OpenSceneGraph

Ryan Kroiss

Object-Oriented Analysis and Design
CSCI 5448 - Spring 2011

What is it?

- High-performance 3D graphics toolkit
- Object-oriented framework on top of OpenGL
- Open source
- Cross platform
- Based on "scene graph" concept

What is a scene graph?

- Data structure used to describe a graphical scene
- Typically a directed, acyclic graph (DAG)
- Nodes represent aspects of scene
  - Leaf nodes represent geometry to be rendered
  - Non-leaf nodes establish hierarchy and scene manipulations

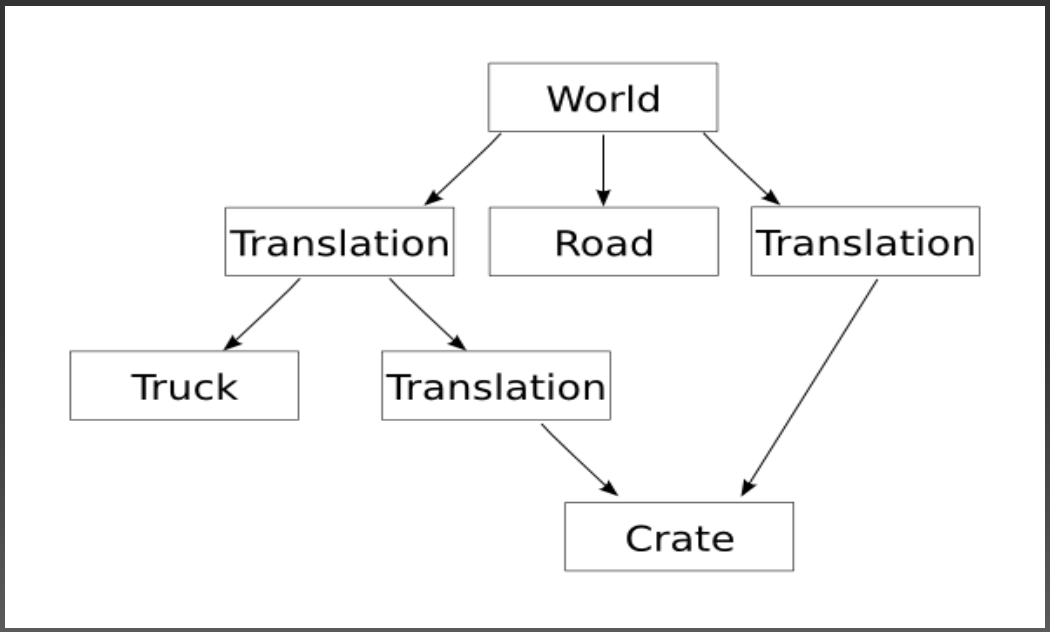


Figure 1: Scene graph for a scene with a truck on a road with one crate in the truck and another on the road

Scene Graph in OSG

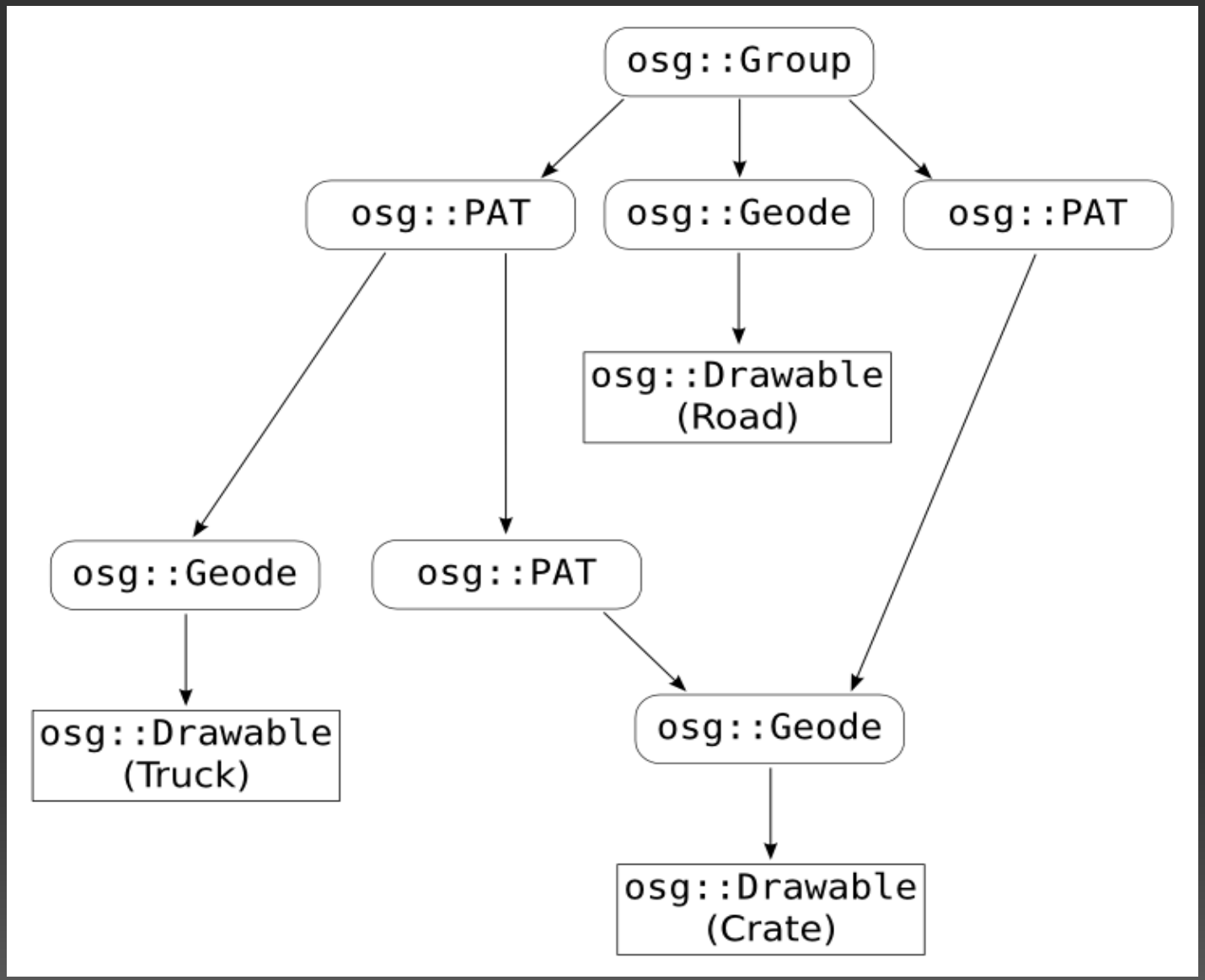
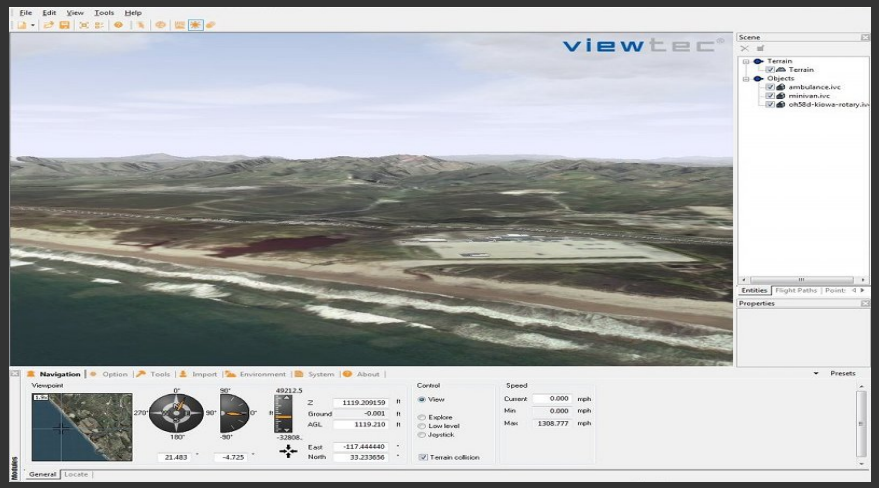


Figure 2: OSG representation of scene graph in Figure 1

Common Uses of OSG

Flight simulators (Flightgear)



Visualization (TerrainView)

Games (Pok3D)

Virtual Reality (J3Tech - J3Reality)

Brief History

- 1998: Started as hobby project by Don Burns
- 1999
  - Robert Osfield joined the team
  - Code became open source
- 2000: Development gets serious
- 2001: Don and Robert form separate companies for commercial support of OSG
- 2002: Community involvement gets serious
- 2003: Development focused on scalability

Future Plans

- Mid-term goals
  - Improve shadow support
  - Develop node kit for 3D GUI
  - Develop node kit for volume rendering
  - Support geometry shader
- Long-term goals
  - Integrate OpenGL ES
  - Integrate OpenGL 3.0

Getting started with OSG

- Download here
- Obtain pre-reqs/dependencies
  - CMake
  - Appropriate compiler for your platform
- Configure; make; make install
- Experiment with example applications
- Start coding!

Important Classes

- osg::Node
  - Node in a scene graph
  - Important subclasses
    - osg::Geode
    - osg::Group
- osg::Drawable
  - Actual rendered geometry
  - Not a node in scene graph - must be attached to an osg::Node
  - Important subclasses
    - osg::Geometry
    - osg::ShapeDrawable
- osgViewer::Viewer
  - Controls viewing of scene

Basic OSG Program

```
#include <osg/Group>
#include <osg/ShapeDrawable>
#include <osgViewer/Viewer>
#include <osgGA/TrackballManipulator>

int main() {

    // root node in scene graph
    osg::Group* root = new osg::Group();
    // unit cube centered at origin (Box class derived from Shape)
    osg::Box* cube = new osg::Box(osg::Vec3(0,0,0), 1.0f);
    // associate Shape with Drawable
    osg::ShapeDrawable* cubeDrawable = new osg::ShapeDrawable(cube);
    // create geode and add cube drawable to it
    osg::Geode* cubeGeode = new osg::Geode();
    cubeGeode->addDrawable(cubeDrawable);
    // add cube geode to root node
    root->addChild(cubeGeode);

    // turn off lighting since normals are not defined
    root->getOrCreateStateSet()->setMode(GL_LIGHTING,
                                         osg::StateAttribute::OFF);

    // create viewer
    osgViewer::Viewer viewer;
    // hand scene graph off to viewer for rendering
    viewer.setSceneData(root);
    // set up camera manipulator
    viewer.setCameraManipulator(new osgGA::TrackballManipulator());
    // set up window
    viewer.realize();

    // render new frames until exit input is received
    while(!viewer.done())
        viewer.frame();
    // return to OS
    return 0;
}
```

Figure 3: OSG program to draw a cube

Basic OSG Program (2)

```
// root node in scene graph
osg::Group* root = new osg::Group();
// unit cube centered at origin (Box class derived from Shape)
osg::Box* cube = new osg::Box(osg::Vec3(0,0,0), 1.0f);
// associate Shape with Drawable
osg::ShapeDrawable* cubeDrawable = new osg::ShapeDrawable(cube);
// create geode and add cube drawable to it
osg::Geode* cubeGeode = new osg::Geode();
cubeGeode->addDrawable(cubeDrawable);
// add cube geode to root node
root->addChild(cubeGeode);
```

- Set up scene graph
  - Define root node
  - Define elements of scene graph
  - Make shape drawable
  - Attach drawable shape to graph node
  - Add graph node to scene graph
- Groups allow manipulation to cascade
- Scene graph can obviously be much more complex

Basic OSG Program (3)

```
// turn off lighting since normals are not defined
root->getOrCreateStateSet()->setMode(GL_LIGHTING,
                                     osg::StateAttribute::OFF);
```

- Define the StateSet for this Group
  - Here, simply turn off lighting
- StateSets define the OpenGL state for a node when it is rendered
  - Textures
  - Culling
  - Lighting
  - etc.
- OSG can sort nodes at render time to minimize state switching

Basic OSG Program (4)

```
// create viewer
osgViewer::Viewer viewer;
// hand scene graph off to viewer for rendering
viewer.setSceneData(root);
// set up camera manipulator
viewer.setCameraManipulator(new osgGA::TrackballManipulator());
// set up window
viewer.realize();
```

- Set up the view of the scene
  - Set the scene graph for the viewer
  - Set the camera manipulator
  - Set up the windows
- Viewer holds a single view of a single scene
- Note how easy it is to define a camera manipulator!

Basic OSG Program (5)

```
// render new frames until exit input is received
while(!viewer.done())
    viewer.frame();
// return to OS
return 0;
```

- Turn control over to OSG
- Viewer listens for signals to exit

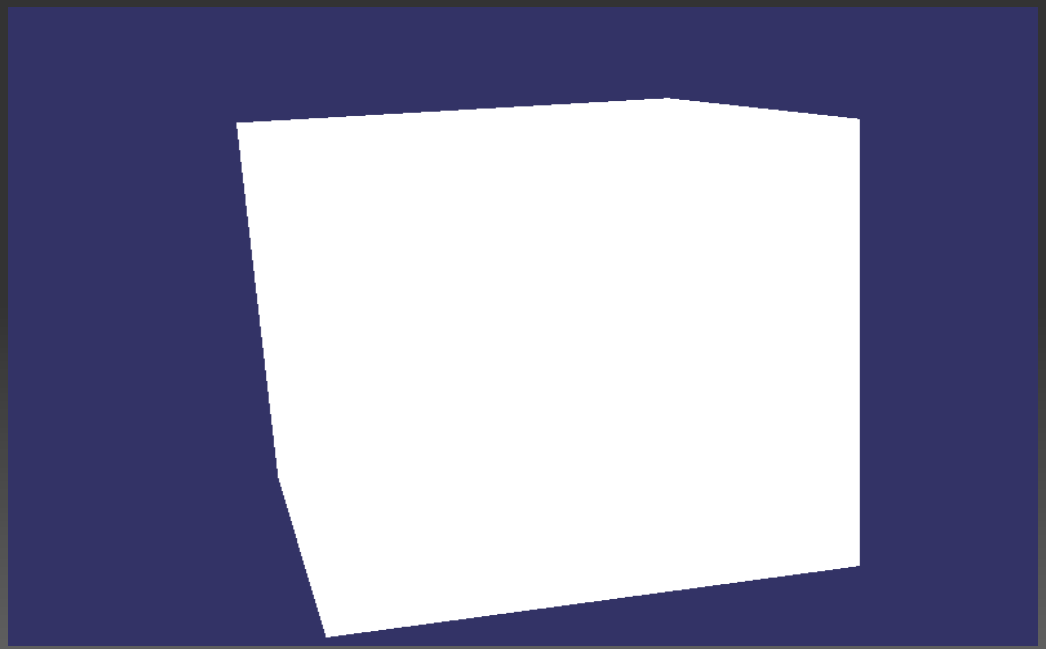


Figure 4: Output of program (boring, but only 20 lines of code)

Basic OSG Program Summary

- ~20 lines of C++ code
- Render 3D object with one camera that can be manipulated via the mouse
- Important concepts
  - Scene graph structure
  - StateSets
  - Viewers
  - Camera manipulation
- Barely scratching the surface of OSG's potential!

Features

- Written in C++ and OpenGL
- Uses Standard Template Library (STL) and Design Patterns
- Advantages
  - Performance
  - Productivity
  - Data loaders
  - Node kits
  - Portability
  - Scalability
  - Multi-language support

Design Patterns in OSG

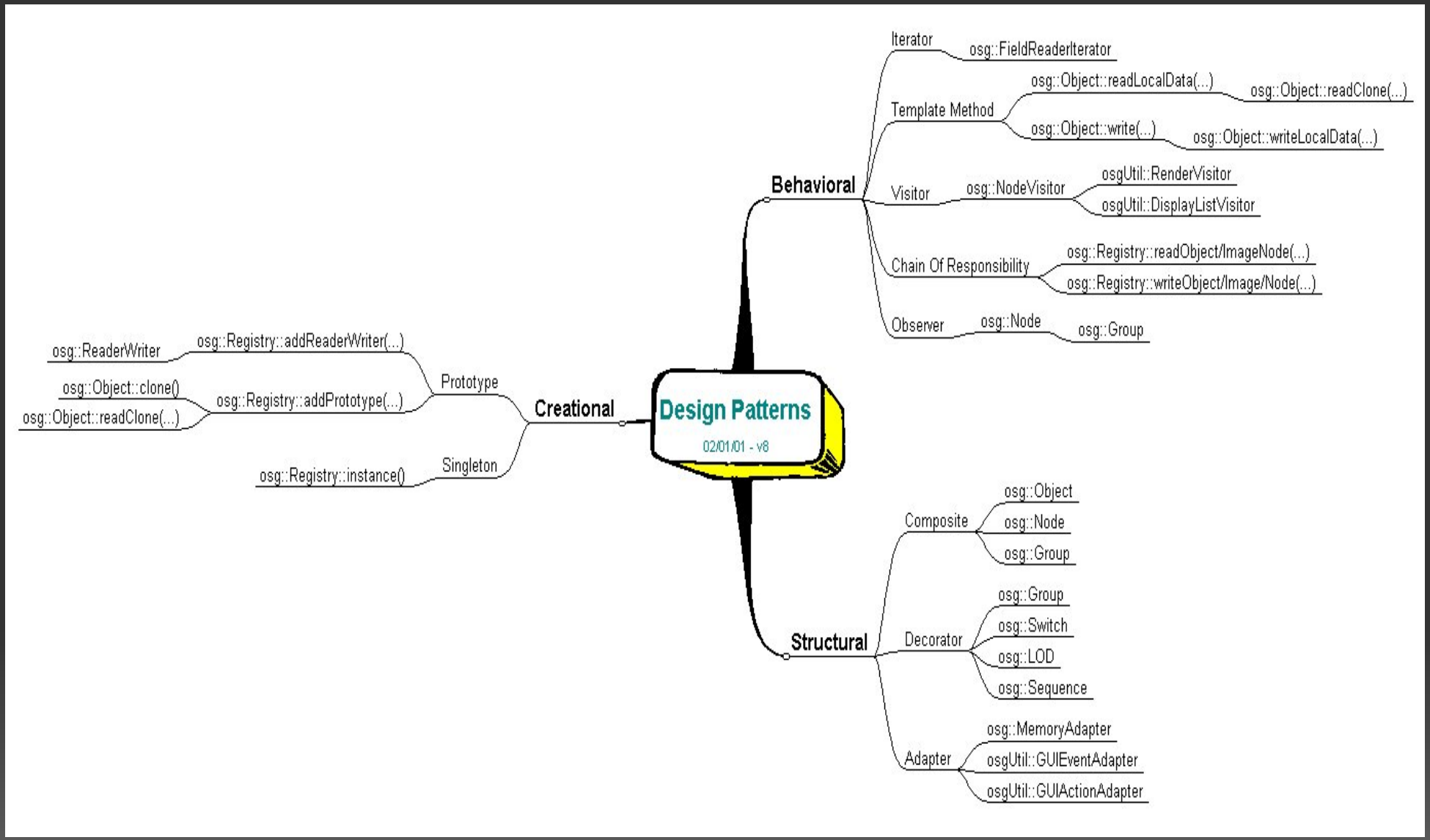


Figure 5: Design patterns in OSG

Design Patterns in OSG (2)

- If you don't know why design patterns are useful, then you haven't been paying attention in class!
- UML diagram of OSG
- Noteworthy patterns for OSG developers
  - Visitor - traversal and rendering of scene graph
  - Observer - notifications for groups of nodes
  - Decorator - dynamically change behaviors of scene graph

Performance

- Culling: view-frustum, occlusion, small feature
  - Don't draw features that you can't see
- Vertex arrays
  - Reduce number of function calls
  - Let OSG/OpenGL do the work
- Level of Detail nodes
  - Don't render detail when you don't need it
- OpenGL Shader Language
  - Harness the parallelism of your GPU
- Display lists
  - "Compile" complex, frequently used geometry for faster rendering

Data loaders

- Supports loading of graphical models
- Extensible plugin mechanism for new loaders
- Supported formats
  - COLLADA
  - LightWave
  - WaveFront
  - DirectX
  - 3D Studio MAX
  - and many more!
- Terrain loaders

Figure 6: Model to load in OSG

Node Kits

- Libraries that can be used with OSG
- Allows for more advanced graphics
  - osgParticle - particle systems
  - osgFX - special effects
  - osgShadow - shadows
  - osgManipulator - 3D interactive control
  - osgAnimation - character animation
  - and more...
- Possible uses
  - Skeletal animation
  - Bump mapping - makes flat surfaces appear bumpy
  - Animate explosions
- Community can develop node kits too

Portability

- Only requires standard C++ and OpenGL
- Ported to many platforms
  - Linux
  - Windows
  - Mac OS X
  - even Playstation 2
- Independent of windowing system
  - Can be used with Qt, Cocoa, GLUT, etc.

Scalability

- Supports multiple graphics contexts
- Uses mostly read-only operations to draw scene graph
- Multi-threaded
- Enables OSG to scale
- Mobile device -> Multi-core/multi-GPU -> clusters

Multi-language support

- Bindings for various languages
  - Python
  - Lua
  - Java
- Available as community projects

Drawbacks to OSG

- Latest version of OpenGL not supported (yet)
- Restricts development to a particular data model
- Debugging is difficult (as with graphics in general)
- Less fine-grained than using OpenGL alone
- Assumes it's smarter than you
  - Overrides your settings in certain situations
- Custom keyboard bindings can be difficult to set up

Alternatives to OSG

- OGRE (Object-Oriented Graphics Rendering Engine)
- Irrlicht Engine
- Delta3D
- OpenSG (different from OpenSceneGraph)
- Crystal Space
- Visualization Library
- Plain, old OpenGL

Conclusion

- High-performance 3D graphics toolkit
- Open source, cross platform
- Based on concept of scene graph
- Program structure straight-forward
- Scalable and portable
- Highly extensible

References

- www.openscenegraph.org
- http://www.stackedboxes.org/~lmb/asittbpo-open-scene-graph/
- www.blendswap.com