OpenScenGraph's Virtual Planet Builder/OSGDEM

Virtual Planet Builder (VPB) is OSG's terrain-database-building tool. It loads GIS elevation data for a region, combines it with imagery and produces a spatially-balanced quadtree tiled database that can be navigated simply by loading the top-level file. All of the LOD Ranges are pre-configured and the DatabasePager automatically loads and unloads different segments of terrain without any supporting userspace code required. VPB can build terrain models ranging from a small area to the entire planet. These databases can be fluidly navigated at 60fps from outer space down to ground level with no interruptions. The terrain models are built of regular grid arrays with optional edge-skirts to conceal any cracks caused by adjacent LOD differences.

Advantages

The core OSG HeightField, osgTerrain and even Geometry/PrimitiveSet classes allow you to build terrain-like surfaces yourself without using VPB at all. However, there are economies of scale that are very significant to maximum OpenGL performance. OpenGL prefers geometric data to be delivered to the hardware pipeline in batches of moderate size. Batches that are too small have more overhead setting up and cleaning up from each batch, and stall the pipeline. Batches that are too large may end up delivering data that extends outside the viewing frustum, making the hardware do needless work on data that is not needed. Intersection testing also performs poorly on tiles with too many data cells.

VirtualPlanetBuilder breaks all terrain up into grids and subgrids of an optimal size (which can be overridden at database-build time should hardware economies change in the future). Every piece that exceeds the threshold will be automatically split into sub-pieces. Each of these sub-pieces will be replaced by a larger number of smaller, higher-detail sub-pieces as more detail is called for. Optimally, a VPB database displays nearly a constant amount of polygons as you zoom in or out. Un-necessary surrounding data is culled or represented with lower detail as important in-view data nearby is loaded as higher and higher levels of detail.

VPB databases and the DatabasePager can even handle multiple cameras with multiple independent views simultaneously roaming a single large terrain model without hiccuping. Re-creating this scheme yourself is both difficult, time-consuming and pointless. This is the purpose VPB was made for. If you plan to do multi-resolution terrain models in OSG, use VPB.

Build process

VPB consists of two main tools, OSGDEM and VPBMaster. OSGDEM is what actually builds terrain datasets from input data. VPBMater is a controller tool that coordinates one or more runs of OSGDEM on one or more CPUs/cores on one or more computers simultaneously to finish the database building sooner. Simple examples will use OSGDEM here.

A basic OSGDEM execution might look like:

```
osgdem.exe --geocentric -t <imagefile> -d <terraindatafile> --terrain
-o OutputFile.ive
```

The --geocentric option, instructs osgdem to build a round-earth model with its origin at the center of the Earth instead of a flat-Earth CAD-like database. The --terrain option tells OSGDEM to output HeightField data arrays that will be instantiated into regular tesselated grids when the dataset is viewed.

Sample data

The sample data used here is from the USGS Seamless site, and consists of a large area high-resolution color orthophoto on top of a similar extent of moderate detail terrain DEM data. The area in question is near Rollinsville, Colorado where the Moffat Tunnel goes under the Continental Divide to Winter Park.

NED30mLarge.bil (6.23Mb)/NED30m.bil (53K), Rollinsville1m.tif (1Gb)/RV1mOrtho.tif(7.9Mb)

```
osgdem -t Rollinsville1m.tif -d NED30mLarge.bil -l 4 -v 1.0 --terrain -o RVSample.ive
osgdem -t Small\RV1mOrtho.tif -d Small\NED30m.bil -gb -105.6544254866 39.8999140007
-105.6255360338 39.9079734079 -l 4 -v 1.0 --terrain -o MoffetSample.ive
```

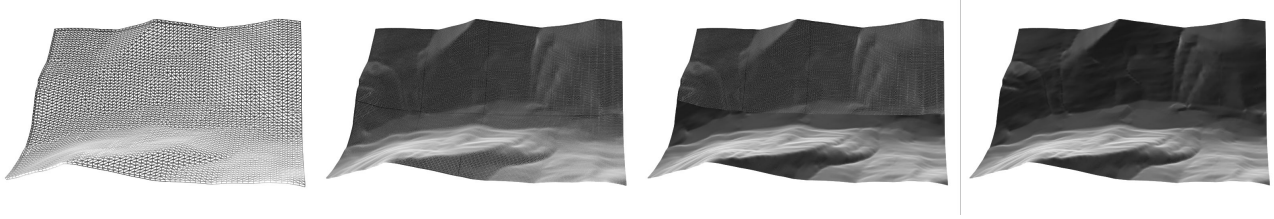
Terrain Theory

Structure of a typical VirtualPlanetBuilder/OSGDEM terrain database:

Root OSG file looks like:
CoordinateSystemNode: Top-level, defines ellipsoid shape of globe
- PagedLOD
  - Child0:TerrainTile (lowest LOD of terrain model)
  - Child1:Basename_root_L0_X0_Y0/Basename_L0_X0_Y0_subtile.osg
    (file containing LODs that cover the same extent as Child 0)

A VPB TerrainTile (for example, L0_X0_Y0) looks like:
TerrainTile (lowest LOD of terrain model)
- Locator (defines Coordinate System and Transform of TerrainTile)
- ElevationLayer
  - HeightFieldLayer
    - HeightField: UniqueID, Origin, X/Y Interval, SkirtHeight, Heights [Array]
- ColorLayer
  - ImageLayer
    - file Basename_L0_X0_Y0.dds

SubTile Basename_root_L0_X0_Y0/Basename_L0_X0_Y0_subtile.osg looks like:
Group
- PagedLOD
  - Child0: TerrainTile
  - Child1: Basename_L1_X0_Y0_subtile.osg (Another subtile like this one)
- PagedLOD
  - Child0: TerrainTile
  - Child1: Basename_L1_X1_Y0_subtile.osg
- ...more, similar PagedLODs may exist here...



In operation, the Root OSG file is loaded by the caller. This file contains a TerrainTile with Color and Elevation data, so as soon as the initial root load is complete, there is a lo-resolution terrain surface visible. OSG immediately begins cull traversal of this scene graph. During cull traversal, the PagedLOD node probably discovers that the currently visible child (child 0, with the TerrainTile) is insufficient LOD for the current view. The only alternate LOD is Child1, which is an external file. PagedLOD informs the DatabasePager to begin loading this external file and continues on the cull. The load operation has a priority level that hints to the DatabasePager how important this node is, allowing multiple loads to be queued according to how close each one is to the viewer.

After the first cull operation is complete, a number of cull/draw cycles may execute while the DatabasePager thread completes loading and possibly compiling the loaded data for use in the scenegraph. During each cull operation that the loaded external child is not yet available, the PagedLOD re-requests the child node to be loaded. This allows the priority of the node in the queue to adjust if the viewpoint moves prior to the node getting loaded.

When ready, the DatabasePager inserts the loaded subgraph (see structure of Basename_L0_X0_Y0_subtile.osg) into the scenegraph. At this point, the top PagedLOD is satisfied that it has an appropriate LOD, and subsequent cull operations will traverse the Child1 of the top PagedLOD Node.

Child1 itself contains a non-trivial subgraph. At the top of the subgraph are several PagedLOD nodes. Each of these contains its own Child0 LOD, which is not an external file but a TerrainTile complete with ElevationLayer and ColorLayer already loaded in and compiled as part of the Basename_L0_X0_Y0_subtile. So, when the Root-level Child0 TerrainTile is replaced with the root-level Child1 external file, the Basename_L0_X0_Y0_subtile's Child0s immediately cover the exact same visual extent of terrain without any more loading delay.

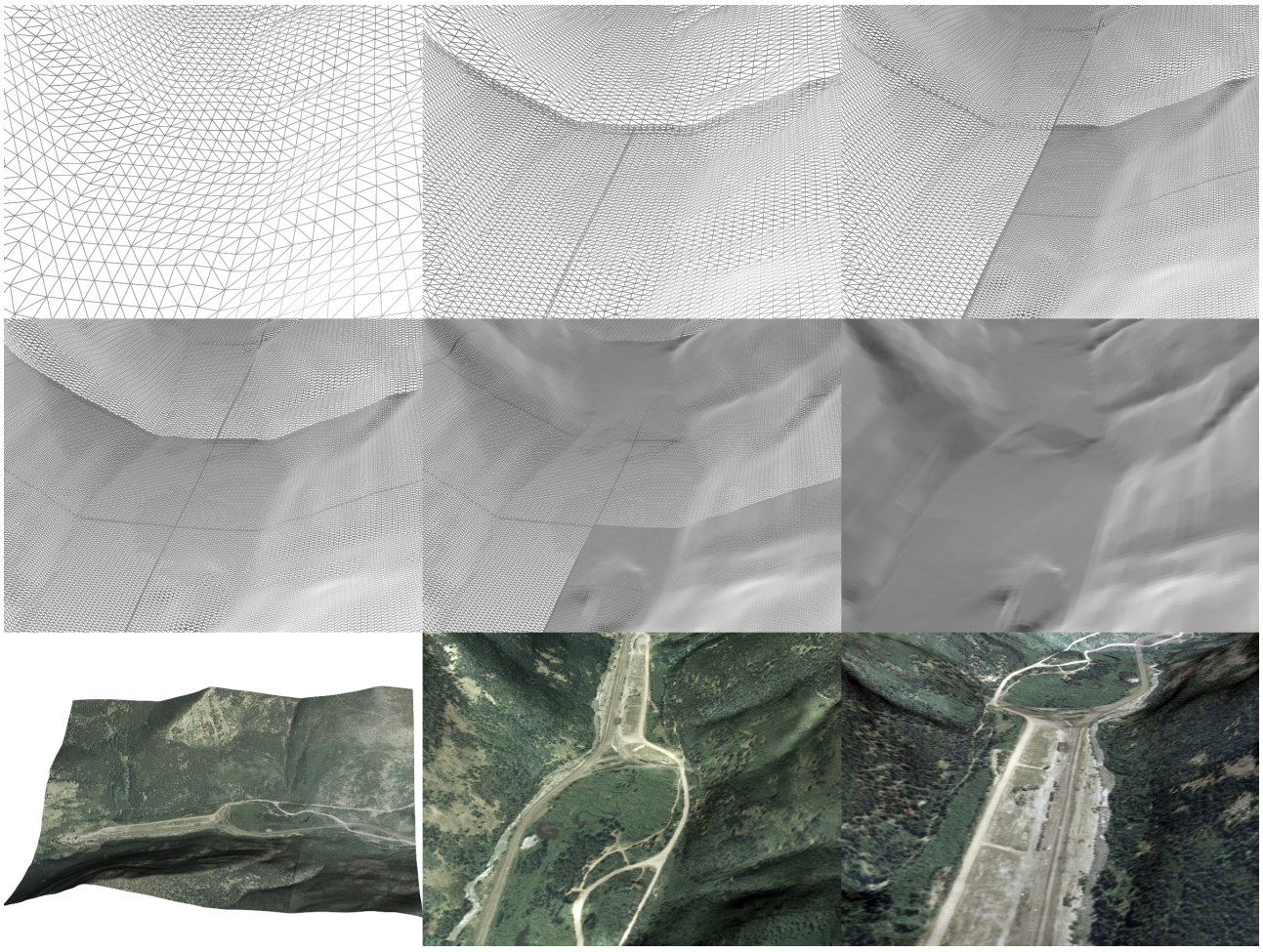
As soon as the Basename_L0_X0_Y0_subtile subgraph is merged into the scenegraph and traversed by cull, the PagedLODs in L0_X0_Y0 will evaluate their LOD criteria (typically expressed not as actual distance, but in realtive-apparent-size-on-screen) and most likely discover they too, are insufficient for the current view. At this point, each of the PagedLODs will decide to load their next LOD child (Child1). Because each of these children will be located at different locations with respect to the viewer, they will each individually determine a different priority, which the DatabasePager will respect when queuing the DatabaseRequest.

PagedLOD class has several conditions. The base LOD class and the Group class beneath that require that the container (a std::vector) that contains the actual LODs must be contiguous – no empty slots are permitted in the container. The PagedLOD class defines a parallel container called PerRangeDataList, which is also a vector holding the LOD ranges and external filenames for each child that can be loaded. The contents of these two containers are associated only by the subscript of their contents. What this means in practice is that the LOD children in the Group must be loaded contiguously – no higher LOD can be loaded if a lower LOD is not already loaded. PagedLOD will always load LOD #0 first. If that LOD is insufficient, it will load #1. It will continue loading LODs, in order, one at a time, until it reaches one that is sufficient. This is because the LOD and Group classes cannot permit "gaps" in the container of Group children, and the PagedLOD PerRangeDataList must load children into the Group child slot corresponding with the PerRangeData.

VPB-created databases never have PagedLODs with more than two children. The first (Child 0) is always an "internal" TerrainTile that is loaded completely along with the PagedLOD itself. It has a blank "external" filename in its PerRangeData (to keep the order of PerRangeDataList and the Group child container in sync). The second child (Child1) supplies no "internal" representation. It has an external filename that is loaded by the DatabasePager.

Lower LODs are not ever unloaded because LOD class wants to be able to immediately switch to a lower (and faster-performing) LOD without needing to wait for loading the lower LOD. This is the conservative approach so that OSG won't have both a high-detail tile just loaded, but still be showing a different, too-high-detail tile elsewhere while it waits for a lower detail tile to load. Having too much unintended simultaneous detail loaded at once could cause degraded redraw performance and break frame.

Child #0 cannot be unloaded or reloaded because it is not an external file. In order to unload it, the actual PagedLOD it is embedded within (and possibly the file that PagedLOD is contained within, which might itself contain several other PagedLODs) would need to be unloaded/reloaded.



Optimizing

Tuning LODScale at runtime
Using -O "compressed" option to compress texture maps (may not work on OpenGL ES)
Cropping area generated to be smaller than input data coverage
`"-gb xA yA xB yB"`

Data sources

USGS Seamless site (US 30m DEM, very good quality, also color orthoimagery)
http://seamless.usgs.gov/index.php
USDA NAIP (color orthoimagery)
http://datagateway.nrcs.usda.gov/aboutLightHouse.html
Landcover.org (Landsat 30m, global, needs processing)
http://landcover.org/index.shtml
TelaScience Landsat (30m, less processing needed)
http://collections.sdsc.edu/dac2/telascience/telascience_data/onearth/
TelaScience SRTM (90m global DEM, some defects):
http://hypersphere.telascience.org/elevation/cgiar_srtm_v4/tiff/zip/
ASTER DEM (global DEM, some defects)
http://asterweb.jpl.nasa.gov/gdem-wist.asp

Input formats

GDAL can read almost anything, so see what kinds of data are available to you. Data that does not have georeferencing are rare in the GIS world – check your data sources if you encounter this. It's often a sign of sloppy processing or possibly other more serious issues like using non-GIS tools that may have damaged the precision or accuracy in other ways. If you must use non-georeferenced data, OSGDEM lets you manually specify the extent of each data source

Output File Formats

OSGB or IVE preferred. Typically write to OSGB/IVE, you can convert to .OSG later if necessary. OSGB and IVE format files embed the imagery within the main file.

Scaling VPB to Huge Datasets

Cache, VPBMaster
Pre-add overviews (source data mipmaps) with gdaladdo

Placing objects on Terrain

Use osg::EllipsoidModel and osgSim::HeightAboveTerrain

Determine Height above terrain at X,Y point:

```
osgSim::HeightAboveTerrain terrainHeight;
terrainHeight.addPoint();
terrainHeight.computeIntersections();
```

Orient model to local up vector for this location of the world with EllipsoidModel:

```
osg::MatrixTransform matrixPosPlusRot;
osg::EllipsoidModel ellipseModel;
ellipseModel.convertLatLongHeightToXYZ();
ellipseModel.computeLocalUpVector();
matrixPosPlusRot.setMatrix();
```

Add object as child of matrixPosPlusRot, then add matrixPosPlusRot to root of scenegraph.

Mass-Placing Objects on Terrain

Using post-load callback to populate the terrain.
Iterate list of coordinate, object pairs, using Ellipsoid and HAT to place each one that falls within the bounds of the current terrain tile.