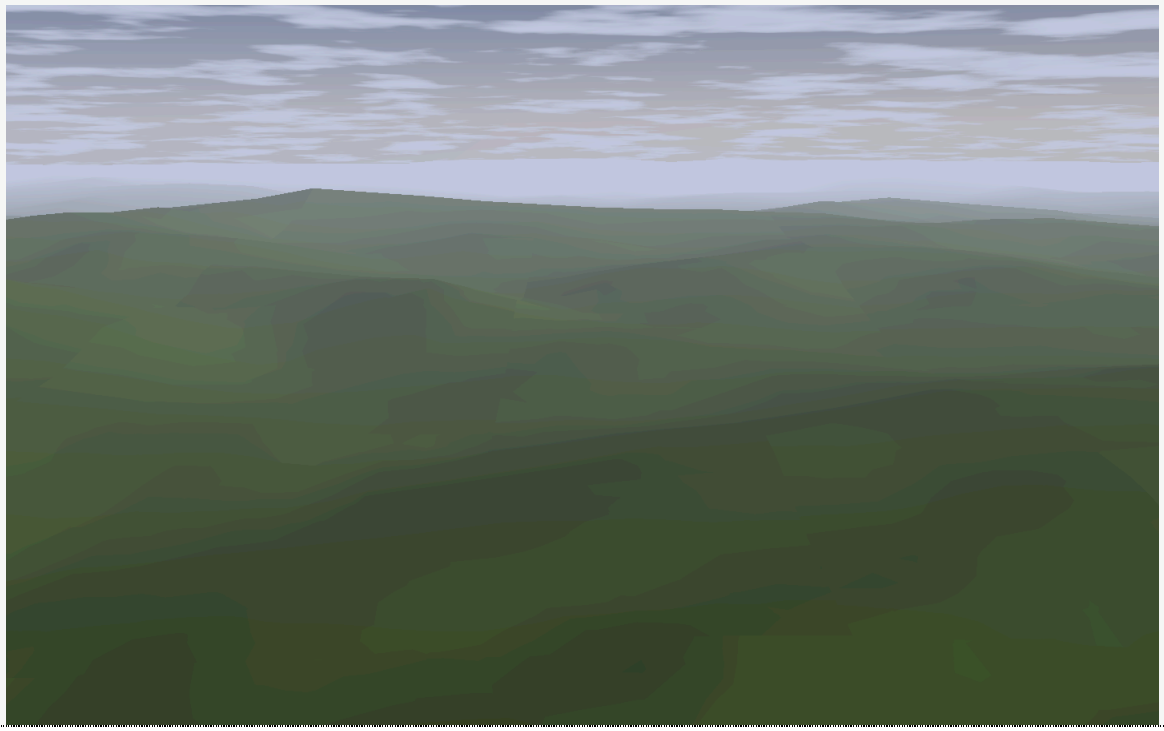
```
viewer->ephemeris()->setFogEnableFlag(true);
viewer->ephemeris()->setVisibility(200000);
```

Medium Heavy Fog



```
viewer->ephemeris()->setFogEnableFlag(true);
viewer->ephemeris()->setVisibility(100000);
```

Heavy Fog



```
viewer->ephemeris()->setFogEnableFlag(true);
viewer->ephemeris()->setVisibility(50000);
```

visibility

Description:

Returns the visibility factor.

Syntax:

```
ossim_float64 visibility()const;
```

Example:

```
std::cout << "visibility: " << viewer->ephemeris()->visibility() <<
std::endl;

Result:
// Clear Day
visibility: 2000000

// Very Light Fog
visibility: 1000000

// Light Fog
visibility: 500000

// Medium Fog
visibility: 200000

// Medium Heavy Fog
visibility: 100000

// Heavy Fog
visibility: 50000
```

setSunTextureFromFile

Description:

Allows one to pass a texture in by file. It uses this texture to show the position of the sun.

Syntax:

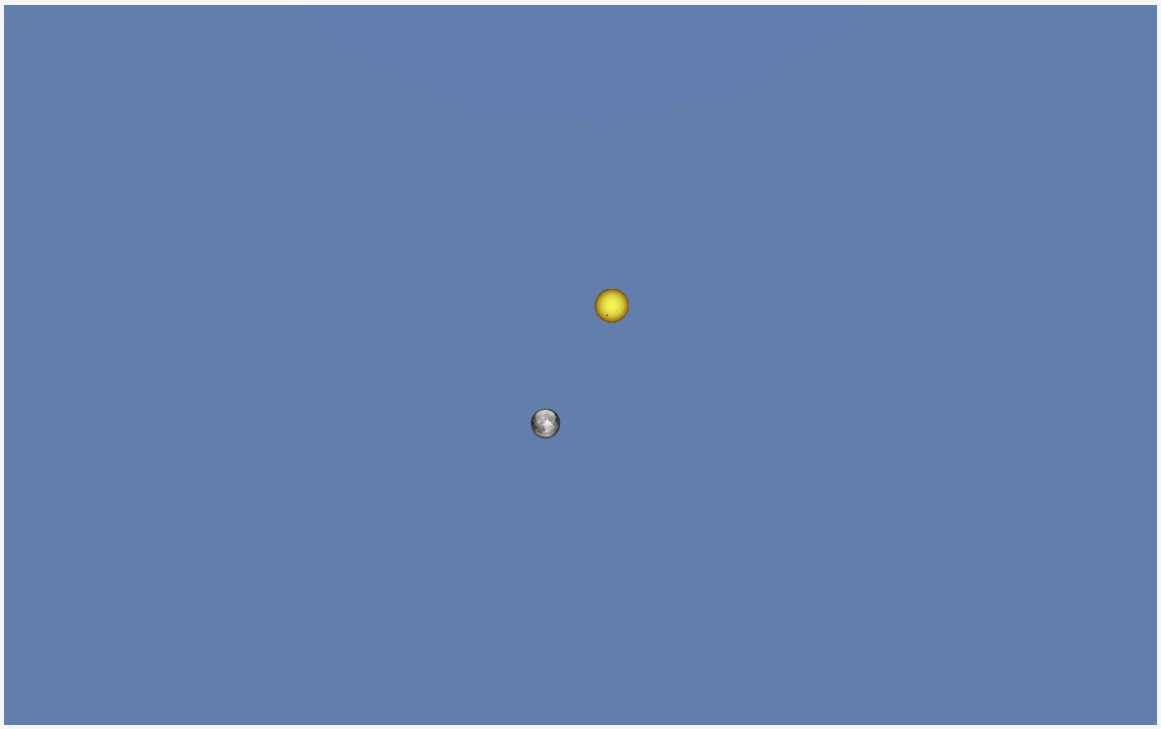
```
void setSunTextureFromFile(const ossimFilename& texture);
```

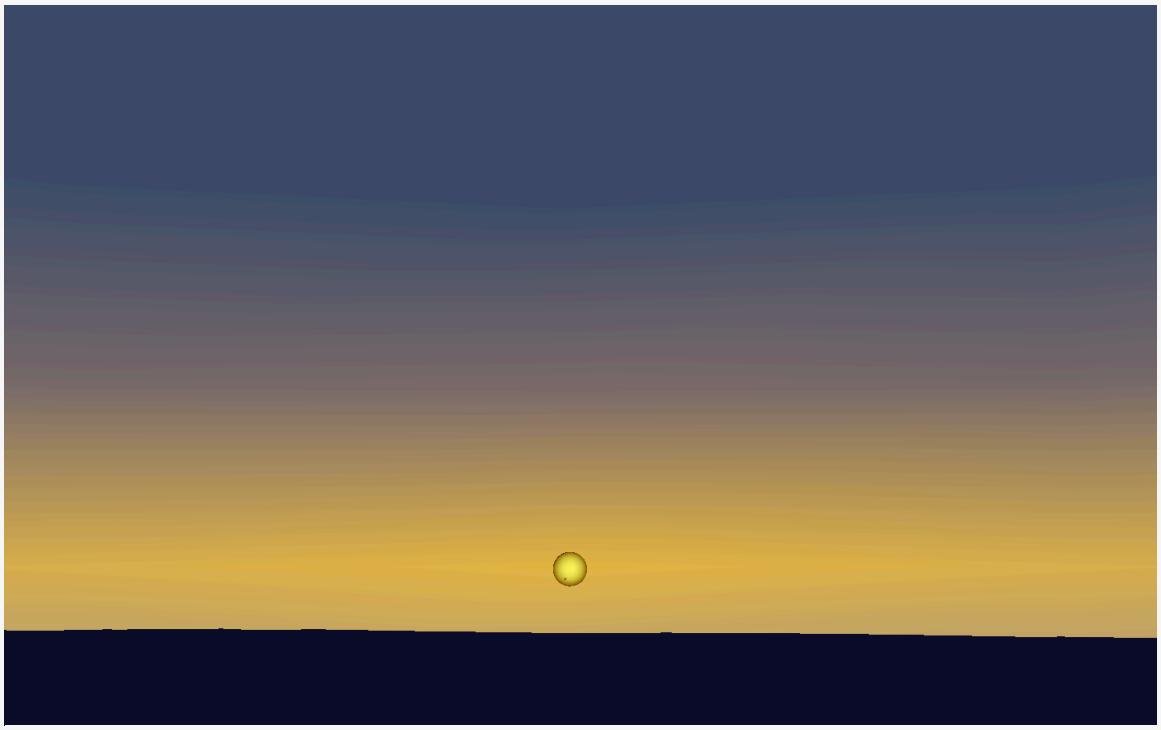
Parameter:

```
const ossimFilename& texture
```

texture - Filename for the texture. Typically a file format supported by OSG and has an alpha channel. Best is an RBA in png format.

Example:





```
ossimFilename sunTextureFile =
"/data/ossimplanettest/images/icons/sun.png");

viewer->ephemeris()->setSunTextureFromFile(sunTextureFile);
```

setSunTextureFromImage

Description:

Allows one to pass a texture by osg::Image. It uses this texture to show the position of the sun.

Syntax:

```
void setSunTextureFromImage(osg::Image* texture);
```

Parameter:

```
osg::Image* texture
```

texture - Image for the texture. Should be an RGBA type image.

setMoonTextureFromFile

Description:

Allows one to pass a texture in by file. It uses this texture to show the position of the moon.

Syntax:

```
void setMoonTextureFromFile(const ossimFilename& texture);
```

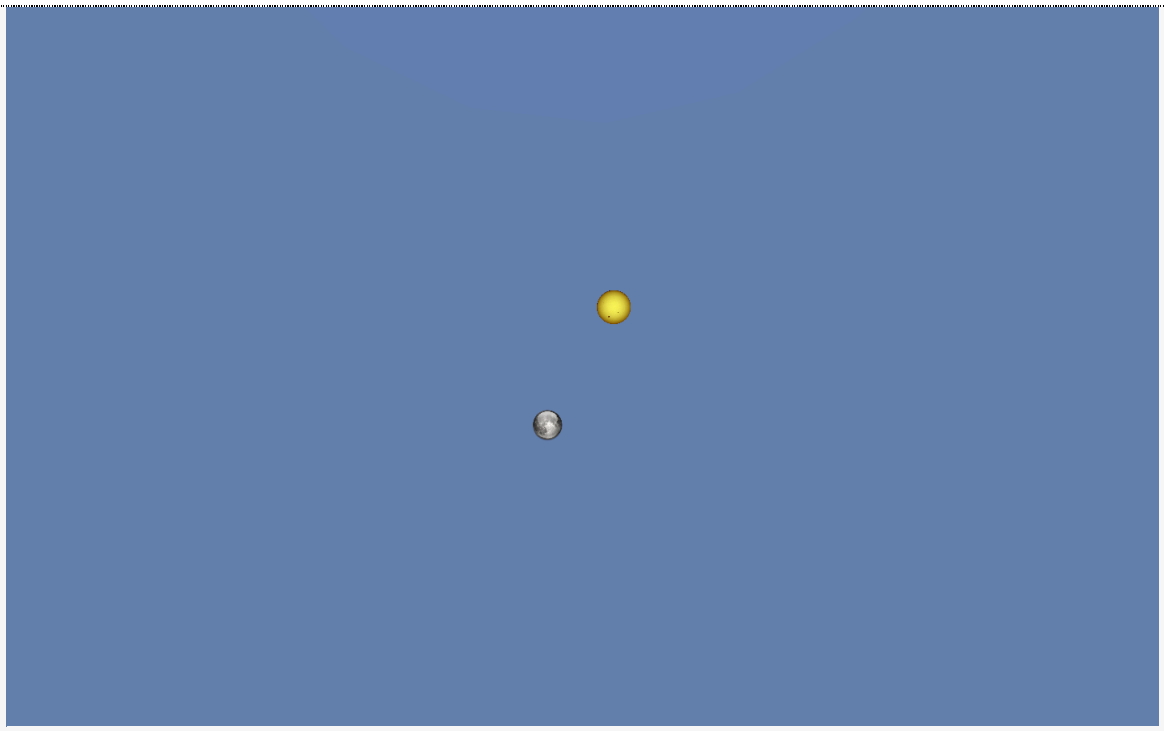
Parameter:

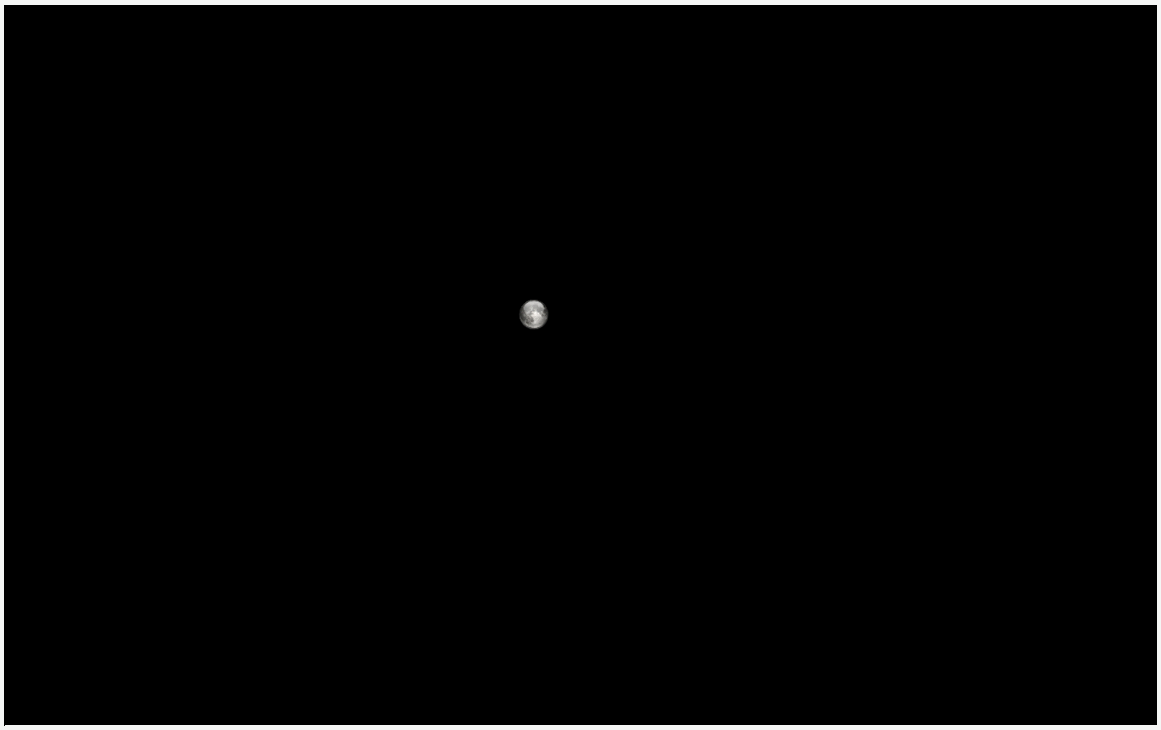
```
const ossimFilename& texture
```

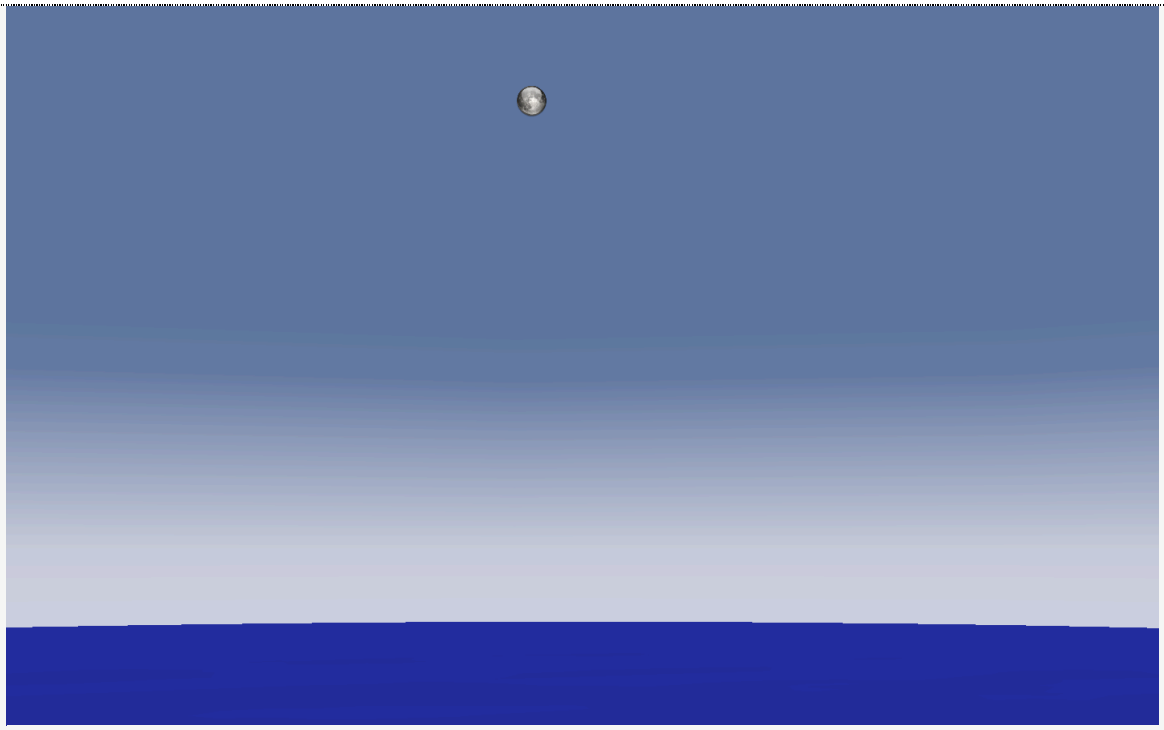
texture - Filename for the texture. Typically a file format supported by OSG and has an alpha channel. Best is an RBA in png format.

Example:

```

```







```
ossimFilename moonTextureFile =
"/data/ossimplanettest/images/icons/moon.png");

viewer->ephemeris()->setMoonTextureFromFile(moonTextureFile);
```


setMoonTextureFromImage

Description:

Allows one to pass a texture by osg::Image. It uses this texture to show the position of the moon.

Syntax:

```
void setMoonTextureFromImage(osg::Image* texture);
```

Parameter:

```
osg::Image* texture
```

texture - Image for the texture. Should be an RGBA type image.

setSunMinMaxPixelSize

Description:

Allows one to set the minimum and maximum sun pixel size.

Syntax:

```
void setSunMinMaxPixelSize(ossim_uint32 minPixelSize,
                           ossim_uint32 maxPixelSize);
```

Parameters:

```
ossim_uint32 minPixelSize
```

minPixelSize - The minimum sun pixel size to set.

```
ossim_uint32 maxPixelSize
```

maxPixelSize - The maximum sun pixel size to set.

Example:

```
viewer->ephemeris()->setSunMinMaxPixelSize(64, 64);
```

setMoonMinMaxPixelSize

Description:

Allows one to set the minimum and maximum moon pixel size.

Syntax:

```
void setMoonMinMaxPixelSize(ossim_uint32 minPixelSize,
                            ossim_uint32 maxPixelSize);
```

Parameters:

```
ossim_uint32 minPixelSize
```

minPixelSize - The minimum moon pixel size.

```
ossim_uint32 maxPixelSize
```

maxPixelSize - The maximum moon pixel size.

Example:

```
viewer->ephemeris()->setMoonMinMaxPixelSize(64, 64);
```

setMaximumAltitudeToShowDomeInMeters

Description:

Allows one to set the maximum altitude to show the dome in meters.

Syntax:

```
void setMaximumAltitudeToShowDomeInMeters(ossim_float64 maxAltitude);
```

setMaximumAltitudeToShowFogInMeters

Description:

Allows ones to sets the maximum altitude to show fog in meters.

Syntax:

```
void setMaximumAltitudeToShowFogInMeters(ossim_float64 maxAltitude);
```

setSkyColorAdjustmentTable

Description:

This is an intensity/brightness factor multiplied by the sky color based on sun elevation angle with respect to the eye. The table is a pair of values where the first value is the elevation angle and the second value is the intensity. This table is used

to do a bilinear interpolation of the brightness factor applied to the current sky color.

Syntax:

```
void setSkyColorAdjustmentTable(IntensityTableType& table);
```

Parameter:

```
IntensityTableType& table
```

table - a std::map of values that takes pairs elevation angle in degrees and brightness factor.

skyColorAdjustmentTable

Desription:

Returns the adjustment table used to modify the sky color. This is a table of elevation angles in degrees and brightness factors used to attenuate the sky color.

Syntax:

```
const IntensityTableType* skyColorAdjustmentTable()const;
```

setBaseSkyColor

Description:

Allows one to set the base sky color value. The base sky color is a normalized r, g, b value and range from 0.1.

Syntax:

```
void setBaseSkyColor(const osg::Vec3d& color);
```

Parameter:

```
const osg::Vec3d& color
```
color - The base sky color value to set.

Example:

```
viewer->ephemeris()->setBaseSkyColor(osg::Vec3d(0.39, 0.5, 0.74));
```

getBaseSkyColor

Description:

Returns the base sky color value.

Syntax:

```
osg::Vec3d getBaseSkyColor()const;
```

Example:

```
std::cout << "getBaseSkyColor: " << viewer->ephemeris()-
>getBaseSkyColor() << std::endl;

Result:
getBaseSkyColor: 0.39 0.5 0.74
```

setBaseFogColor

Description:

Allows one to set the base fog color value. The base fog color is a normalized r, g, b value and range from 0.1.

Syntax:

```
void setBaseFogColor(const osg::Vec3d& color);
```

Parameter:

```
const osg::Vec3d& color
```

color - The base fog color value to set.

Example:

```
viewer->ephemeris()->setBaseFogColor(osg::Vec3d(0.84, 0.87, 1));
```

getBaseFogColor

Description:

Returns the base fog color value.

Syntax:

```
osg::Vec3d getBaseFogColor()const;
```

Example:

```
std::cout << "getBaseFogColor: " << viewer->ephemeris()-
>getBaseFogColor() << std::endl;

Result:
getBaseFogColor: 0.84 0.87 1
```

setFogMode

Description:

This mirrors the GL fog. You can currently have LINEAR, EXP, and EXP2 fog affects. See Open gl for further information on the GL fog paramters. LINEAR uses the near and far to set fog attenuation the others attenuate based on distance.

Syntax:

```
enum FogMode
   {
      LINEAR = 0, ///<Linear fog
```

```
      EXP,        ///<Exponential fog
      EXP2,       ///<Exponential squared fog
};

void setFogMode(FogMode mode);
```

Parameter:

```
FogMode mode
```

mode – The fog mode to use.

See FogMode enumeration for more details.

Example:

```
viewer->ephemeris()->setFogMode(ossimPlanetEphemeris::LINEAR);

Result:
Linear Fog
```

```
viewer->ephemeris()->setFogMode(ossimPlanetEphemeris::EXP);

Result:
Exponential Fog
```

```
viewer->ephemeris()->setFogMode(ossimPlanetEphemeris::EXP2);

Result:
Exponential Squared Fog
```


setFogNear

Description:

Allows one to set the near plane where the fog begins.

Syntax:

```
void setFogNear(ossim_float64 value);
```

Parameter:

```
ossim_float64 value
```

value - The far plane value in meters.

Example:

```
viewer->ephemeris()->setFogNear(20.0);
```

setFogFar

Description:

Set the far plane where the fog ends. This is automatically set based on visibility.

Syntax:

```
void setFogFar(ossim_float64 value);
```

Parameter:

```
ossim_float64 value
```

value - The far plane value in meters.

Example:

```
viewer->ephemeris()->setFogNear(40.0);
```

setFogDensity

Description:

Density is automatically calculated based on visibility.

Syntax:

```
void setFogDensity(ossim_float64 value);
```

Parameter:

```
ossim_float64 value
```

value - The density value to use. This should be set after the visibility is set.

setFogEnableFlag

Description:

Allows one to enable and disable fog in the scene.

Syntax:

```
void setFogEnableFlag(bool flag);
```

Parameter:

```
bool flag
```

flag - enables and disables the fog. Pass in true to enable and false to disable.

Example:

```
viewer->ephemeris()->setFogEnableFlag(true);

Result:
Fog enabled
```

```
viewer->ephemeris()->setFogEnableFlag(false);

Result:
Fog disabled
```

setNumberOfCloudLayers

Description:

Resizes the list to the specified number of cloud layers.

Syntax:

```
void setNumberOfCloudLayers(ossim_uint32 numberOfLayers);
```

Parameter:

```
ossim_uint32 numberOfLayers
```

numberOfLayers The number of cloud layers to resize the list to.

cloudLayer

Description:

Returns the cloud layer specified at idx. If idx is out of range then a value of null or 0 is returned.

Syntax:

```
ossimPlanetCloudLayer* cloudLayer(ossim_uint32 idx);
```

Parameter:

```
ossim_uint32 idx
```

idx - The index of the cloud layer to return.

numberOfCloudLayers

Description:

Returns the number of cloud layers.

Syntax:

```
ossim_uint32 numberOfCloudLayers()const;
```

createCloudPatch

Description:

This is a utility method that allows one to create a patch of clouds centered at the give lat lon height. In short, we shift the creation of the patch to lat lon height to 0,0,0 so the patch is square and then use a Matrix transform to move it to a new location defined by the center lat lon height. The seed, coverage and sharpness are exposed to give you some control over the texture generated. The seed is used to initialize the random number generator so you should be able to give the same parameters and generate the same texture. The coverage and sharpness are used to control how much clouds there are and how soft they appear. For example coverage of say 20 and sharpness of .96 gives a nice look. If you want them to be more overcast and softer, then increase the coverage to say 150 and the sharpness to around .99.

Syntax:

```
void createCloudPatch(ossim_uint32 cloudLayerIndex,
                      const osg::Vec3d& theCenterLatLonHeight,
                      ossim_float64 numberOfMeshSamples,
                      ossim_float64 patchSizeInDegrees,
                      ossim_uint64  seed,
                      ossim_float64 coverage,
                      ossim_float64 sharpness);
```

Parameters:

```
ossim_uint32 cloudLayerIndex
```

cloudLayerIndex - The cloud layer to create a patch for.

```
const osg::Vec3d& theCenterLatLonHeight
```

theCenterLatLonHeight - The center point to place the cloud patch.

```
ossim_float64 numberOfMeshSamples
```

numberOfMeshSamples - This is square so a value of 128 will create a mesh sample of 128x128.

```
ossim_float64 patchSizeInDegrees
```

patchSizeInDegrees Defines the patch size in depgrees.

```
ossim_uint64  seed
```

seed - This is the seed value used to generate a cloud texture.

```
ossim_float64 coverage
```

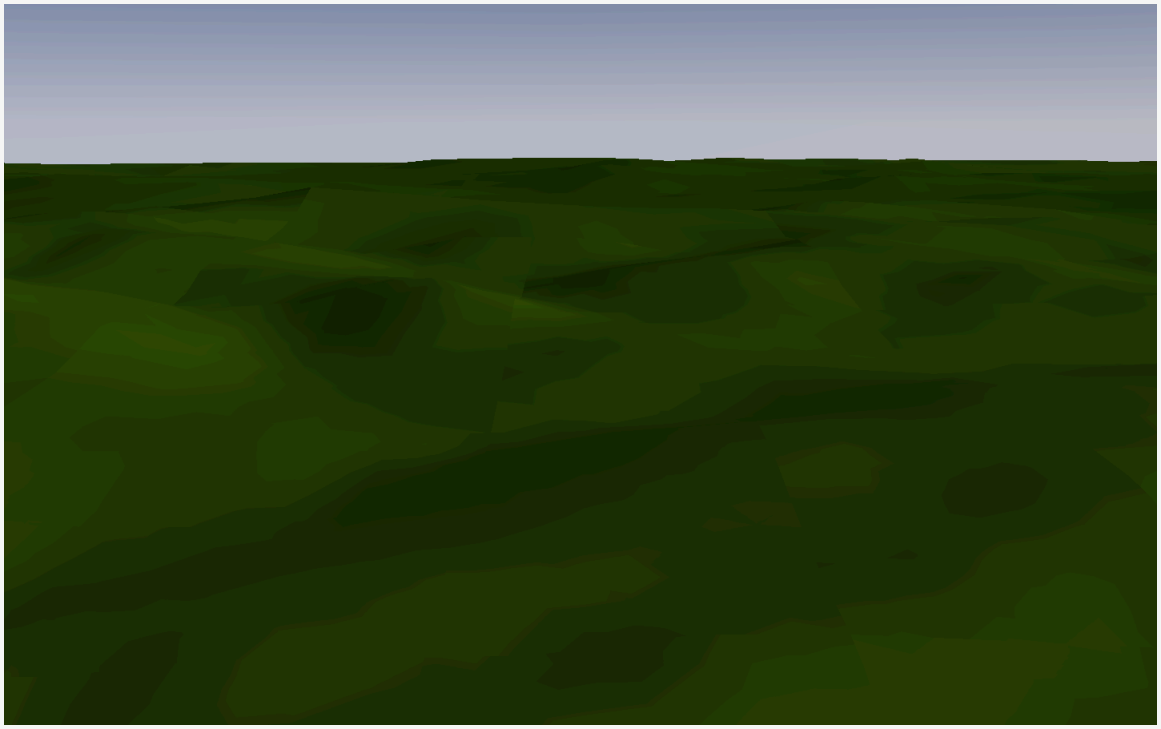
coverage - This is used to generate a texture. This value is not a percentage and is a value to control the way the clouds look.

```
ossim_float64 sharpness
```

sharpness - The closer the value is to 1.0 the softer the clouds and the closer you go to 0 the harder the clouds look.

Chapter Summary

Ephemeris model.



```
// Add the ephemeris and set active members
viewer->addEphemeris(ossimPlanetEphemeris::SUN_LIGHT
   //|ossimPlanetEphemeris::MOON_LIGHT
   //|ossimPlanetEphemeris::AMBIENT_LIGHT
   |ossimPlanetEphemeris::SUN
   |ossimPlanetEphemeris::MOON
   |ossimPlanetEphemeris::SKY
   |ossimPlanetEphemeris::FOG
   );

// Set the global ambient light.
viewer->ephemeris()->setGlobalAmbientLight(osg::Vec3d(0.1, 0.1, 0.1));

// Set the date.
ossimLocalTm date;
date.now();
viewer->ephemeris()->setDate(date);

// Turn off simulation time offset.
viewer->ephemeris()->setApplySimulationTimeOffsetFlag(false);

// Set the visibility.
ossim_float64 visibility = 1000000000.0;
viewer->ephemeris()->setVisibility(visibility);

// Set the sun attributes.
ossimFilename sunTextureFile =
"/data/ossimplanettest/images/icons/sun.png";
viewer->ephemeris()->setSunTextureFromFile(sunTextureFile);
viewer->ephemeris()->setSunMinMaxPixelSize(64, 64);

// Set the moon attributes.
ossimFilename moonTextureFile =
"/data/ossimplanettest/images/icons/moon.png";
viewer->ephemeris()->setMoonTextureFromFile(moonTextureFile);
viewer->ephemeris()->setMoonMinMaxPixelSize(64, 64);

// Set the fog mode.
viewer->ephemeris()->setFogMode(ossimPlanetEphemeris::LINEAR);

// Set the near fog.
ossim_float64 fogNear = 20.0;
viewer->ephemeris()->setFogNear(fogNear);

// Enable ephemeris.
viewer->ephemeris()->setFogEnableFlag(true);
```

ossimPlanetViewer.h

```
#include <ossimPlanet/ossimPlanetViewer.h>
```

The viewer class is located in *ossimPlanet/include/ossimPlanetViewer.h*.

Description:

ossimPlanetViewer will serve as a higher level interface to most of the ossimPlanet's inner workings. During the update phase it will determine if the scene graph is in a static state ad will notify the user through a virtual method callback. At the time of this documentation we have not written then API to auto setup the planet node. This is an example to setup the planet node and then set it to the Viewer. Note this code will eventually be moved into a single setup routine on ossimPlanetViewer.

Public Member Functions:

addAnnotation

Description:

This will take the passed in annotation node and add it to the annotation layer in planet.

Syntax:

```
bool addAnnotation(osg::ref_ptr<ossimPlanetAnnotationLayerNode>
annotation);
```

Parameter:

```
osg::ref_ptr<ossimPlanetAnnotationLayerNode> annotation
```

annotation - The annotation to add to the planet.

addImageTexture (reference layer)

Description:

This will take a texture layer and add it to the top of the reference layer. This will return true if the layer was added successfully.

This return true if the layer was added successfully.

Syntax:

```
bool addImageTexture(osg::ref_ptr<ossimPlanetTextureLayer>
imageTexture);
```

Parameter:

```
Osg::ref_ptr<ossimPlanetTextureLayer> imageTexture
```

imageTexture - Texture layer to be added.

addImageTexture (root layer)

Description:

Will open all entries and return as a root image layer. If this image file only contains a single image then that layer is returned. If multiple images are found then it will open each image and create a Layer group and return the images all grouped together.

This will return the Root layer node.

Syntax:

```
bool addImageTexture(osg::ref_ptr<ossimPlanetTextureLayer>
imageTexture);
```

Parameter:

```
const ossimString& file
```

file - The file to open and add to the graph.

addElevation

Description:

Allows one to add an elevation database.

Syntax:

```
bool addElevation(osg::ref_ptr<ossimPlanetElevationDatabase> database,
bool sortFlag=false);
```

Parameters:

```
osg::ref_ptr<ossimPlanetElevationDatabase> database
```

database - The elevation database to add.

```
bool sortFlag=false
```

sortFlag - The flag to enable resolution sorting in the elevation database.

addKml

Description:

Allows one to add a kml resource to the kml layer.

Syntax:

```
void addKml(const ossimFilename& file);
```

Parameter:

```
const ossimFilename& file
```

file - The Kml resource to add.

addEphemeris

Description:

Allows one to add ephemeris environment to the viewer.

Syntax:

```
enum Members
   {
      NO_MEMBERS    = 0,
      SUN_LIGHT     = 1,  // Enable the default Sun lighting
      MOON_LIGHT    = 2,  // Enable the default moon lighting
      AMBIENT_LIGHT = 4,
      SUN           = 8,
      MOON          = 16,
      SKY           = 32,
      FOG           = 64,
      ALL_MEMBERS   =
SKY|SUN_LIGHT|MOON_LIGHT|SUN|MOON|AMBIENT_LIGHT|FOG
   };

void addEphemeris(ossim_uint32 memberBitMask);
```

Example:

```
viewer->addEphemeris(ossimPlanetEphemeris::SUN_LIGHT
   //|ossimPlanetEphemeris::MOON_LIGHT
   //|ossimPlanetEphemeris::AMBIENT_LIGHT
   |ossimPlanetEphemeris::SUN
   |ossimPlanetEphemeris::MOON
   |ossimPlanetEphemeris::SKY
   |ossimPlanetEphemeris::FOG
   );
```

removeEphemeris

Description:

Allows one to remove ephemeris environment from the viewer.

Syntax:

```
void removeEphemeris();
```

Example:

```
viewer->removeEphemeris();
```

Chapter Summary

```
viewer->addEventHandler(new osgGA::StateSetManipulator(theCanvas-
>getCamera()->getOrCreateStateSet()));

viewer->addEventHandler(new osgViewer::StatsHandler());

viewer->setThreadingModel(osgViewer::Viewer::SingleThreaded);

osg::ref_ptr<ossimPlanet> planet = new ossimPlanet;

planet->setupDefaults();

planet->land()-
>setCurrentFragementShaderType(ossimPlanetShaderProgramSetup::NO_SHADER
);

viewer->setSceneData(planet.get());

viewer-
>addImageTexture("/data/ossimplanettest/images/textures/reference/earth
.jpg");

/*
This example sets some GUI action handlers supported by OSG.  We
allocate a planet and then call planets setupDefaults().  I then set
the scene to the viewer and then add an image to the earth.  The last
thing we did was setup the manipulator that will manipulate the eye
matrix of OSG.  We have our own manipulator called
ossimPlanetManipulator.
*/
```

ossimPlanetAnnotationLayer.h

```
#include <ossimPlanet/ossimPlanetAnnotationLayer.h>
```

The annotation layer class is located in *ossimPlanet/include/ossimPlanet/ossimPlanetAnnotationLayer.h*.

defaultIconTexture

Syntax:

```
    osg::ref_ptr<osg::Texture2D> defaultIconTexture()
    {
      return theDefaultIconTexture;
      }
```

defaultFont

Syntax:

```
osg::ref_ptr<osgText::Font> defaultFont()
   {
      return theDefaultFont.get();
      }
```

defaultFont (constant)

Syntax:

```
const osg::ref_ptr<osgText::Font> defaultFont()const
   {
      return theDefaultFont.get();
      }
```

removeByNameAndId

Syntax:

```
void removeByNameAndId(const ossimString& name, const ossimString& id);
```

ossimPlanetAnnotationLayerNode.h

```
#include <ossimPlanet/ossimPlanetAnnotationLayerNode.h>
```

The annotation layer node class is located in *ossimPlanet/include/ossimPlanet/ossimPlanetAnnotationLayerNode.h*.

setColor (Annotation Color Style)

Description:

Allows one to set the color.

Syntax:

```
void setColor(const osg::Vec4d& color)
   {
      theColor = color;
   }
```

Parameter:

```
const osg::Vec4d& color
```

color – The color to set.

color

Description:

Returns the color.

Syntax:

```
const osg::Vec4d& color()const
   {
      return theColor;
   }
```

setColorMode

Description:

Allows one to set the color mode.

Syntax:

```
void setColorMode(ossimPlanetAnnotationColorMode mode)
   {
      theColorMode = mode;
   }
```

Parameter:

```
ossimPlanetAnnotationColorMode mode
```

mode – The color mode to set.

colorMode

Description:

Returns the color mode.

Syntax:

```
ossimPlanetAnnotationColorMode colorMode()const
   {
      return theColorMode;
   }
```

setScale (Annotation Label Style)

Description:

Allows one to set the scale.

Syntax:

```
void setScale(ossim_float64 scale)
   {
      theScale = scale;
   }
```

Parameter:

```
Ossim_float64 scale
```

scale – The scale to set.

scale

Description:

Returns the scale.

Syntax:

```
Ossim_float64 scale()const
   {
      return theScale;
   }
```

setDuration (Annotation Time Expire)

Description:

Allows one to set the duration.

Syntax:

```
void setDuration(double value)
   {
      theDuration = value;
   }
```

Parameter:

```
double value
```
value – The duration to set.

duration
Description:

Returns the duration.

Syntax:

```
double duration()const
   {
      return theDuration;
   }
```

ossimPlanetTimeUnit
Description:

Returns the time unit.

Syntax:

```
ossimPlanetTimeUnit unit()const
   {
      return theUnit;
   }
```

initTimeStamp
Description:

Allows one to initialize the time stamp.

Syntax:

```
virtual void initTimeStamp()
```

```
    {
        theInitialStamp = osg::Timer::instance()->tick();
    }
```

hasExpired

Syntax:

```
virtual bool hasExpired()const
    {
        return osg::Timer::instance()->delta_s(theInitialStamp,
osg::Timer::instance()->tick())>=theDuration;
    }
```

setExtrudeFlag (Annotation Geometry)

Description:

Allows one to set the extrude flag.

Syntax:

```
    virtual void setExtrudeFlag(bool flag)
    {
        theExtrudeFlag = flag;
        }
```

Parameter:

```
bool flag
```

flag – The flag to set extrude.

setAltitudeMode

Description:

Allows on to set the altitude mode.

Syntax:

```
virtual void setAltitudeMode(ossimPlanetAltitudeMode mode)
{
   theAltitudeMode = mode;
   }
```

Parameter:

```
ossimPlanetAltitudeMode mode
```

mode – The altitude mode to set.

altitudeMode

Description:

Returns the altitude mode.

Syntax:

```
ossimPlanetAltitudeMode altitudeMode()const
{
   return theAltitudeMode;
   }
```

setCoordinate

Description:

Allows one to set the coordinate.

Syntax:

```
void setCoordinate(const osg::Vec3d& coordinate)
{
   theCoordinate = coordinate;
   }
```

Parameter:

```
const osg::Vec3d& coordinate
```

coordinate – The coordinate to set.

coordinate

Description:

Returns the coordinate.

Syntax:

```
const osg::Vec3d& coordinate()const
{
   return theCoordinate;
   }
```

setModelCoordinate

Description:

Allows one to set the model coordinate.

Syntax:

```
void setModelCoordinate(const osg::Vec3d& modelCoordinate)
{
   theModelCoordinate = modelCoordinate;
   }
```

Parameter:

```
const osg::Vec3d& modelCoordinate
```

modelCoordinate – The model coordinate to set.

modelCoordinate

Description:

Returns the model coordinate.

Syntax:

```
const osg::Vec3d& modelCoordinate()const
{
   return theModelCoordinate;
   }
```

traverse

Syntax:

```
Virtual void traverse( osg::NodeVisitor& nv)
{
   if(theMatrixTransform.valid())
   {
         theMatrixTransform->accept(nv);
   }
   }
```

setMatrixTransform

Syntax:

```
void setMatrixTransform(osg::ref_ptr<osg::MatrixTransform> m)
{
   theMatrixTransform = m;
   }
```

matrixTransform

Syntax:

```
osg::ref_ptr<osg::MatrixTransform> matrixTransform()
{
   return theMatrixTransform;
   }
```

matrixTransform

Syntax:

```
const osg::ref_ptr<osg::MatrixTransform> matrixTransform()const
{
   return theMatrixTransform;
   }
```

ossimPlanetViewMatrixBuilder.h

```
#include <ossimPlanet/ossimPlanetViewMatrixBuilder.h>
```

The view matrix builder is located in *ossimPlanet/include/ossimPlanet/ossimPlanetViewMatrixBuilder.h*.

Description:

The view matrix builder serves several purposes. It can be used to allow one to do free form manipulation of the scene based on position and Euler heading pitch roll orientation. Also, one can use the class to "track" another node whether you are a fixed point looking at a moving object or a moving object looking at another moving object or a moving object looking at a fixed point or looking from a fixed point to a fixed point. To support all these scenarios we have decomposed the stages internally to build the matrix into self-contained parameters where each can be adjusted by a manipulator. Tracking in the true since of the word is really done by the ossimPlanetManipulator, which owns a ViewMatrixBuilder and listens for changes in nodes that either we are looking from or looking to and adjusts the parameters of the view matrix accordingly.

Public Member Functions

setLookFromNodeOffset

Description:

This method allows you to define a view that tracks a node. You can either sit at the node's defined center of mass or displace a distance "range" along line of site and/or 3d vector displacement. The relative orientation flag defines how the heading pitch roll argument is to be used to calculate the final orientation. If any flag is set for Heading, Pitch, and/or roll then that orientation parameter is relative to that orientation defined in the axis of the node. If any flag is not set then it is relative to the East North Up axis. So if you want a camera to always say rotate with the heading of that node then you will just set the HEADING flag in the relativeOrientationFlag.

Syntax:

```
enum OrientationFlags
   {
      NO_ORIENTATION = 0,
      HEADING        = 1,
      PITCH          = 2,
      ROLL           = 4,
      ALL_ORIENTATION = (HEADING|PITCH|ROLL)
   };

void setLookFromNodeOffset(osg::Node* node,
                           const osg::Vec3d& hpr,
                           double range,
                           int relativeOrientationFlag);
```

Parameters:

```
osg::Node* node
```

node - This is the node you wish this ViewMatrixBuilder to Lock to.

```
const osg::Vec3d& hpr
```

hpr - The passed in Heading Pitch and Roll to apply in angular degrees.

```
double range
```

range - The meter distance along the look axis.

```
int relativeOrientationFlag
```

relativeOrientationFlag - Which orientation parameter(s) do you wish to be relative to the Node's Axes. A value of ALL_ORIENTATION and an hpr parameter of all 0's will allow you to rotate with the object.

Examples:

No Orientation

```
thePlanetManipulator->viewMatrixBuilder()-
>setLookFromNodeOffset(thePointModel1.get(),
                       osg::Vec3d(0.0, 0.0, 0.0),
                       -range*10,
                       ossimPlanetViewMatrixBuilder::NO_ORIENTATION)
```

Heading

```
thePlanetManipulator->viewMatrixBuilder()-
>setLookFromNodeOffset(thePointModel1.get(),
                       osg::Vec3d(0.0, 0.0, 0.0),
                       -range*10,
                       ossimPlanetViewMatrixBuilder::HEADING)
```

Pitch

```
thePlanetManipulator->viewMatrixBuilder()-
>setLookFromNodeOffset(thePointModel1.get(),
                       osg::Vec3d(0.0, 0.0, 0.0),
                       -range*10,
                       ossimPlanetViewMatrixBuilder::PITCH)
```

Roll

```
thePlanetManipulator->viewMatrixBuilder()-
>setLookFromNodeOffset(thePointModel1.get(),
                       osg::Vec3d(0.0, 0.0, 0.0),
                       -range*10,
                       ossimPlanetViewMatrixBuilder::ROLL)
```

All Orientation

```
thePlanetManipulator->viewMatrixBuilder()-
>setLookFromNodeOffset(thePointModel1.get(),
                       osg::Vec3d(0.0, 0.0, 0.0),
                       -range*10,
```

```
                        ossimPlanetViewMatrixBuilder::All_ORIENTATION)
```

setLookFromLocalDisplacement

Description:

This allows for a look from displacement. So for instance if you want to position the camera truly in the cockpit and not the center of mass used for the local space transform you can do so. Currently this displacement will follow the local point center of mass relative to the orientation axis of the plane.

Syntax:

```
void setLookFromLocalDisplacement(const osg::Vec3d& displacement)
   {
      updateFromLocalDisplacement(displacement);
   }
```

Parameter:

```
const osg::Vec3d& displacement
```

displacement - This takes a local space displacement in meters and positions along the orientation of the node or current relative axis the amount to displace to the new center.

Example:

```
thePlanetManipulator->viewMatrixBuilder()-
>setLookFromNodeOffset(thePointModel1.get(),
                        osg::Vec3d(0.0, 0.0, 0.0),
                        -range*10,
                        ossimPlanetViewMatrixBuilder::All_ORIENTATION)

thePlanetManipulator->viewMatrixBuilder()-
>setLookFromLocalDisplacement(osg::Vec3d(0.0,0.0,0.0));
```

fromNode

Description:

Returns the current from node.

Syntax:

```
    osg::Node* fromNode()
    {
        OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
        return theFromNode.get();
    }
```

fromPositionLlh

Description:

This is the starting from position in lat lon height form. This does not include the displacement information or range information.

Return the from position in lat lon height.

Syntax:

```
    const osg::Vec3d& fromPositionLlh()const
    {
        OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
        return theFromPositionLLH;
    }
```

fromRelativeHpr

Description:

This is for the relative heading pitch and roll setting done in the set relative to a Node call. Also, the internal matrix builder will set the relative positioning if setting using the fixed point method.

Returns the realtive Hpr setting.

Syntax:

```
   const osg::Vec3d& fromRelativeHpr()const
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theFromRelativeHpr;
   }
```

fromRelativeOrientationFlags

Description:

Returns the current setting for the relative orientation flags.

Syntax:

```
   OrientationFlags fromRelativeOrientationFlags()const
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theFromRelativeOrientationFlags;
   }
```

computeFromOrientation

Description:

This will give a composited from orientation based on the Relative flags. Internally the source orientation could be changing each frame and the relative orientation offsets the source orientation vector based on the orientation flags. This is a utility function to allow one to call this in a single call.

Returns the composited orientation vector for heading pitch and roll in degrees.

Syntax:

```
osg::Vec3d computeFromOrientation()const;
```

fromHpr

Description:

This is the from source orietation vector.

Returns the from Heading pitch ad roll as a vec3d.

Syntax:

```
   const osg::Vec3d& fromHpr()const
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theFromHpr;
   }
```

fromRange

Description:

Returns the from range value in meters.

Syntax:

```
   Double fromRange()const
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theFromRange;
   }
```

fromNode

Description:

Returns the current from node as a constant value

Syntax:

```
   const osg::Node* fromNode()const
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theFromNode.get();
   }
```

setLookFrom

Description:

This method allow you to set fixed point to look from with a defined orientation and a displacement along the look as defined by the "range" parameter.

Syntax:

```
void setLookFrom(const osg::Vec3d& llh,
                 const osg::Vec3d& hpr,
                 double range);
```

Parameters:

```
const osg::Vec3d& llh
```

llh - This is the <lattitude, longitude, height> where the lat and lon are expressed in angular degrees and the height is in meters.

```
const osg::Vec3d& hpr
```

hpr - This is the <Heading, Pitch, Roll> parameters for the orietation axis relative to the local tangent plane defined by the point llh. The values are in angular degrees.

```
double range
```

range - This indicates the displacement from the axis origin along the line of site or look vector. Negative value displaces backwards from the look and positive displaces along the direction of the resulting look.

setLookToNode

Description:

If a look to node is specified some of the look from parameters are overridden since the orientation is dictated by the resulting look vector to the "look to" node that is passed in here.

Syntax:

```
void setLookToNode(osg::Node* node);
```

Parameter:

```
osg::Node* node
```

node - The node you wish to look to. It will take the node passed in and use it's center point information to define to help define a look direction as a vector going from the "from" location to the "to" node.

setLookTo

Description:

This is a method to define a fixed look to point.

```
void setLookTo(const osg::Vec3d& llh);
```

Parameter:

```
Const osg::Vec3d& llh
```

llh - This is a <lattitude, longitude, height> vector and the lat and lon are in angular degrees and the height is in meters.

setLookToLocalDisplacement

Description:

This allows for a look to displacement. This displaces along the look axis the amount along the x, y, and z. This is applied after the from axis orientation and positioning is performed.

Syntax:

```
void setLookToLocalDisplacement(const osg::Vec3d& dispalcement);
```

Parameter:

```
const osg::Vec3d& displacement
```

displacement - This takes a local space displacement in meters and positions along the orientation of the node or current relative axis the amount to displace to the new center.

setLookToRange

Description:

This allows one to specify a displacement along the look axis to the point you are looking. This is applied after the look to local displacement.

Syntax:

```
void setLookToRange(double range);
```

Parameter:

```
double range
```

range – The range to set.

setAttitudeHpr

Description:

This is the last applied transform and will generate the final orientation matrix for the view. This is done after a look from and to calculations and allows one to move the head around.

Syntax:

```
void setAttitudeHpr(const osg::Vec3d& hpr)
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      theAttitudeHpr = hpr;
      setComputeViewMatrixFlag(true);

   }
```

Parameter:

```
const osg::Vec3d& hpr
```

hpr - This is the relative heading pitch and roll about the final orientation axis.

attitudeHpr

Description:

Returns the attitude of the final look position. This is the Heading pitch and roll relative to the final orientation axes.

Syntax:

```
const osg::Vec3d& attitudeHpr()const
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theAttitudeHpr;
   }
```

toInformationSetFlag

Description:

Returns the to information set flag. A value of true specifies that the look to information was set.

Syntax:

```
bool toInformationSetFlag()const
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theToInformationSetFlag;
   }
```

toNode

Description:

Returns the current to node.

Syntax:

```
osg::Node* toNode()
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theToNode.get();
   }
```

toNode (constant)

Description:

Returns the current to node as a constant value.

Syntax:

```
const osg::Node* toNode()const
   {
```

```
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theToNode.get();
   }
```

toPositionLlh

Description:

This should only be called if a look to was set.

Returns the Look to position lat lon height as a Vec3d.

Syntax:

```
const osg::Vec3d& toPositionLlh()const
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theToPositionLLH;
   }
```

toDisplacement

Description:

This is a displacement along the current Rotational axis.

Syntax:

```
const osg::Vec3d& toDisplacement()const
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theToDisplacement;
   }
```

setLookAxis

Description:

This allows one to change the look axis you wish to look down. After the orientation is applied you can set the look axis you wish to use. This is not used if you are looking at another node or a fixed point for the look direction is already known. This is used only when you are in a from location only and will allow you to swap between axis to look down. So you can easily look up, down, left, right, ... etc

Syntax:

```
void setLookAxis(LookAxis axis)
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      theLookAxis = axis;
      setComputeViewMatrixFlag(true);
   }
```

Parameter:

```
LookAxis axis
```

axis - The axis to look down, see LookAxis enumeration for possible values.

lookAxis

Description:

Returns the current look axis.

Syntax:

```
LookAxis lookAxis()const
   {
      OpenThreads::ScopedLock<OpenThreads::Mutex>
lock(thePropertyMutex);
      return theLookAxis;
   }
```

viewMatrix

Description:

This matrix you can think of as placing objects out into the scene. Basically a local to world transform.

Returns the composited view Matrix as defined by the look to and from parameter settings.

Syntax:

```
const osg::Matrixd& viewMatrix()const;
```

inverseViewMatrix

Description:

This takes world coordinates and puts it into this local orientation. This is typically used to set a Camera Matrix. Call this to set a osg::Camera matrix or to set a manipulators setByMatrix.

Returns the inverse of the view matrix.

Syntax:

```
const osg::Matrixd& inverseViewMatrix()const;
```

isValid

Description:

Returns true or false if there is enough information to compute a matrix. We will need the geoRefModel for coodrinate transformations and at least the From point information needs to be set.

Returns true if a call to viewMatrix would be valid or false otherwise.

Syntax:

```
bool isValid()const{return
(theModel.valid()&&theFromInformationSetFlag);}
```

setParametersByInverseMatrix

Description:

See setParametersByMatrix.

This just wraps the call to the setParamtersByMatrix to invert the passed in matrix.

Syntax:

```
void setParametersByInverseMatrix(const osg::Matrixd& m)
   {
      osg::Matrixd newM(osg::Matrixd::inverse(m));
      setParametersByMatrix(newM);
   }
```

Parameter:

```
const osg::Matrixd& m
```

m – The inverted matrix to use.

setParametersByMatrix

Description:

This will take a matrix and flatten the parameters of the ViewMatrixBuilder to be purely a from unlocked type ViewMatrixBuilder. It will take the matrix passed in and extract out the eye position and orientation hpr. All other values such as range any to values will be zeroed out. All nodes that we were locked to will be nulled out.

Syntax:

```
void setParametersByMatrix(const osg::Matrixd& m);
```

Parameter:

```
const osg::Matrixd& m
```

m - The matrix to use to convert the ViewMatrixBuilder to.

convertToAFromViewMatrix

Description:

Converts the parameters to a from only. If the range is to be preserved by setting the flatten range flag to false and there is a to point that we are looking at then it will preserve the distance in the from range and setup a from only parameter ViewMatrixBuilder.

When using this call the Attitude orientation is not taken into account since this is used as a final displacement. So, the Attitude values are preserved but are removed from the orientation calculation for the new from HPR.

Syntax:

```
void convertToAFromViewMatrix(bool flattenRangeFlag = false);
```