osgEarth Documentation

Release 3.2

Pelican Mapping

Aug 25, 2022

Contents

osgEarth is a 3D mapping SDK for OpenSceneGraph.

- Get a 2D or 3D map up and running quickly and easily using one simple XML file
- Visualize massive amounts of imagery, elevation data, and vector features
- Use open-standards map data services like WMS, TMS, and GeoTIFF
- Work locally or over the Internet
- Develop applications using a full-featured C++11 API

System Requirements

- Windows 10, Linux, or Mac OSX
- OpenGL 3.3+ compatible GPU with 8+ GB

Let's get started!

CHAPTER 1

Installing osgEarth

1.1 Windows

You can install the osgEarth SDK and toolset using vcpkg.

First, download and bootstrap vcpkg following the instructions on the web site.

Then you can install osgEarth like so:

```
vcpkg install osgearth:x64-windows
```

CHAPTER 2

Building osgEarth

The documentation here is focused on Windows.

2.1 Building with vcpkg

vcpkg is an extremely useful package manager. It works on Windows, Linux and MacOS but for this guide we will focus on Windows.

Step 1 - Configure vcpkg

First, download and bootstrap vcpkg following the instructions on the page.

Next install the dependencies required to build a fully functional osgEarth. This example assume s 64-bit Windows build; you can alter that to correspond to your platform/architecture of choice.

Install the required dependencies:

```
vcpkg install osg:x64-windows gdal:x64-windows curl:x64-windows
```

For full functionality, you can install optional dependences as well:

```
vcpkg install sqlite3:x64-windows protobuf:x64-windows geos:x64-windows blend2d:x64-
↪windows libwebp:x64-windows basisu:x64-windows draco:x64-windows libzip:x64-windows
```

This will take awhile the first time you run it as this pulls down lots of dependencies, so go get a cup of coffee.

Once all the dependencies are built, you'll need to actually build osgEarth.

Step 2 - Clone the repository

Pull down the source from GitHub and create a `build` folder for your out-of-source build. We always recommend doing an out-of-source build to avoid problems down the road!

```
git clone https://github.com/gwaldron/osgearth.git
mkdir build
```

Step 3 - Configure CMake

vcpkg provides a CMake toolchain file that helps osgEarth find all of its dependencies.

Note: You'll need to specify a different build directory based on your build configuration (Release, RelWIthDebInfo, Debug) and specify the build type using `-DCMAKE_BUILD_TYPE`. This is because some dependencies of osgEarth don't pick up both debug and release versions without specifying the build type. Hopefully this will be fixed in future CMake versions.

Most developers will use a RelWithDebInfo build, like so:

```
cmake -S osgearth -B build -G "Visual Studio 15 2017 Win64" -DCMAKE_BUILD_
↪TYPE=RelWithDebInfo -DWIN32_USE_MP=ON -DCMAKE_INSTALL_PREFIX=[installroot] -DCMAKE_
↪TOOLCHAIN_FILE=[vcpkgroot]\scripts\buildsystems\vcpkg.cmake
```

Step 4 - Build and install osgEarth

You can build and install osgEarth on the command line using CMake or you can open up the Visual Studio solution and build it from there.

```
cmake --build build --target INSTALL --config RelWithDebInfo
```

Step 5 - Set up your runtime environment

You'll need to make sure that the vcpkg dependencies and osgEarth are in your path:

```
set PATH=%PATH%;c:\vcpkg\installed\x64-windows\bin
set PATH=%PATH%;c:\vcpkg\installed\x64-windows\tools\osg
set PATH=%PATH%;[installroot]
```

2.2 Building for OpenGL CORE Profile

You may wish to build osgEarth with support for the OpenGL CORE profile. In fact is a requirement for some platforms including Apple OSX and VMWare. Doing to requires that you first build OpenSceneGraph with CORE profile support. The OpenSceneGraph dependency in *vcpkg* does NOT have GLCORE support (at the time of this writing) so you will have to build it yourself.

2.2.1 Build OpenSceneGraph for GLCORE

1. First, download the GL CORE include files from Khronos and place them somewhere on your system. We'll call this the GLCORE folder.
2. In CMake, set the `OPENGL_PROFILE` property to "GLCORE".
3. In CMake, set the `GLCORE_GLCOREARB_HEADER` property to the location of the GL folder you downloaded from Khronos. For example, if you include file is at `C:\glcore\GL\glcorearb.h` you should set this property to `C:\glcore`.
4. In CMake, set the following properties to `ON` :
   - `OSG_GL3_AVAILABLE`
5. In CMake, set the following properties to `OFF` :
   - `OSG_GL1_AVAILABLE`
   - `OSG_GL2_AVAILABLE`
   - `OSG_GLES1_AVAILABLE`

- OSG_GLES2_AVAILABLE
- OSG_GL_DISPLAYLISTS_AVAILABLE
- OSG_GL_FIXED_FUNCTION_AVAILABLE
- OSG_GL_MATRICES_AVAILABLE
- OSG_GL_VERTEX_ARRAY_FUNCS_AVAILABLE
- OSG_GL_VERTEX_FUNCS_AVAILABLE

6. Configure and build OpenSceneGraph.

2.2.2 Build osgEarth for GLCORE

Now that you have OSG built with GLCORE support, time to build osgEarth.

1. In CMake, set the OSGEARTH_GLCORE_INCLUDE_DIR property to the same folder holding the Khronos include files (the same value of the GLCORE_GLCOREARB_HEADER in your OSG build).
2. Configure and build osgEarth.

Test you build by running this on the command line (Windows)

```
set OSG_GL_CONTEXT_VERSION=4.6
osgearth_version --caps
```

If all went well, it should report **"Core Profile = yes"**.

You can disable the CORE profile and select a compatibility profile by setting a profile mask like so

```
set OSG_GL_CONTEXT_PROFILE_MASK=1
```

The context version and profile mask are also settable via the osg::DisplaySettings class in the OpenSceneGraph API.

2.3 Tips for VMware Users

Running osgEarth in a virtual machine environment can be tricky since they usually don't have direct access to the graphics hardware by default. If you are having trouble you can try these tips.

First, build OSG and osgEarth for GL CORE profile (as above).

Next, assess the situation with a capabilities check:

```
osgearth_version --caps
```

The output will look something like this:

```
GPU Vendor:         WMware, Inc.
GPU Renderer        Gallium 0.3 on llvmpipe
GL/Driver Version: 1.2 Mesa 11.2.0
```

If is reports a Mesa driver, and the version is less than 3.3, you will need to configure a couple environment variables to move forward (Windows):

```
set OSG_GL_CONTEXT_VERSION=3.3
set MESA_GL_VERSION_OVERRIDE=3.3
osgearth_version --caps
```

Good luck!

CHAPTER 3

The Earth File

osgEarth uses the familiar **Map/Layer** paradigm for organizing data. The **Map** is comprised of a collection of **layers**. The renderer draws each visible layer one after the next, from bottom to top, to display the final scene. *You can see all the different layer types here*.

An **Earth File** is an XML file that describes the contents of a **Map** in osgEarth.

3.1 My First Earth File

Here is a very simple earth file that you can find in the `tests` folder of the repository:

```
<Map name="Hello, World">
    <GDALImage name="World imagery">
        <url>../data/world.tif</url>
    </GDALImage>
</Map>
```

This map contains one layer that points to a local GeoTIFF file. In this case, the location is relative to the location of the earth file itself. You can see this map by running one of the osgEarth command line tools:

```
osgearth_toc simple.earth
```

That's it! It is that easy to get a map up and running.

3.2 Using Multiple Layers

You can add as many layers to your **Map** as you like. Here's an example called `hires-inset.earth`:

```
<Map name="Hello, World">
    <!-- Worldwide image -->
    <GDALImage name="World">
```

(continues on next page)

(continued from previous page)

```
        <url>../data/world.tif</url>
    </GDALImage>

    <!-- Higher resolution inset of Boston -->
    <GDALImage name="Boston">
        <url>../data/boston-inset-wgs84.tif</url>
    </GDALImage>

    <!-- Higher resolution inset of New York City -->
    <GDALImage name="New York">
        <url>../data/nyc-inset-wgs84.tif</url>
    </GDALImage>
</Map>
```

In this case, osgEarth will draw the "World" layer first, followed by "Boston" and finally "New York" on top. We also interspersed some XML comments in there.

3.3 Adding Terrain Elevation Data

Now let's add some height field data, also knows as a **DEM** or Digital Elevation Model:

```
<Map name="ReadyMap">
    <TMSImage name="ReadyMap 15m Imagery">
        <url>http://readymap.org/readymap/tiles/1.0.0/7/</url>
    </TMSImage>

    <TMSElevation name="ReadyMap 90m Elevation">
        <url>http://readymap.org/readymap/tiles/1.0.0/116/</url>
        <vdatum>egm96</vdatum>
    </TMSElevation>

    <Viewpoints>
        <Viewpoint name="San Francisco, California">
            <heading>10/heading>
            <height>4500.0</height>
            <lat>37.5581</lat>
            <long>-122.334</long>
            <pitch>-34</pitch>
            <range>78000</range>
        </Viewpoint>
    </Viewpoints>
</Map>
```

A couple things going on here: first we see a `TMSImage` layer that loads imagery from a Tile Map Service layer over the Internet. Then we have some 90m digital elevation data coming from the same server.

Finally you can see a `Viewpoints` layer. This is not a visible layer at all! Instead if stores data - in this case, a set of pre-set viewpoints the user can navigate to. osgEarth stores all kinds of data as **layers**, not all of it visible. Non-visible layers can occur anywhere in the map. Their order or appearance does not matter.

3.4 Drawing Vector Features

Finally, let's look at some vector feature data. This is GIS data in the form of points, lines, and polygons. osgEarth has various methods of displaying this data, but let's keep it simple for now. You can find this one in `feature_rasterize.earth`:

```
<Map name="Rasterize Vectors">
    <xi:include href="readymap_imagery.xml"/>

    <OGRFeatures name="world-data">
        <url>../data/world.shp</url>
    </OGRFeatures>

    <FeatureImage name="Countries" opacity="0.75">
        <features>world-data</features>
        <styles>
            <style type="text/css">
                default {
                    fill:          #ff7700;
                    stroke:        #ffff00;
                    stroke-width:  5km;
                }
            </style>
        </styles>
    </FeatureImage>
</Map>
```

First, we see the use of `<xi:include>`, a handy XML directive to include another XML file inline. We're using this to get our imagery.

Next is the `OGRFeatures` layer. This is a data layer - it won't be rendered directly - but rather it just points to an ESRI Shapefile we have stored locally.

Finally, the `FeatureImage` layer points at the data layer and describes how to draw it using a `StyleSheet`.

3.5 More Examples

Please look in the `tests` folder of the repository for lots of examples of earth files. They range from very simple to quite complex and cover a wide range of the available functionality in osgEarth!

CHAPTER 4

Working with Data

osgEarth can load many different open standard data formats of imagery, elevation data, and vector features. Read on for tips on preparing your data to get the best results.

4.1 Preparing your Data

osgEarth needs to chop the source data up into grids of "tiles" for display. You can usually load data directly from source with no pre-processing, but if the data is not optimized, it might be slow! So a little pre-processing is usually a good idea when using large datasets.

Load you data directly first. If it's fast enough, you are good to go! Otherwise, here are some tips to optimize your data for tiled access. There are two ways to approach it: optimizing a GeoTIFF, or building tilesets. We will also discuss how to combine a folder full of data into a single layer for use in osgEarth.

4.1.1 Optimizing a GeoTIFF

GeoTIFF is the most common format for local imagery or elevation data. Here are some steps you can take to speed up osgEarth's access to GeoTIFFs.

Reproject your imagery

osgEarth will reproject your data on-the-fly if it does not have the necessary coordinate system. For instance, if you are trying to view a UTM image on a geodetic globe (epsg:4326), osgEarth needs to do that conversion on the fly – but doing it offline beforehand will be faster.

You can use any tool you want to reproject your data such as GDAL, Global Mapper or ArcGIS.

For example, to reproject a UTM image to geodetic using `gdal_warp`:

```
gdalwarp -t_srs epsg:4326 my_utm_image.tif my_wgs84_image.tif
```

Build internal tiles

Typically formats such as GeoTiff store their pixel data in scanlines. However, using a tiled dataset will be more efficient for osgEarth because of how it uses tiles internally.

To create a tiled GeoTiff using `gdal_translate`, issue the following command:

```
gdal_translate -of GTiff -co TILED=YES input.tif output.tif
```

Take it a step further and use compression to save space. You can use internal JPEG compression if your data contains no transparency:

```
gdal_translate -of GTiff -co TILED=YES -co COMPRESS=JPG input.tif output.tif
```

Build overviews

Adding overviews (also called ''pyramids" or ''rsets") can sometimes increase the performance of a large data source in osgEarth. You can use the gdaladdo utility to add overviews to a dataset:

```
gdaladdo -r average myimage.tif 2 4 8 16
```

Spatial indexing for feature data

Large vector feature datasets (e.g., shapefiles) will benefit greatly from a spatial index. Using the `ogrinfo` tool (included with GDAL/OGR binary distributions) you can create a spatial index for your vector data like so:

```
ogrinfo -sql "CREATE SPATIAL INDEX ON myfile" myfile.shp
```

For shapefiles, this will generate a ".qix" file that contains the spatial index information.

4.1.2 Building Tilesets

Pre-tiling your imagery can speed up load time dramatically, especially over the network. In fact, if you want to serve your data over the network, this is the only way!

`osgearth_conv` is a low-level conversion tool that comes with osgEarth. One useful application of the tool is tile up a large GeoTIFF (or other input) in a tiled format. Note: this approach only works with drivers that support writing (MBTiles, TMS).

To make a portable MBTiles file:

```
osgearth_conv --in driver GDALImage --in url myLargeFile.tif
              --out driver MBTilesImage --out filename myData.mbtiles
              --out format jpg
```

If you want to serve tiles from a web server, use TMS:

```
osgearth_conv --in driver GDALImage --in url myLargeData.tif
              --out driver TMSImage --out url myLargeData/tms.xml
              --out format jpg
```

That will yield a folder (called "myLargeData" in this case) that you can deploy on the web behind any standard web server like Apache.

Tip: The `jpg` format does NOT support transparency. If your data has an alpha channel, use `png` instead.

Just type `osgearth_conv` for a full list of options. The `--in` and `--out` options correspond directly to properties you would normally include in an Earth file.

4.1.3 Loading a Directory of Files

Sometimes the data you have will consist of lots of individual files that make up a single dataset. DTED elevation data is a common example of this. Instead of loading each individual file into osgEarth as a separate layer, it is best to combine them into one "virtual" dataset.

Use the GDAL `gdalbuildvrt` utility to create a VRT. A VRT is a "virtual format" that combines multiple files into a single data source.

Say you have a folder full of ".dt1" files. You can create a single layer like so:

```
gdalbuildvrt output.vrt *.dt1
```

Now you can just load `output.vrt` directly in osgEarth, like so:

```
<GDALElevation name="My DTED data">
    <url>output.vrt</url>
    <vdatum>egm96</vdatum>
</GDALElevation>
```

4.2 Where to Find Data

Help us add useful sources of freely available data to this list. Always check on attribution and distribution requirements from the provider when using 3rd party data!

Raster data

- ReadyMap.org - 15m imagery, 90m elevation, and street tiles for osgEarth developers. Free for development and demo purposes only.
- USGS National Map - Elevation, orthoimagery, hydrography, geographic names, boundaries, transportation, structures, and land cover products for the US.
- NASA BlueMarble - NASA's whole-earth imagery (including topography and bathymetry maps)
- Natural Earth - Free vector and raster map data at various scales
- Bing Maps - Microsoft's worldwide imagery and map data ($)

Elevation data

- CGIAR - World 90m elevation data derived from SRTM and ETOPO (CGIAR European mirror)
- GEBCO - General Bathymetry Chart of the Oceans

Feature data

- OpenStreetMap - Worldwide, community-sources street and land use data (vectors and rasterized tiles)
- Natural Earth - Free vector and raster map data at various scales
- DIVA-GIS - Free low-resolution vector data for any country

CHAPTER 5

osgEarth Layers

These are the public layer types native to osgEarth.

5.1 Raster Data

5.2 Vector Data

5.3 Miscellaneous Layers

5.4 Feature Sources

CHAPTER 6

FAQ

6.1 Using the API

6.1.1 How do I place a 3D model on the map?

The `osgEarth::GeoTransform` class inherits from `osg::Transform` and will convert map coordinates into OSG world coordinates for you. Place an object at a geospatial position like this:

```
GeoTransform* xform = new GeoTransform();
GeoPoint point(srs, -121.0, 34.0, 1000.0);
xform->setPosition(point);
```

If you want your object to automatically clamp to the terrain surface, assign a terrain and leave off the altitude:

```
GeoTransform* xform = new GeoTransform();
xform->setTerrain(mapNode->getTerrain());
GeoPoint point(srs, -121.0, 34.0);
xform->setPosition(point);
```

6.1.2 Why does my model have no texture or lighting?

Everything under an osgEarth scene graph is rendered with shaders. So, when using your own models (or creating geometry by hand) you need to create shader components in order for them to render properly.

osgEarth has a built-in shader generator for this purpose. Run the shader generator on your node like so:

```
osgEarth::Registry::shaderGenerator().run( myNode );
```

After that, your node will contain shader snippets that allows osgEarth to render it properly and for it to work with other osgEarth features like sky lighting.

6.1.3 Why are my Lines or Annotations not rendering?

Lines render using a shader that requires some initial state to be set. You can apply this state to your top-level camera (or anywhere else above the geometry) like so:

```
#include <osgEarth/GLUtils>
...
GLUtils::setGlobalDefaults(camera->getOrCreateStateSet());
```

For Annotations (`FeatureNodes`, `PlaceNodes`, etc.) best practice is to place an Annotation node as a descendant of the `MapNode` in your scene graph. You can also add them to an `AnnotationLayer` and add that layer to the Map.

Annotations need access to the `MapNode` in order to render properly. If you cannot place them under the `MapNode`, you will have to manually install a few things to make them work:

```
#include <osgEarth/CullingUtils>
#include <osgEarth/GLUtils>
...

// Manully assign the MapNode to your annotation
annotationNode->setMapNode(mapNode);

// In some group above the annotation, install this callback
group->addCullCallback(new InstallViewportSizeUniform());

// In some group above the annotation, set the GL defaults
GLUtils::setGlobalDefaults(group->getOrCreateStateSet());
```

Again: MapNode does all this automatically so this is only necessary if you do not place your annotations as descendants of the MapNode.

6.1.4 Text annotations (LabelNode, PlaceNode) are not rendering. Why?

Rendering text requires that you disable OSG's small-feature culling like so:

```
view->getCamera()->setSmallFeatureCullingPixelSize(-1.0f);
```

Note: you must do this for each camera.

6.2 Community and Support

6.2.1 What is the best practice for using GitHub?

The best way to work with the osgEarth repository is to make your own clone on GitHub and to work from that clone. Why not work directly against the main repository? You can, but if you need to make changes, bug fixes, etc., you will need your own clone in order to issue Pull Requests.

1. Create your own GitHub account and log in.
2. Clone the osgEarth repo.
3. Work from your clone. Sync it to the main repository periodically to get the latest changes.

6.2.2 How do I submit changes to osgEarth?

We accept contributions and bug fixes through GitHub's Pull Request mechanism.

First you need your own GitHub account and a fork of the repo (see above). Next, follow these guidelines:

1. Create a *branch* in which to make your changes.
2. Make the change.
3. Issue a *pull request* against the main osgEarth repository.
4. We will review the *PR* for inclusion.

If we decide NOT to include your submission, you can still keep it in our cloned repository and use it yourself. Doing so maintains compliance with the osgEarth license since your changes are still available to the public - even if they are not merged into the master repository.

6.2.3 Can I hire someone to help me with osgEarth?

Of course! We at Pelican Mapping are in the business of supporting users of the osgEarth SDK and are available for contracting, training, and integration services. The easiest way to get in touch with us is through our web site contact form.

CHAPTER 7

Release Notes

7.1 Version 3.2 (August 2021)

This is primarily a performance and bug-fix release.

Release Highlights:

- New ImGui integration, including the new `osgearth_imgui` command line tool.
- `ObjectIDPicker`, a more reliable replacement for `RTTPicker`
- `ContourMap` has new for custom color stops in the earth file
- Build system now uses Git Submodules for some inline dependencies

7.2 Version 3.1 (December 2020)

As of this release, osgEarth requires C++11.

GEOS: We transitioned from the GEOS C++ API to the C API for stability reasons. If you see GEOS compile/linker errors, this is likely the reason and you should make sure to link with the C library from now on. (GEOS is an optional dependency that enables some feature processing operations.)

Release Highlights:

- New `TerrainConstraintLayer` for masking and custom terrain tessellation. Please see `constraints.earth` for sample applications. This replaces and extends the old `MaskLayer` type.
- New `LERCImageLayer` (ESRI format)
- New `ArcGISTilePackageElevationLayer`
- New `ArcGISServerElevationLayer`
- New `DebugImageLayer` `show_tessellation` property to display the terrain mesh; handy for visualizing constraints created with the new `TerrainConstraintLayer`

- Map-wide default texture compression setting allows you to enable automatic texture compression in the terrain options section of your earth file
- SONAME for Linux builds is now properly used. This was preventing ABI stability for some package managers.
- `osgearth_conv` supports a geocell index (`--index`) that can greatly increase the performance of a large tiling operation by implementing a gridded spatial index.
- New `SelectExtentTool` for drawing a bounding box on the map and firing a callback.
- XYZ layers now support the {-y} notation for Y inversion, a common notation used in web mapping URLs
- Write support for `TMSElevationLayer`
- Improved task cancellation support throughout. Task cancellation occurs when the results of data-loading task are no longer required (because the camera moved) and we want to cut short the operation.
- Improved polygon tessellation (fixes various edge cases)
- Faster and better vector rasterization using the excellent `Blend2D` library (optional dependency)Transitioned to the GEOS C API for stability
- Improved parallelization for some drivers
- Mutex contention analysis in Tracy - helps us identify and mitigate contention to improve parallelization
- Normalized on C++11 threading primitives, almost completely removing the dependency on OpenThreads
- Various speed improvements and bug fixes
- Simplified CMake configuration process
- Support for GDAL 3.1 (mostly - see the spherical mercator note in SRS.cpp)
- Refreshed the documentation site
- GitHub actions for CI on Linux, Windows, and MacOS

7.3 Version 3.0 (June 2020)

- Layer API overhaul - no more “Options/Config” structures; no more “drivers”
- Namespace overhaul - rolled Util/Features/Symbology/Annotation into the core
- ImGui integration - user intefaces - eventual replacement for “Controls”
- CompositeElevation/Image/LandCover Layers
- GrassLayer (splatting subsystem)
- Powerline Layer
- Wind Layer - add winds that will affect the GrassLayer
- Decal Layers - add geospatial decals to the terrain
- TiledFeatureModelLayer - fast feature rendering for pre-tiles data
- Arbitrary region invalidation and refresh for terrain engine
- Geocoder (OGR - optional build)
- LandCoverLayer - new fractal refinement
- 3D-Tiles Layer
- glTF support (partial, for 3D-Tiles)

- Cesium Ion Layer
- GDALDEM Layer - hillside shading, etc.
- NetworkMonitor tool
- WEBP loader - fast compressed imagery
- BASIS support - image compression
- DRACO support (GLTF) - geometry compression
- Support for >2GB ZIP files (new OSG ZIP plugin)
- Tracy integration - profiling
- Better error reporting infrastructure
- Performance improvements & bug fixes galore
- New documentation structure

7.4 Version 2.10 (November 2018)

- REX terrain engine promoted to default. Old MP engine is now in legacy support mode.
- Removed the osgEarthQt nodekit from the SDK, along with all Qt examples
- Cleanup of the internal serialization architecture (i.e. osgEarth::Config)
- Compatibility with OSG 3.6.x release/branch
- GL3 and GLCORE profile support
- VirtualProgram performance improvements
- New LineDrawable and PointDrawable classes for cross-GL-profile support
- Better progress/cancelation handling throughout the SDK, including feature subsystem
- Prototype support for ECI reference frames
- Support for "new" osgText implementation in VirtualProgram framework
- New ClusterNode utility class for clustering proximite objects
- Removed deprecations: MaskNode, Profiler, StateSetLOD, TileKeyDataStore, WrapperLayer, MarkerResource, MarkerSymbol, StencilVolumeNode, TritonNode, AnnotationEvents, PolyhedralLineOfSight, some CullingUtils objects

7.5 Version 2.9 (February 2018)

- New "REX" terrain engine that supports random access tile loading, terrain morphing, faster add/remove
- New Map/Layer architecture to begin standardizing "everything is a layer" approach
- Per-layer shaders, configuration from earth file (rex only)
- Experimental screen-space GPU lines
- Better support for GLCORE, GL 3.3+, and VAOs
- Transition several Extension/etc. to Layers (AnnotationLayer, MGRSGraticule, FeatureModelLayer, SimpleOceanLayer)

- Reworked the mask generate for REX to support skirts
- Synchronous pre-loading of first-LOD terrain data
- GeoTransform node, Annotations self-discover terrain (don't need to pass in MapNode anymore)
- Experimental FlatteningLayer to flatten the terrain based on feature data
- Combine multiple shaders in a single file/string with [break]
- New ViewFitter class fits to view to a set of points
- Refactored splatting into SplatLayer, GroundCoverLayer
- New improved ephemeris calculator for sun position
- New PagedNode class for easier paging
- Support new OSG 3.5.8 text implementation
- Support GEOS 3.6+
- Added core LandCover/LandCoverLayer classes for classification data
- Added Future/Promise construct for asynchronous operations
- Re-written MGRS, UTM and GARS graticules
- Lots of bug fixes

7.6 Version 2.8 (September 2016)

- Disabled feature tessellation tiling in BuildGeometryFilter unless max_polygon_tiling_angle is explicitly set. Cropping code was causing issues especially around the poles. Need to come up with a more general solution in the future.
- Better support for osg::Fog in VirtualPrograms with FogEffect. Implemented multiple fog modes.
- Always applying min_range and max_range in MPGeometry to prevent uniform leakage.
- Proper support for centroid clamping for MultiPolygons.
- New requirement to call open() on TileSources and Layers when creating at runtime. This lets you explicitly get the Status of a layer and report errors to users.
- Fixes to EGM96 vertical datum grid.
- BUILD_OSGEARTH_EXAMPLES cmake option for disabling building examples.
- Added nearest sampling support for heightfields
- New feature_join for adding attributes from intersecting
- osgearth_deformation demo
- Scatter filter support for pointsets. Simply places models at each point in the PointSet.
- Performance optimizations when discarding features in javascript style selectors when returning null styles
- Feature geometry caching support
- New min_expiry_frames and min_expiry_time options to TerrainOptions.
- Proper createTile implementation for Rex engine.
- RocksDB cache plugin.

- New osgearth_server application (based on Poco networking libraries). Serve up osgEarth tiles rendered on the GPU to your favorite web mapping tools like Leaflet, OpenLayers and Cesium!
- Packager now supports writing to MBTiles
- New osgearth_skyview example for drawing an "inside out" earth. Turns out osgearth is a great photosphere viewer!
- Experimental WinInet support to replace CURL. New osgearth_http test app.
- Upgraded duktape to version 1.4.0
- Memory usage testing support (osgearth_viewer –monitor to enable)
- New osgearth_3pv utility application.
- Better support for pretiled datasets like TFS and Mapnik Vector Tiles in FeatureRasterSource (and agglite driver)
- Better support for node tethering in EarthManipulator
- Doxygen support
- New openstreetmap vector tiles demos (openstreetmap_buildings.earth and openstreetmap_full.earth)
- Support for Mapnik Vector Tiles datasets
- Fixed improper inversion of y tilekey in FeatureModelGraph and updated all drivers.
- CURLOPT_ENCODING support. If you've built curl against zlib, proper HTTP headers for gzip and deflate will be added and automatically decompressed.
- New osgearth_splat example
- New osgEarthSplat NodeKit
- New "template" plugin based on NLTemplate that allows you to write templatized earth files
- Support for xi:include in earth files
- Minimum OpenSceneGraph version is 3.4.0
- Removed MINIZIP dependency
- New Triton and Silverlining NodeKits
- New feature_elevation driver that produces features from
- New raster to feature driver for turning rasters to features
- 330 compatibiity default shader version for GLSL
- Normal mapping integrated into MP, removed normal map extension.
- TravisCI and Coverity support

7.7 Version 2.7 (July 2015)

- New ObjectIndex system for picking and selection
- New RTT-based picker that works for all geometry including GPU-modified geometry
- Extensions - modular code for extending the capabilities of osgEarth
- New procedural texture splatting extension
- Upgraded ShaderLoader for better modularization of VirtualProgram code

- New “elevation smoothing” property to MP terrain engine
- New support for default MapNodeOptions
- Logarithmic depth buffer lets you extend your near and far planes
- Better Triton and Silverlining support
- Overhaul of the elevation compositing engine and ElevationQuery utility
- New Raster Feature driver lets you generate features from raster data
- Attenuation and min/max range for image layers
- New shader-based geodetic graticule
- New day/night color filter
- Viewpoint: consolidation of look-ats and tethering
- New CoverageSymbol for rastering features into coverage data; agglite driver support
- New feature clustering and instancing algorithms for better performance and scalability
- Noise extension for creating a simplex noise sampler
- New TerrainShader extension lets you inject arbitrary shader code from an earth file
- VirtualProgram: specify all VP injection criteria with GLSL #pragmas
- Normal mapping extension with automatic edge-normalization
- Bump map extension for simple detail bumping
- Performance improvements based on GlowCode profiling results

7.8 Version 2.6 (October 2014)

Maintenance Release. Release notes TBD.

7.9 Version 2.5 (November 2013)

Terrain Engine

The terrain engine (”MP”) has undergone many performance updates. We focused on geometry optimization and GL state optimization, bypassing some the OSG mechnisms and going straight to GL to make things as fast as possible.

MP has a new optional “incremental update” feature. By default, when you change the map model (add/remove layers etc.) osgEarth will rebuild the terrain in its entirely. With incremental update enabled, it will only rebuild tiles that are visible. Tiles not currently visible (like those at lower LODs) don’t update until they actually become visible.

Caching

Caching got a couple improvements. The cache seeder (osgearth_cache) is now multi-threaded (as it the TMS packager utility). The filesystem cache also supports expiration policies for cached items, including map tiles.

JavaScript

We updated osgEarth to work with the newest Google V8 JavaScript interpreter API. We also now support JavaScript-Core as a JS interpreter for OSX/iOS devices (where V8 is not available).

Terrain Effects

A new TerrainEffect API makes it easy to add custom shaders to the terrain. osgEarth has several of these built in, including NormalMap, DetailTexture, LODBlending, and ContourMap.

New Drivers

There is a new Bing Maps driver. Bing requires an API key, which you can get at the Bing site.

We also added a new LibNOISE driver. It generates parametric noise that you can use as terrain elevation data, or to add fractal detail to existing terrain, or to generate noise patterns for detail texturing.

Other Goodies

- Shared Layers allow access multiple samplers from a custom shader
- A new "AUTO_SCALE" render bin scales geometry to the screen without using an AutoTransform node.
- PlaceNodes and LabelNodes now support localized occlusion culling.
- The Controls utility library works on iOS/GLES now.

7.10 Version 2.4 (April 2013)

- New "MP" terrain engine with better performance and support for unlimited image layers (now the default)
- Shader Composition - reworked the framework for more flexible control of vertex shaders
- EarthManipulator - support for mobile (multitouch) actions
- GPU clamping of feature geometry (ClampableNode)
- TMSBackFiller tool to generate low-res LODs from high-res data
- OceanSurface support for masking layer
- New RenderSymbol for draw control
- Fade-in control for feature layers
- OverlayDecorator - improvements in draping; eliminated jittering
- Added feature caching in FeatureSourceIndexNode
- ShaderGenerator - added support for more texture types
- Draping - moved draping/clamping control into Symbology (AltitudeSymbol)
- Lines - add units to "stroke-width", for values like "25m", also "stroke-min-pixels"
- PolygonizeLines operator with GPU auto-scaling
- New Documentation site (stored in the repo) at http://osgearth.readthedocs.org
- Decluttering - new "max_objects" property to limit number of drawables
- New ElevationLOD node
- SkyNode - added automatic ambient light calculation
- New DataScanner - build ImageLayers from a recursive file search
- Qt: new ViewWidget for use with a CompositeViewer
- Map: batch updates using the beginUpdate/endUpdate construct
- GLSL Color Filter: embed custom GLSL code directly in the earth file (glsl_filter.earth)
- Agglite: Support for "stroke-width" with units and min-pixels for rasterization

- Terrain options: force an elevation grid size with <elevation_tile_size>
- Better iOS support
- New "BYO" terrain engine lets you load an external model as your terrain
- New "first_lod" property lets you force a minimum LOD to start at
- Better support for tiled data layers
- Lots of bug fixes and performance improvements
- New documentation site stored in the osgEarth repo (docs.osgearth.org)

CHAPTER 8

Upgrading from osgEarth 2.x to osgEarth 3.x

The goal for osgEarth 3.x was to make the SDK easier to work with for both developers (writing against the API) and end users (creating earth files). This document serves to help you update existing code and earth files for use with osgEarth 3.x.

The primary objectives of 3.x were:

- Treat everything as a Layer, so that a Map can be a container for almost all data.
- Make all the various Layer types explicit and reduce the dependency on using plug-ins.
- Simplify the API for creating and adding Layers by eschewing the "Options" structure pattern.
- Simplify the approach to making new or custom Layer types.

8.1 Earth File Changes

Following the idea of downplaying plugin layer types, Layers in an earth file are now explicit. For a GDAL layer in 2.x you would have had something like this:

```
<image name="My Layer" driver="gdal">
    <url>file.tif</url>
</image>
```

In 3.x the layer type is explicit, like so:

```
<GDALImage name="My Layer">
    <url>file.tif</url>
</GDALImage>
```

Image and Elevation layer types are still separate, and use explicit types are well. A 3.x GDAL elevation layer looks like:

```
<GDALElevation name="My DEM">
    <url>dem.tif</url>
</GDALElevation>
```

Here is a partial table mapping old setups to the new 3.x format:

8.1.1 Composite layers

Composite layers in 2.x used the "composite" driver in an image or elevation layer. In 3.x there is a new first-class `CompositeImage` and `CompositeElevation` types. Here is an example:

```
<Map>
    <CompositeImage name="combined">
        <layers>
            <TMSImage>
                <url>http://readymap.org/readymap/tiles/1.0.0/7/</url>
            </TMSImage>
            <GDALImage>
                <url>locallInset.tif</url>
            </GDALImage>
        </layers>
    </CompositeImage>
</Map>
```

8.1.2 Feature sources

Feature layers (layers containing feature data) are not new to 3.x, but they are also explicit just like the visible layer types above. For example, in osgEarth 2.x you might define a simple feature layer like so:

```
<feature_model name="states">
    <features name="states" driver="ogr">
        <url>../data/usa.shp</url>
    </features>
    <styles>
        <style type="text/css">
            states {
               stroke: #ffff00;
            }
        </style>
    </styles>
</feature_model>
```

In osgEarth 3.x, the definition (using an embedded feature source) looks like this:

```
<FeatureModel name="US States">
    <OGRFeatures name="US-Data">
        <url>../data/usa.shp</url>
    </OGRFeatures>
    <styles>
        <style type="text/css">
            states {
               stroke:#ffff00;
            }
        </style>
```

(continues on next page)

(continued from previous page)

```
    </styles>
</FeatureModel>
```

Or, you can define your feature data as a separate Layer and reference it. This way, multiple visible Layers can use the same feature source:

```
<OGRFeatures name="data:states">
    <url>../data/usa.shp</url>
</OGRFeatures>

<FeatureModel name="US States" features="data:states">
    <styles>
        <style type="text/css">
            states {
               stroke: #ffff00;
            }
        </style>
    </styles>
</FeatureModel>
```

Even a stylesheet can be a separate layer now. You can rewrite the same thing above as:

```
<OGRFeatures name="data:states">
    <url>../data/usa.shp</url>
</OGRFeatures>

<Styles name="data:styles">
    <style type="text/css">
        states {
           stroke: #ffff00;
        }
    </style>
</Styles>

<FeatureModel name="US States" features="data:states" styles="data:styles">
</FeatureModel>
```

8.2 API Changes

In 2.x, you created map layers (and various other things) by creating an "Options" structure and then passing that to the constructor of the new Layer. In 3.x you no longer need to do that; you can just create a new Layer and call its setter functions.

Here is an example of creating a WMS image layer in osgEarth 2.x:

```
#include <osgEarthDrivers/wms/WMSOptions>
...
WMSOptions wms;
wms.url() = "http://mesonet.agron.iastate.edu/cgi-bin/wms/nexrad/n0r.cgi";
wms.format() = "png";
wms.layers() = "nexrad-n0r";
wms.srs() = "EPSG:4326";
wms.transparent() = true;
```

(continues on next page)

(continued from previous page)

```
ImageLayerOptions wmsLayerOptions("WMS NEXRAD", wms);
wmsLayerOptions.cachePolicy() = CachePolicy::NO_CACHE;

ImageLayer* layer = new ImageLayer(wmsLayerOptions);
map->addLayer(layer);
```

In 3.x the API is more intuitive, without any intermediate structures:

```
#include <osgEarth/WMS>
...
WMSImageLayer* wms = new WMSImageLayer();
wms->setURL("http://mesonet.agron.iastate.edu/cgi-bin/wms/nexrad/n0r.cgi");
wms->setFormat("png");
wms->setLayers("nexrad-n0r");
wms->setSRS("EPSG:4326");
wms->setTransparent(true);
wms->options().cachePolicy() = CachePolicy::NO_CACHE;
map->addLayer(wms);
```

Follow the same pattern for all layer types. See the `osgearth_map.cpp` example for several examples.

CHAPTER 9

Getting Support

Since osgEarth is a free open source SDK, the code is available to anyone and we welcome and encourage community participation when it comes to testing, adding features, and fixing bugs.

9.1 GitHub Repository

Use the osgEarth GitHub repository to access the source, download releases, and post bug reports.

9.2 Discussion Forum

Join the osgEarth Discussion Forum to interact with other users. Please read and follow these guidelines for using the forum. FOLLOWING THESE GUIDELINES will make it MUCH MORE LIKELY that someone will respond and try to help:

- Sign up for an account and use your real name. You can participate anonymously, but using your real name helps build a stronger community. Sign your posts too!
- Limit yourself to one topic per post. Asking multiple questions in one post makes it too hard to keep track of responses.
- Always include as much supporting information as possible. Post an earth file or short code snippet. Post the output to osgearth_version –caps. Post the output to gdalinfo if you are having trouble with a GeoTIFF or other data file. List everything you have tried so far.
- Be patient!

9.3 OpenSceneGraph Support

osgEarth operates on top of OpenSceneGraph (OSG), an open source 3D rendering toolkit. If you are unable to find answers to your problems on GitHub or in the osgEarth Forum, the OSG Google Group is another place to look.

9.4 Professional Services

The osgEarth team supports its efforts through professional services. At Pelican Mapping we do custom software development and integration work involving osgEarth (and geospatial technologies in general). We are based in the US but we work with clients all over the world. Contact us if you need help!

CHAPTER 10

License

osgEarth is Copyright © Pelican Mapping and licensed under the LGPL free open source license.

This means:

- You can link to the osgEarth SDK in any commercial or non-commercial application free of charge. If you make any changes to osgEarth itself, you must make those changes available as free open source software under the LGPL license. (Typically this means contributing your changes back to the project, but it is sufficient to host them in a public GitHub clone.)
- If you redistribute the osgEarth source code in any form, you must include the associated copyright notices and license information unaltered and intact.

iOS / static linking exception: The LGPL requires that anything statically linked to an LGPL library (like osgEarth) also be released under the LGPL. We grant an exception to the LGPL in this case. If you statically link osgEarth with your proprietary code, you are NOT required to release your own code under the LGPL.

osgEarth is maintained by your friends at Pelican Mapping.