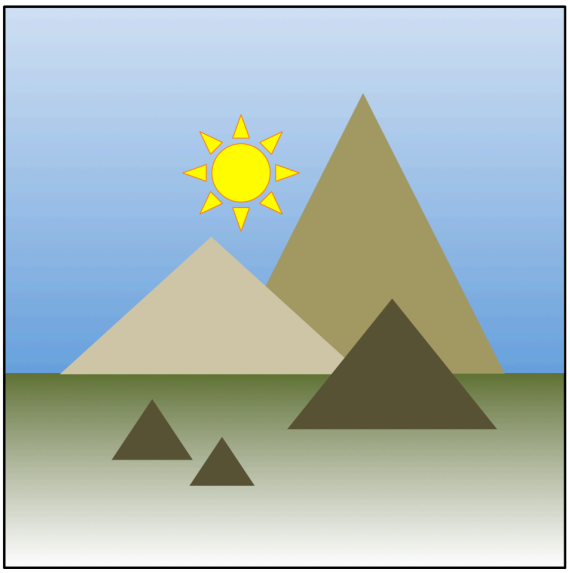
Desktop Exploration of Remote Terrain



Software Design

Introduction

Remote explorers including satellites, the space shuttle, planes, and robots collect data for digital terrain models (DTM) via stereoscopic imagery, LIDAR, and other remote sensing methods. Models of locations on the Earth, Mars, the Moon, and other planetary bodies they generally consist of a digital elevation model (DEM), or height map, accompanied by one or more spatially referenced orthoimages, or layers. NASA missions have produced a significant number of these data sets and their 3D nature provides a compelling aspect for visual analysis and collaboration. However, very large and complex, they are underutilized due to the lack of dedicated software tools and documentation. *Desktop Exploration of Remote Terrain (DERT)* is a virtual environment for exploring NASA DTMs.

A number of existing software applications provide capabilities for viewing DTMs. However, they are not widely used for these data sets. They include but are not limited to:

- Google Earth by Google
- WorldWind by NASA
- ARCGIS by ESRI
- ENVI by Exelis
- Matlab by Mathworks
- Blender by Blender Foundation
- ISIS 3 by USGS

These products were originally developed for other purposes including visualizing virtual globes, mapping, processing images, and editing mathematical models. The commercial products are advanced applications and can have high per seat costs, while others require technical expertise to install. Most have scalability limitations, losing data resolution. Fundamentally, all are complex and require a significant investment to learn and use. DERT does not attempt to replace these full featured products but instead provides a simple "viewer" for NASA's DTMs with the combination of purpose built, open source software and freely available support applications.

Prior Work

The Intelligent Systems division at NASA Ames Research Center has a history of developing virtual environments for planetary missions. Viz was created as an aid for science planning activities for Mars Polar Lander. Mercator was developed for the Phoenix Mars Lander, and Antares for Mars Science Laboratory. These projects provided two significant components for DERT development: a body of existing algorithms and visualization techniques, and a set of use cases. Many of the algorithms and techniques are incorporated into the DERT baseline functionality,

reducing software development time and effort. The use cases provide a basis for requirements development.

Requirements

As stated above, DERT is a viewer for NASA DTMs. The following are baseline requirements.

1. Visualize a terrain model in 3D.
2. Provide interactive performance on Mac and Linux laptop platform.
3. Accommodate commonly used file formats and handle file sizes > 2GB.
4. Display multiple layers (orthoimages) at varying opacities.
5. Interoperate with other software using standards.
6. Preserve geo-reference information from DEMs and orthoimages.
7. Scale in terms of visualization and DTM size.
8. Navigate through the terrain in 3D.
9. Handle both projected and unprojected DTMs.
10. Display the DEM alone without any orthoimages.

The following requirements for DERT are derived from feedback from scientists during NASA planetary missions.

11. Provide a contour line layer.
12. Save and restore viewpoints.
13. Simulate lighting.
    a. Solar positioning using ephemerides.
    b. Artificial light positioning.
    c. Hill shading.
14. Simulate shadows.
15. Provide interactive terrain exaggeration.
16. Save and restore session state.
17. Provide multiple views of same DTM.
18. Provide movable landmarks.
19. Provide tools that:
    a. Measure distance, strike, and dip.
    b. Measure distance along a path.
    c. Measure area, surface area, and volume.
    d. Determine where a plane intersects the surface topography.
    e. Profile terrain.
    f. Display a grid, both Cartesian and radial.
20. Display for an arbitrary point:
    a. The location in projected and unprojected coordinates.
    b. The surface normal vector.
    c. The direction to the light.
21. Provide derivative layers:
    a. Color mapped elevation.

    b. Color mapped slope.
    c. Color mapped aspect.
    d. Grid lines draped on surface.
22. Display a user supplied image in the scene.
    a. Display a billboard of the image at a designated point.
    b. Open the image in the platform viewer.
23. Simulate through-the-lens view for a camera.
    a. Display in a separate window.
    b. Simulate viewshed as a layer.
24. Display lines from an ESRI shape file.
25. Provide a stereo display (anaglyph or hardware).
26. Render view of virtual world to a high-resolution screenshot.
27. Handle unstructured landscape meshes (for caves and overhangs).
28. Eliminate need to read documentation in order to use the software.

Additional requirements became apparent during development.

29. Provide a color map editor.
30. Fly through a list of viewpoints.
31. Any data preparation software should have both a graphical user interface (GUI) and be scriptable.
32. Provide a heads up display (HUD).
33. Load landscapes from a tile server.
34. Provide hyperwall support.

Design

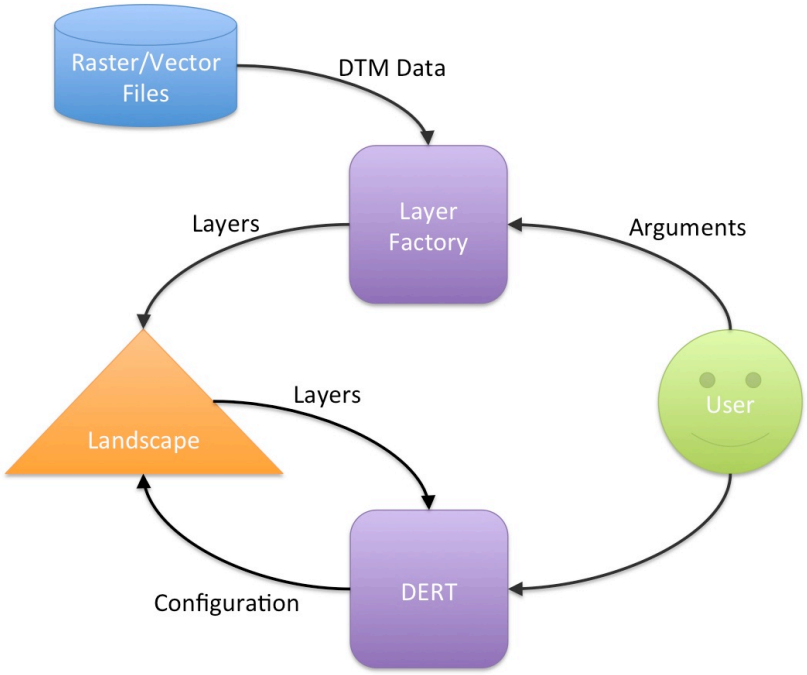
DERT presents two major design challenges. The first challenge is to transform the terrain model into something a virtual environment can ingest. This requires significant data processing, but only needs to be done once for a given DTM. One of the most common forms for terrain model data is a multi-resolution tiled image pyramid. This structure was chosen for DERT for its simplicity and interoperability with other software.

One type of multi-resolution tiled image pyramid maintains a quad tree of files in a directory structure or database. Each quad, or tile, in the tree represents a rectangle of the DTM at a specific resolution. A quad will have 4 children if there is higher resolution data for the region it covers. Each tile image contains the same number of pixels but a child only covers 1 quarter of the parent's area so a child has twice the resolution of its parent. As the viewpoint changes, higher resolution tiles are loaded for terrain close to the viewpoint and terrain far from the viewpoint is replaced with lower resolution tiles. Depending on available memory, tiles are cached. This approach maintains a reasonable level of performance for the application.

One image pyramid represents a *layer* of the terrain model. For DERT, a layer is processed from one raster data set. For example, a layer could be an orthoimage, or an elevation height map. It could also be an elevation derivative such as a slope map. All layers are geo-referenced to the elevation layer. The collection of the elevation layer pyramid and the referenced layer pyramids is called a *landscape*.

The second challenge is to simulate the terrain model in a virtual environment. This environment should be usable on a laptop and attempt to stay true to the terrain model as much as possible in terms of scale and color. Image pyramids lend themselves well to scene graphs and DERT makes use of this data structure. Additionally, a scene graph is useful for implementing tools, landmarks, and other requested objects. The virtual world containing the scene graph of the landscape, tools, and other objects, is called the *world*.

The software consists of two components: DERT and a support application called LayerFactory. DERT provides the virtual environment for the world scene graph and LayerFactory builds layers for landscapes.



Data flow among DERT components

A virtual environment, by nature, is a complicated application. Every possible attempt was made to simplify DERT. This is visible in the choice of language, windowing system, and de-scoped requirements. The following design elements address the requirements listed above. They were influenced by feedback from prototype testing, the behavior and availability of third party libraries, and issues with both OpenGL and Java on the Apple and Linux platforms.

General

Architecture: A 64-bit architecture is fairly ubiquitous and is necessary for the significant memory usage of this software.

Language: Java is cross platform and offers a number of features (garbage collection, for example) that can be leveraged to shorten development time. Version 6 was chosen for implementation but the software can be built and executed in Java 7 & 8. User laptop configuration issues required a baseline implementation in version 6.

Mouse: As with many 3D visualization applications, this one is best used with a 3-button mouse. However it is also possible to use it with a Mac laptop track pad.

Windowing System: In experiences developing previous applications, rich client platform libraries such as Eclipse RCP exhibited a number of difficult and unintuitive issues, presented a steep learning curve to the developer, and did not contribute significantly to the user experience. As such, DERT was designed with standard Java AWT/Swing to provide a simpler baseline for open source.

Landscape and Layers

Data Storage: A layer could be stored in a directory of files, a single tiled TIFF file, or a database. The simplest structure is a directory containing the pyramid as files and a Java properties file called *layer.properties* for metadata.

Metadata Access: Layer.properties contains geo-reference information and other metadata for the layer. This file is human readable and easily edited. Putting the metadata in a single editable file reduces disk usage and makes customization easier.

Pyramid File Structure: A quad-tree directory structure where each "quad" consists of a directory containing a single file labeled "0.png" that contains the tile data and optionally 4 child quad subdirectories. The children are labeled "1", "2", "3", and "4" respectively. Each child covers a quadrant of the parent where 1 is in the upper left, 2 is in the upper right, 3 is in the bottom left, and 4 is in the bottom right. Each child tile contains the same number of pixels as the parent but covers a physical area that is half as wide and half as long resulting in an increase in data resolution. The hierarchy continues until the maximum resolution is achieved. This structure provides a convenient method for tile access and caching. The file path to the tile provides a unique key (e.g. "elevation/1/2/3/4/0.png").

Tile File Format: The file format of the tile is Portable Network Graphics (PNG). This is a commonly used, lightweight, and compressible image format. Unfortunately, there is no standard floating-point version of PNG and it is required for the elevation layer. As a work-around, LayerFactory writes DEM tiles as RGBA images and DERT converts them back to floating-point with no data loss. This comes in handy in that elevation details in the DEM tiles are visible when viewed with the platform image viewer. TIFF could have been used here but has too much overhead for single tiles.

Tile Dimensions: The dimensions of a tile are 2^n+1 x 2^m+1. The additional row or column is the overlap for connecting the neighboring tile. Limiting the tiles to squares would simplify the implementation. However, allowing rectangular tiles reduces additional disk space due to padding.

Pyramid Scalability: A sub-pyramid may be appended to an existing quad tree without filling in the rest of the original pyramid.

Software Extensibility: This design could easily be extended to use a database or tile server.

Layer Generation

User Interface: If the LayerFactory invocation does not have enough command line arguments then a simple GUI will appear. LayerFactory runs headless and can be included in a script for processing multiple layers at once.

Pyramid Levels: LayerFactory copies the original raster to a temporary file padding it to a power-of-2 width and length. The original raster provides the highest resolution level, which resides at the base of the pyramid. Each level moving up the pyramid is an average of the original raster using a mean kernel of increasing size.

Supported Raster Formats: LayerFactory supports GeoTIFF and NASA's PDS .IMG formats. To leverage existing software and standards, libtiff, wrapped with a Java Native Interface (JNI), handles reading and writing TIFF (including BigTIFF) files. The PDS loader is pure Java.

Very Large Files (> 2GB): LayerFactory uses the Java MappedByteBuffer class (memory mapped I/O) to provide memory to process very large raster files.

Prototype Data: The following data sources were used to prototype LayerFactory and DERT:

- MRO HiRISE and CTX
- Results from Ames Stereo Pipeline
- USGS EarthExplorer SRTM and Landsat 8
- Mars Express HRSC
- NASA PDS LOLA and LROC
- DTMs generated from MSL rover terrain data

Data Preparation: The Geospatial Data Abstraction Library (GDAL), a widely used free software library, is used to prepare raster files for LayerFactory, resizing and re-projecting a raster, for example. GDAL can be downloaded from www.gdal.org.

Contour Map Layer: GDAL generates contour vectors from a DEM, producing a vector file. LayerFactory renders the vectors as image tiles. It uses an OpenGL pBuffer as an off-screen renderer to implement this feature so it can remain headless.

Virtual Environment

Graphics Engine: Ardor3D provides a Java scene graph library and was chosen over others because it is free with no license requirements. It is oriented to gaming and therefore performance driven, but also offers the features necessary for scientific visualization. Ardor3D is a layer over OpenGL and over the course of DERT development, OpenGL changed significantly. As a result, Ardor3D has become somewhat obsolete, justifying a future rewrite of the DERT visualization component.

Virtual World: The virtual world consists of a single landscape plus added map elements. DERT is limited to exploring only one world at a time, as more than one greatly complicates its architecture.

Dimensional Range: The dimensional range of landscape data can be very large. Consider rover camera terrain models on the order of centimeters verses satellite models that are many kilometers in dimension. Surface normal calculations involving these numbers result in an even wider range, requiring double precision. The graphics card is only 32 bits so numbers that fall outside the floating point range will cause rendering artifacts. DERT normalizes landscape coordinates for each tile by subtracting the bounds to reduce very large numbers or multiplying by a constant to enlarge very small numbers.

OpenGL Clipping Planes: The OpenGL camera maintains near and far clipping planes, bounding the depth buffer. Rendering artifacts occur when these planes are not positioned optimally for the dimensions of the virtual world. DERT computes the location of these planes based on the current position of the viewpoint relative to the scene.

User Interface: The DERT main GUI consists of a single large window that displays a view of the virtual world (the *worldview*) and a toolbar at the top. Each control on the toolbar opens a non-modal view that covers a specific feature. Shortcuts to some functions are provided through a context menu invoked by clicking in the worldview.

Session Persistence: DERT saves a collection of views, viewpoints, objects, and other state into an object called a configuration. This is persisted to an XML file using the xStream library. This is a simplified version of a feature found in more complicated windowing libraries such as Eclipse RCP.

Messages and Logs: DERT provides a console view for messages and writes all messages as well as exceptions and other errors in a log file. The log file can then be included in a trouble report.

Help: The help feature is implemented as web pages displayed in a simple browser. This approach provides a familiar base for viewing and creating help pages. The help content is formatted in a familiar FAQ style.

Through-the-lens Camera View: DERT provides a camera object placed in the scene graph that can be grabbed and dragged with the cursor. A separate through-the-lens view changes as the camera moves and/or pans and tilts.

Multiple Views of Same Landscape: For a fully featured form of multiple views, two copies of DERT can be run concurrently.

User Interaction

Moving Objects: An object such as a tool or landmark is positioned by dragging it along the terrain. This approach is somewhat limited compared to full 3D movement with 6 degrees of freedom but provides grounding to the user and doesn't require any special hardware. DERT performs a pick test for each object movement over the terrain surface. Limiting objects to a small number of polygons maintains a high level of performance. Additionally, a bounds-check reduces the amount of objects required to undergo the more expensive pick test.

Viewpoint Manipulation: DERT uses a constrained model-centered rotation technique to minimize user disorientation. It allows full rotation of the world around the vertical (Z) axis while limiting vertical rotation to 180 degrees around the horizontal axis. Dragging the terrain with the left mouse button translates the viewpoint along the plane of the terrain. This movement supports kinetic scrolling. Dragging with the middle mouse button pressed translates the viewpoint parallel to the plane of the screen. Scrolling the middle mouse button dollies or zooms. At mouse release, the translation algorithm performs a ray pick to find the center of rotation. This point is used to adjust the speed of the dolly. This approach grounds the user somewhat and doesn't require special hardware. If a user does get lost, pressing the reset button on the toolbar returns the viewpoint to the default overhead position.

Animation: Commonly, 3D worlds are exhibited in an animation, typically flying through the terrain. DERT provides a fly through feature for the viewpoint list or a path.

Feedback: In the worldview, the cursor changes to a hand when an object is grabbed and about to be dragged. DERT maintains a 3D cursor at the center of rotation. The toolbar contains a compass that displays the direction the user is viewing.

Additional feedback includes icon changes, cursor changes, and invalid input beeps. Some dialogs display messages at the top.

Simulation

Math Libraries: DERT uses JPL's JNISpice library for sun position calculations and the Ardor3D math library for vector and matrix calculations. A JNI-wrapped version of the widely used Proj.4 library is used for projection conversion.

Execution: Ardor3D provides continuous updating and rendering at a constant frame rate, an approach used in most simulations. DERT is primarily user driven, and only renders a frame if something has changed in the scene. DERT uses less CPU with this modification, making it more likely that other applications may run concurrently.

Threads: Rendering is carried out on the main Java event thread. Other tasks, such as sampling, run on separate threads to minimize the possibility of blocking the user interface.

Strike and Dip Measurements: Distance, strike and dip is measured with a tool that simulates a tape measure. The tool provides continuous output of endpoint location for easier positioning.

Distance, Area, and Volume Measurements: DERT provides a Path tool that simulates a set of points on the landscape. The tool computes a set of statistics including path distance, polygon area, sampled surface area, sampled mean elevation, sampled mean slope, and sampled volume. It samples from the highest resolution landscape tiles and uses the polygon formed by the points as the lower (or upper) bound of the volume. Performing a ray pick into the polygon determines the bound of a single sample.

Plane/Topography Intersection: A Plane tool, defined by three movable points, computes strike and dip on the fly. It also provides an elevation difference map and scaling options.

Terrain Profile: DERT provides an interactive terrain Profile tool. It displays the elevation at points along a transect line in a graph that is updated as an endpoint is moved. To maintain responsiveness, the tool samples elevation from the currently displayed tiles. When saved to a file, values are sampled from the highest resolution data. For presentations, it is expected that the user will save the data and present them in a fully featured graphing application. This eliminates the need to reproduce complex graphing features in DERT.

Landmarks and Billboards: DERT provides two types of landmarks, one is a simple icon and the other is a solid object that has depth and causes shadows. Icons of billboards images are placed in the landscape with the full images viewed with the platform image viewer.

Grids: DERT provides adjustable, movable, radial and Cartesian grids.

Arbitrary Point Information: A special landmark in the form of a green marble indicates the location of the last picked point. A pick displays the marble coordinates, both projected and unprojected (lat/lon), as well as the direction to the light and the surface normal at the point.

Visualization

Shader Programs: DERT uses shader programs for several techniques. Each shader is a GLSL program generated on the fly. Shader code generation allowed the program to be customized and avoid platform GLSL version issues that occurred in a more parameterized program.

Shadows: Shadows are implemented via a shadow map technique with a shader. This allows the use of ambient shadows.

Multiple Layers at Varying Opacities: A shader controls the contribution of a layer to the overall visualization.

Interactive Terrain Exaggeration: The terrain is exaggerated by scaling the landscape mesh along the Z-axis. Additionally, the Z coordinate of objects on top of the landscape surface is adjusted so they float as the scale changes.

Display DEM Only: DERT computes surface normal vectors as landscape tiles are loaded. Along with lighting, these provide full hill shading capabilities.

Derivative Layers: DERT computes elevation, slope, and aspect maps using the Z coordinate or normal of each landscape mesh vertex and uses a color map to color the location. A shader provides a surface grid.

Stereo Display: DERT uses the Ardor3D stereo feature.

Simulate Viewshed as a Layer: A shader provides the viewshed layer. The implementation is basically the opposite of a shadow map.

Text: Text could be rendered using Ardor3D's BMText class or JOGL's TextRenderer. However, with all the orthoimages, billboards, and shadows, texture requirements

are at a premium. DERT employs a technique that uses GLUT fonts and no textures. This approach reduced complexity and risk and was adequate as the text requirements are modest.

Reduced and De-scoped Requirements

Several of the requirements were reduced or de-scoped due to time or funding constraints. These requirements include:

Multiple views of same DTM (17): This feature complicates the software architecture significantly, increasing risk of failure. The work-around is to run more than one instance of DERT concurrently.

Display lines from ESRI shape file (24): This requirement was reduced. A GeoJSON format file loader is provided by DERT and the ESRI shape file will be converted to GeoJSON by GDAL.

Render a high-resolution screenshot (26). This feature was eliminated due to high memory requirements. Since the video memory is used for rendering the contents of the virtual world, there is no guarantee that there is enough left for an off-screen renderer. Therefore, the screenshot must be rendered off-screen with a RAM based graphics. There is not enough funding or time available for the implementation of this feature.

Provide a heads up display (HUD) (32). This feature would have made DERT easier to use, however it would require a significant design effort. It was de-scoped for lack of time and funding.

Provide hyperwall support (34). This feature is specific to the NASA Lunar Science Institute hyperwall and would be useful for meetings. It was de-scoped for lack of time and funding

Provide a color map editor (29). Color maps may be edited with a text editor. This feature was de-scoped for lack of time and funding.

Handle unstructured landscape meshes (27). This feature requires a significant modification to the software architecture. It was de-scoped for lack of time and funding

Eliminate need to read documentation in order to use the software (28). Every effort was made to meet this requirement. However, DERT is a complicated application and it was not possible given the time available. There is a complete user guide.

Have DERT load landscapes from tile server (33). DERT handles PNG formatted tiles, a common tile format. A tile server loader was not implemented but could be added by extending a single Java class.

Any data preparation software should have both a GUI and be scriptable (31). Scripts are useful when creating a landscape with multiple layers. DERT relies heavily on GDAL for data preparation. GDAL is scriptable, but has no GUI. Writing a GUI for this software would be a major undertaking and is not in the scope of this project. LayerFactory has both a GUI and is scriptable.

Supporting Software

- Graphics Engine: Ardor3D by Ardor Labs (now JogAmp's Ardor3D Continuation)
- OpenGL Library: JOGL by JogAmp
- Ephemerides: JNISpice by NASA's Navigation and Ancillary Information Facility at JPL
- Object Serialization: XStream by Joe Walnes and the XStream Committers
- Cartographic Projections: Proj.4 by Gerald Evenden

Software Architecture for LayerFactory

Layer Factory

The *LayerFactory* class contains the *main* method and is executable. It collects the arguments from the user and passes them to instances of either *RasterPyramidLayerFactory* or *VectorPyramidLayerFactory*.

GUI

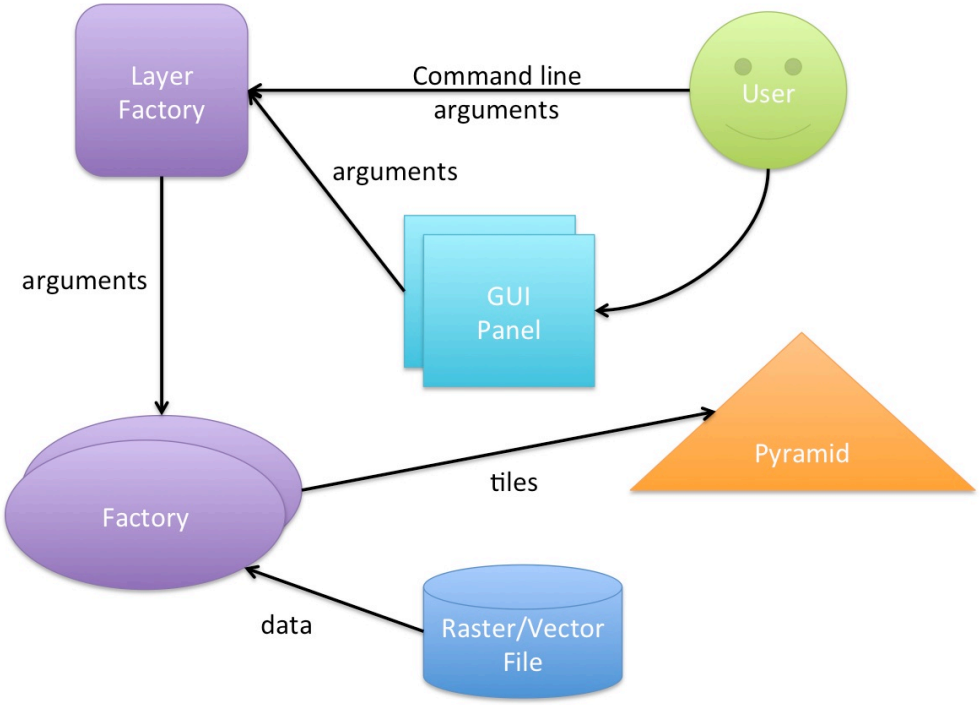
The GUI is only created if all of the necessary arguments are not present. *RasterLayerPanel* and *VectorLayerPanel* are the GUI classes. They collect raster pyramid and vector pyramid arguments respectively.

Raster Factory

The *RasterPyramidLayerFactory* class creates a tile pyramid from a raster file such as an image or DEM. It loads the file into a mapped byte buffer, and from it, writes a padded copy to a second buffer. Next it creates tiles by averaging pixels from the padded copy and writing out the tile directory structure. Finally, it writes the layer properties file.

Vector Factory

The *VectorPyramidLayerFactory* class creates an image tile pyramid from a vector (GeoJSON) file. It creates an off-screen renderer and renders each tile to the buffer at the required resolution by adjusting the OpenGL camera frustum. After each tile is rendered to memory, it is copied to the pyramid directory structure. Finally, the factory creates the layer properties file.



LayerFactory Data Flow

Software Architecture for DERT

DERT

At startup, the main class, *Dert,* creates the main application window with a toolbar, and the console. Once the user selects a configuration, the class *ConfigurationManager* loads the configuration and positions the console window, loads the landscape, creates the world, and puts the worldview in the main window. It then sets up the rest of the views. The landscape must be created before the world boundaries can be calculated and the worldview camera frustum determined.

Configuration

A *Configuration* object is a snapshot of the virtual environment as a list of *State* objects. It contains state objects for the worldview and other views, landscape, lighting, and map elements. A state object contains the attributes needed to re-instantiate a map element or view.

View

A *View* is a secondary, non-modal display that provides interaction with the virtual world. Views include the worldview, viewpoint, surface and layers, map element editing, lighting, console, help, graph, contour, and camera. See the DERT user guide for a description of each of these displays.

OpenGL Canvas

Some views use the Ardor3D graphics engine and OpenGL for rendering. These views employ a *SceneCanvas* object as a rendering surface and manage user mouse input with *InputManager* and *InputHandler* classes. They also use texture renderers and shaders for layer and shadow effects. The class *SceneFramework* provides a continuous rendering frame rate in all OpenGl views. An OpenGL context must be realized before any OpenGL requests (for example, creating a texture map) can be performed. This means that some objects such as shared textures must be initialized in a lazy fashion.

Map Elements

Map elements are landmarks, tools, and lines that the user adds to the virtual world. Map element classes are extended from the Ardor3D Node class and are persisted in

a configuration. Tool map elements may control the contents of a view. See the DERT user guide for a description of map elements.

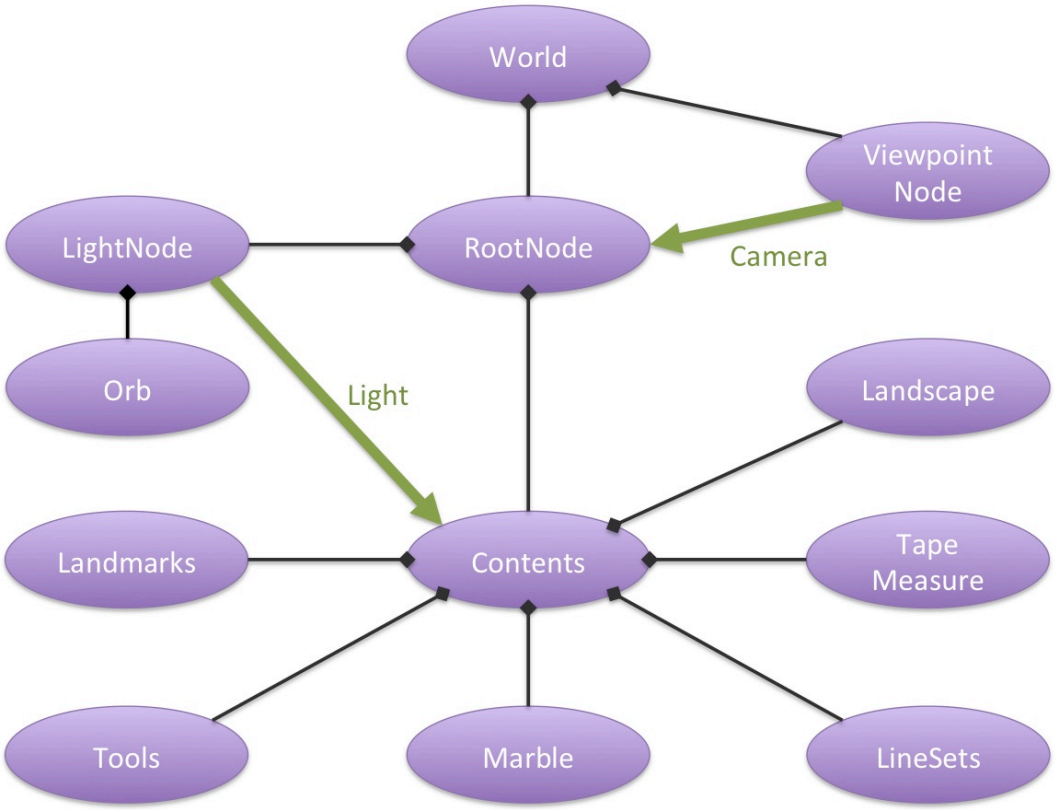
Viewpoint

The *ViewpointNode* is a scene graph node that manages the OpenGL camera for the worldview. It handles user navigation and is moved with the mouse via the input handler.

World

The *World* object is a singleton and houses the scene graph. It maintains the contents of the virtual world and contains several objects:

- *Landscape*: A scene graph node that represents the landscape on disk. It provides the 3D mesh from the elevation layer and the image overlays from other layers. The mesh is a quad tree structure that mimics the pyramid on disk. The quad tree consists of tiles that are loaded as needed and cached.
- *Lighting*: This object handles lighting, shadows, and light position. There are two OpenGL lights, one represented by an orb in the scene, and the other attached to the viewpoint.
- *Marble*: A scene graph node that represents the last picked point in the landscape. It maintains a view with information about its location.
- *TapeMeasure*: A scene graph node that provides measurements. It is implemented with a rubber band node and presents its information in a popup window. It is not persisted to a configuration.
- *Landmarks*: A scene graph node that is the parent of all landmark map elements.
- *Tools*: A scene graph node that is the parent of all tool map elements.
- *LineSets*: A scene graph node that is the parent of all lineset map elements.



DERT worldview scene graph: Lighting is performed on the *Contents* sub-graph and the subject of the OpenGL camera is the *RootNode* sub-graph.

Software Organization

DERT code is organized in Java packages. Following standard practices, all of the packages are prefixed with "gov.nasa.arc.".

dert: Top level package with Dert, the main class, and ViewManager.

dert.action: Contains base classes for menus and buttons.

dert.action.edit: Classes for the Edit menu in the toolbar.

dert.action.file: Classes for the File menu in the toolbar.

dert.action.mapelement: Contains classes for the worldview context menu and for the map elements view.

dert.ephemeris: Provides an interface to the SPICE library.

dert.icon: Contains all icons used in DERT as well as a class for handling them.

dert.io: Classes for input/output including those for the landscape tiles, GeoJSON, and comma separated value files.

dert.landscape: Classes for landscape including quad tree factory, cache, and layers.

dert.landscape.factory: Contains the LayerFactory classes.

dert.lighting: Classes that handle lighting and shadows.

dert.raster: Classes for raster files including a raster file base class, memory management, and spatial reference system.

dert.raster.geotiff: Provides GeoTIFF loader for LayerFactory.

dert.raster.pds: Provides PDS .IMG file loader for LayerFactory.

dert.raster.proj: Provides interface to Proj.4 library.

dert.render: Contains the classes that provide rendering capabilities for DERT. These include the rendering surface, scene management, texture rendering, color maps, special layer effects, off screen rendering, shadow maps, and viewshed.

dert.scene: Contains classes for non-landscape objects.

dert.scene.landmark: Contains landmark classes.

dert.scene.tapemeasure: Contains classes for the tape measure.

dert.scene.tool: Contains tool classes.

dert.scene.tool.fieldcamera: Contains classes for the camera object.

dert.scenegraph: Contains helper classes for creating scene objects. These classes are typically extensions of the Ardor3D Spatial or Node classes.

dert.state: Classes for configuration and state management.

dert.ui: Provides several custom GUI components.

dert.util: Utility classes.

dert.view: Provides classes for persistent windows (Views).

dert.view.contour: Classes for the contour view used by the Plane object for its elevation difference map.

dert.view.fieldcamera: Classes for the through-the-lens view for the Camera object.

dert.view.graph: Classes for the graph view used by the Profile object.

dert.view.lighting: Provides a view that handles the lighting properties and light position.

dert.view.mapelement: Provides a view for editing map elements.

dert.view.surfaceandlayers: Provides a view for configuring landscape layers.

dert.view.viewpoint: Provides a view for editing the viewpoint.

dert.view.world: Contains classes for the worldview.

dert.viewpoint: Contains classes for handling the viewpoint.