Introduction to Open Scene Graph

Ruth Aylett
What is Open Scene Graph?

- Designed for real-time scene rendering
  - Uses a scene graph to manage world database;
  - and multiprocessing to improve performance;
- Multi platform (at the moment IRIX, Linux, Windows, FreeBSD, Mac OS X, Solaris, HP-UX and even PlayStation 2)
- C++ API (Java and Python bindings available too);
- Built on industry standard OpenGL library (supports direct calls to OpenGL where necessary);
What is Open Scene Graph?

- Open Source with a large and active community
- Makes Use Of STL and Design Patterns
- Easy to develop plug-ins - lots of them available, esp. loaders
- Supports modern graphic cards features through support of OpenGL Shader Language
- All information and documentation on http://www.openscenegraph.org/
A few examples
A few examples
A few examples
What is in it? – The libraries (1)

- osg - Core scene graph
- osgUtil - Utility library for useful operations and traversers
- osgDB – Database reading and writing library
- osgFX – Special effects framework Nodekit
- osgText - NodeKit which add support for TrueType text rendering
What is in it? – The libraries (2)

- osgParticle - NodeKit which adds support for particle systems
- osgTerrain – Terrain generation Nodekit
- osgSim – Visual simulation Nodekit
- osgGA - GUI abstraction library
- osgProducer - viewer library integrating OSG with producer
What is OpenSceneGraph?

- Functional Components
  - OSG Scene Graph Rendering Elements
  - OSGUtil Traversers Enhancements
  - OSGDB Data Base Loading Plug-in Management
  - Plug-Ins

- OSGText
- OSGSim
- Node Kits…
Namespaces

- Every of the libraries has its own namespace (e.g. osg, osgDB, osgFX, etc.)
- Classes are either referenced including namespace (using scope operator, e.g. osg::Group)
- or without namespace, with additional “using namespace ***” line (e.g. using namespace osg;)
Core OSG library

- Helper classes - *memory management, maths classes*
- `osg::Nodes` - *the internal nodes in the scene graph*
- `osg::Drawables` - *the leaves of the scene graph which can be drawn*
- `osg::State*` - *the classes which encapsulate OpenGL state*
- Traversers/visitors - *classes for traversing and operations on the scene*
The structure of a scene graph

- osg::Group at the top containing the whole graph
- osg::Groups, LOD's, Transform, Switches in the middle
- osg::Geode/Billboard Nodes are the leaf nodes, which contain:
  - osg::Drawables which are leaves that contain the geometry and can be drawn.
  - osg::StateSets attached to Nodes and Drawables, state inherits from parents only.
Group nodes

- `osg::Group` - Branch node, which may have children, also normally top-node
- `osg::Transform` - Transformation of children
- `osg::LOD` - Level-of-detail selection node
- `osg::Switch` - Select among children
- `osg::Sequence` - Sequenced animation node
- `osg::CoordinateSystemNode` - defines a coordinateSystem for children
- `osg::LightSource` - defines a light in the scene
- And many more..
Leaf nodes

- osg::Geode - "geometry node", a leaf node on the scene graph that can have "renderable things" attached to it.
- In OSG, renderable things are represented by objects from the Drawable class.
- so a Geode is a Node whose purpose is grouping Drawables.
- it is however NOT a group node.
- Other leaf node type osg::Billboard - derived form of osg::Geode that orients its osg::Drawable children to face the eye point.
Drawables

- osg::Drawable itself is a pure virtual class
- everything that can be rendered is implemented as a class derived from osg::Drawable
- A Drawable is NOT a node and cannot be directly added to the scene graph (always through a Geode)
- Like Nodes can be children of several parents, also Drawables can be shared between several Geodes
- the same Drawable (loaded to memory just once) can be used in different parts of the scene graph -> good for performance
Drawable Sub Classes

- osg::Geometry – drawable basic geometry
- osg::ShapeDrawable - allows to draw any type of osg::Shape
- osg::DrawPixels – single pixels
- osgParticle::ParticleSystem – allows to draw a particle system
- osgText::Text – drawable true type text
Drawing basic Geometry

- Drawable osg::Geometry allows drawing basic geometry:
  - Assign to it:
    - a vertex array
    - Primitive sets
      - Can be any of the modes POINTS, LINES, LINE_STRIP, LINE_LOOP, TRIANGLES, TRIANGLE_STRIP, TRIANGLE_FAN, QUADS, QUAD_STRIP, POLYGON
      - Direct encapsulation of OpenGL primitives
      - Contains indices of vertices that form the primitive(s)
    - (optional) color, normal and texture coordinate arrays
Shapes

- Pure virtual base class osg::Shape
- Shapes can be used for culling, collision detection, or be drawn via osg::ShapeDrawable
- Some shape sub-classes:
  - osg::Box
  - osg::Sphere
  - osg::Cone
  - osg::Cylinder
  - osg::Capsule
  - osg::InfinitePlane
  - osg::TriangleMesh
Transformations

- Transformation = Translation, Rotation and Scaling
- Base class osg::Transform provides basic Transformation via 4x4 Matrix
- Often better use more accessible subclasses though
- Most important sub class:
  - osg::PositionAttitudeTransform – sets the coordinate transform via a vec3 position and scale and a quaternion attitude
A simple example scene graph

- One box and two spheres

```
osg::Group
  osg::Geode
    osg::PositionAttitudeTransform
      osg::Box
      osg::ShapeDrawable
    osg::PositionAttitudeTransform
      osg::Sphere
      osg::ShapeDrawable
```
**StateSets**

- Stores a set of modes and attributes which represent a set of OpenGL state
- Can be attached to any Node or Drawable
- Defines drawing state for node and its subtree
- Drawing state is always inherited from parents, unless it is overridden
- State’s affect the way OpenGL renders, so the appearance of objects
- For example: textures, fog, transparency …
State Set Example

BLEND Texture Mode

(1) DECAL Texture Mode

(2) FOG, ON, OVERRIDE

(3) FOG, OFF

(4) FOG, OFF, PROTECTED

(5) Change Texture

(no change)
Smart Pointers

- Instead of standard pointers to osg objects, use `osg::ref_ptr<>` template
- Provides a smart pointer that automatically counts references
- Object is removed from memory if reference count drops to zero
- Similar to Java Garbage collection, helps keeping the memory free and simplifies programming
- Example:
  - Dumb pointer: `osg::Group *group1 = new osg::Group();`
  - Smart pointer `osg::ref_ptr<osg::Group> group1 = new osg::Group();`
Third Party Dependencies

- To support multi platform functionality, the open scene graph distribution includes 3rd party libraries:
  - Open Threads for platform independent threads
  - Producer for a platform independent viewer
  - And several file format plugins
Standard steps

- 1. Create a Producer based viewer
- 2. configure the viewer
- 3. Load or create a scene graph, and associate its top node with the viewer
- 4. (optional) optimize the scene graph
- 5. update the scene
- 6. draw the scene
- 7. Create the simulation loop, which loops between 5. and 6.
The simulation loop

Three main steps:

– Update the scene, e.g. location of an object
  • It may be moving

– Update the camera, e.g. zoom in on scene
  • The position of the user for example
  • May require interaction with input devices
  • Normally just the viewer’s update method is called, standard viewer already implements basic mouse camera control
  • non-standard interaction (i.e. other input devices, 1st person cam, etc.) would ideally be implemented in a customized viewer class

– Redraw the frame
Importing 3d-Models

- osgDB library responsible for reading/loading 3d-model-files
- File format plug-ins (loaders) are registered with osgDB
- In your application, no matter which supported file format always use the same function osgDB::readNodeFile, file extension tells osgDB, which loader to use
- Function returns an osg::Group pointer
- Best file format to use: osg’s native format *.osg
- Can quickly save any scene graph in a *.osg file with: osgDB::writeNodeFile
Importing VRML

- VRML loading is handled by Inventor plug-in
- Not part of standard Open Scene Graph distribution, need to compile and register first
- Easier way: use 3D Studio Max to convert wrl file to 3ds file
- 3ds files can be loaded by standard osg distribution
- Whichever way is used, not all VRML is imported, because not everything in a VRML file belongs in a scene graph (e.g. scripts, animations)
Optimization

- You can optimize the scene graph to improve performance
- Use osgUtil::Optimizer
- Makes especially sense for huge loaded models
- Optimization will rearrange scene graph, don’t optimize parts, that you want to modify at runtime, scene graph structure might change
- How can a scene graph be optimized:
  - By removing redundant nodes
  - By minimizing state changes
  - By using more efficient geom. Primitives (e.g. tristrips)
  - ...
Examples

- Jason McVeigh's OpenSceneGraph Tutorial Set.
- http://openscenegraph.org/documentation/NPSTutorials/
Example 1

- Loading geometric models from files and positioning them in a scene
Example 1

root
(osg::Group)

tankXform
(osg::PositionAttitudeTransform)

tankNode
(osg::Node)
Example 2

- Finding named nodes, updating DOF and switch nodes
Example 2

- root (osg::Group)
  - tankOneGroup (osg::Group)
  - tankTwoGroup (osg::Group)
    - tankTwoPAT (osg::PositionAttitudeTransform)
  - tankThreeGroup (osg::Group)
    - tankThreePAT (osg::PositionAttitudeTransform)
Example 3

- Using an update callback to articulate a node within a scene

Articulate tank using a Callback
Manually positioning a camera

1. Create and initialize a matrix with the correct world position and orientation.
2. Get the inverse of this matrix and ...
3. Provide a world up orientation. In this case by rotating from ‘Y’ up to ‘Z’ up.
Example 5

- Using tracking devices
Available Resources

- www.openscenegraph.org
- OpenSceneGraphReferenceDocs.zip
- Tutorials
- Examples
- Source Code
- Mailing List Archives