OpenSceneGraph

Design Issue

Based on materials from http://www.openscenegraph.org/

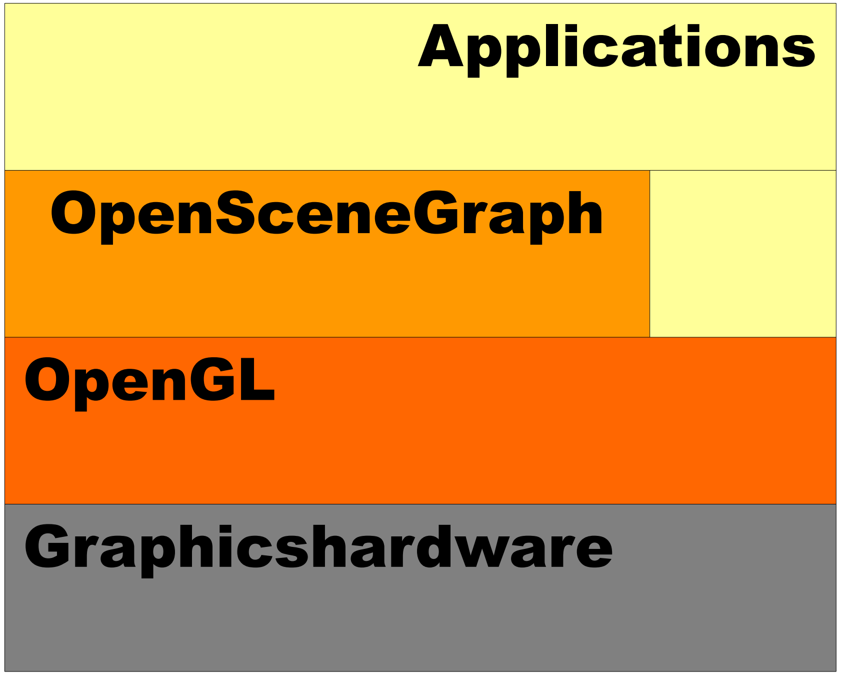
Topics

- What is Open Scene Graph?
- Why Open Source?
- Open Scene Graph design concepts
- Who and how

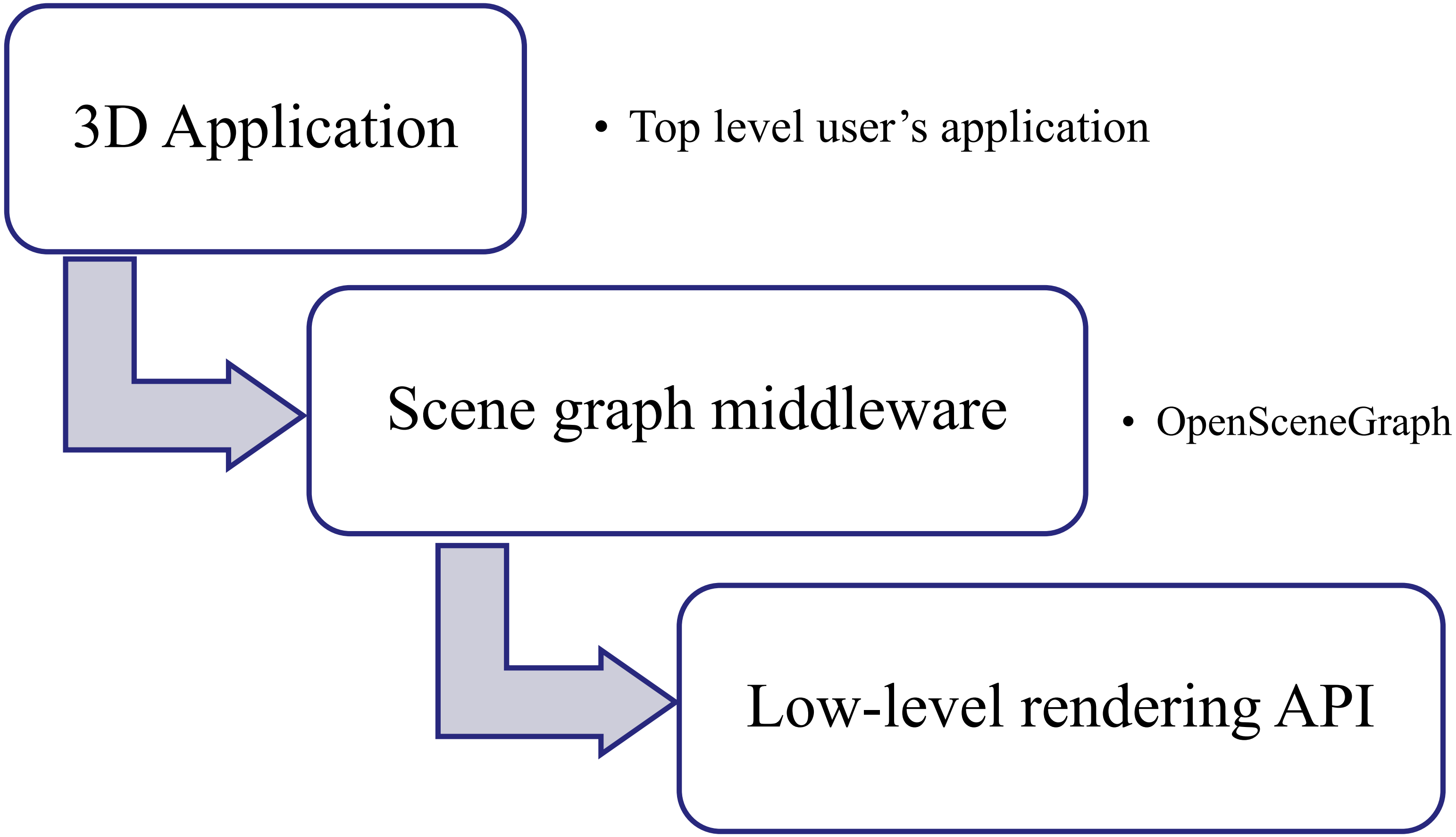
What Is Open Scene Graph?

- A C++ API built on OpenGL for
  - Scene Management
  - Graphics Rendering Optimization
- Cross-platform
- Open Source

Layers



OpenSceneGraph as a “middleware”



3D Application

- Top level user’s application

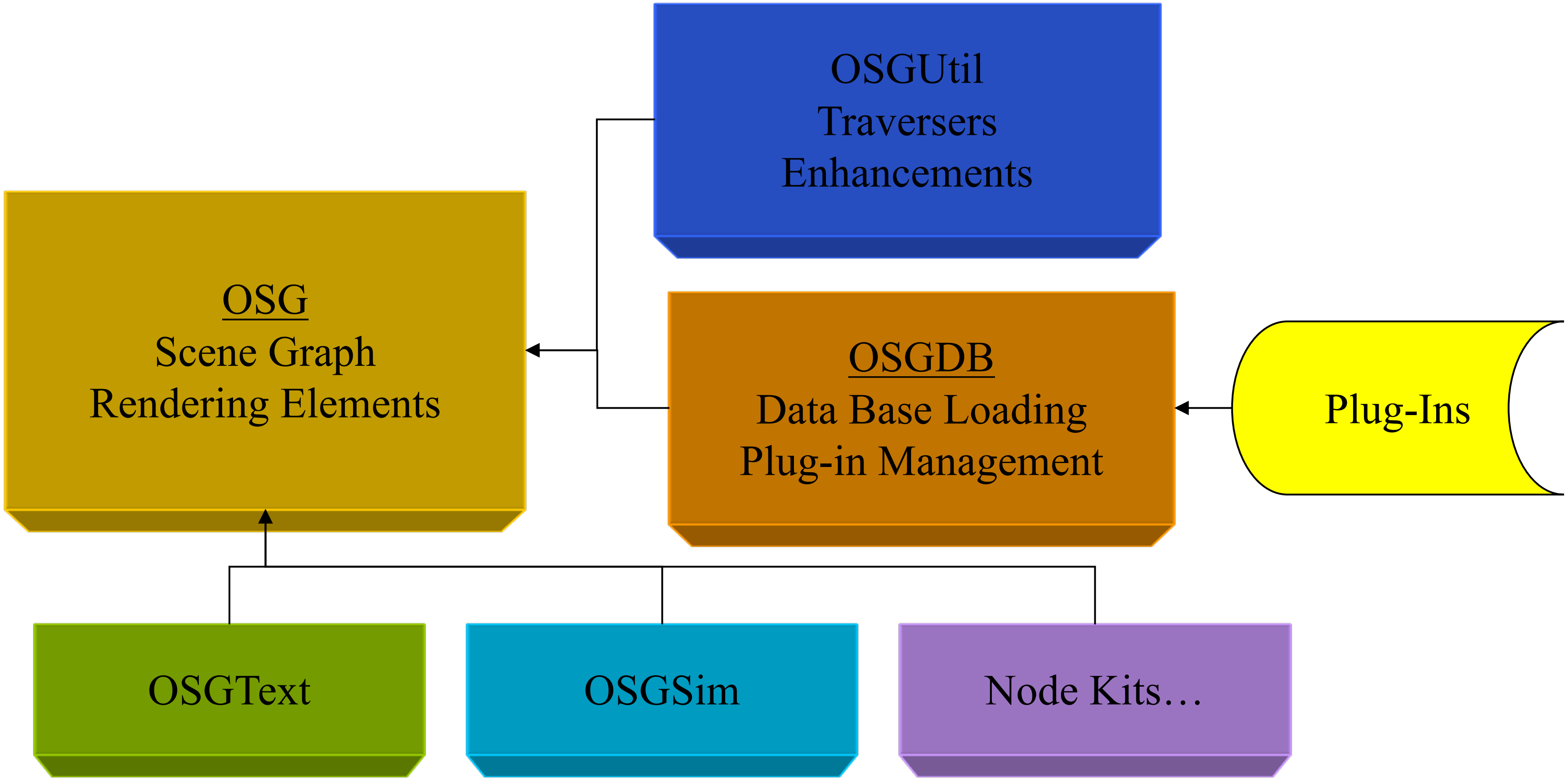
Scene graph middleware

- OpenSceneGraph

Low-level rendering API

- OpenGL

Functional Components



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
- OSGText – Node Kit which add support for TrueType text rendering
- OSGSim – Visual simulation Nodekit

What is in it? – The libraries (2)

- OSGParticle - NodeKit which adds support for particle systems
- OSGTerrain – Terrain generation Nodekit

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;)

File Formats Supported

File Formats

3dc 3ds ac3 dw flt Freetype

iv ive logo lwo md2 obj

osg osgtgz tgz txp directX zip

Image Formats:

bmp dds pic png

pnm qt rgb tga

Why Open Source?

Concept

Design & Implementation

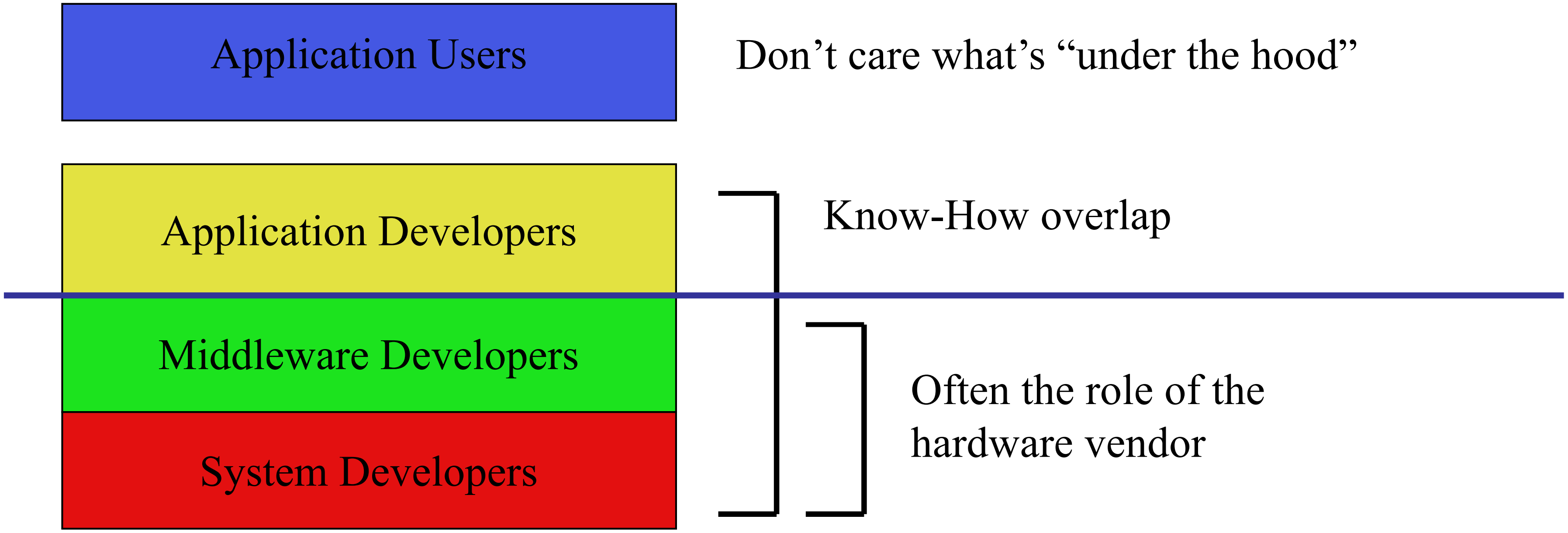
Source Code

Value of software is here

Not here

Why Open Source?

The software “food chain”



Application Users

Don’t care what’s “under the hood”

Application Developers

Know-How overlap

Middleware Developers

System Developers

Often the role of the hardware vendor

Why Open Source?

- Free of intellectual property concerns
- Free of business model restrictions
- Benefits the application developer
- Benefits the middleware developer
- Improved software quality

Crucial Elements for Open Source Success

- Quality
  - Usefulness
  - Stability
  - Design

- Support
  - Responsiveness
  - Thorough
  - Courtesy and Friendliness

Design by Evolution, Evolution by Design

- Adaptive development
- Key Factors
  - Portability
  - Extensibility
  - Scalability
  - Flexibility

Who is using OSG?

- Magic Earth - Geoprobe® - Oil & Gas
- Boeing - Flight simulation
- Indra - Train simulation
- STN Atlas - Simulation
- NASA - Earth visualization
- Norcontrol - Maritime simulation
- Real World Entertainment - Gaming (Releasing Java Bindings)
- Terrex - LOD Paging

Graphics Programming

OpenGL

- Low-level API
- cross-language
- cross-platform
- 2D, 3D computer graphics

OpenSceneGraph

- Higher level, built upon OpenGL
- Written in standard C++
- Windows, Linux, Mac and few more
- 2D, 3D computer graphics

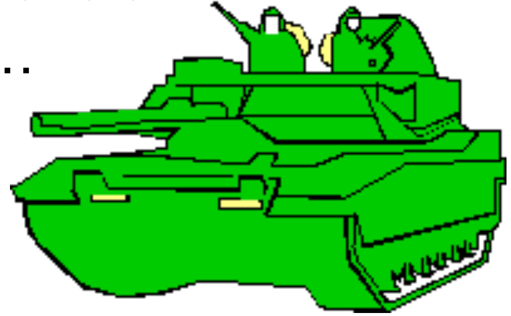
OpenSceneGraph

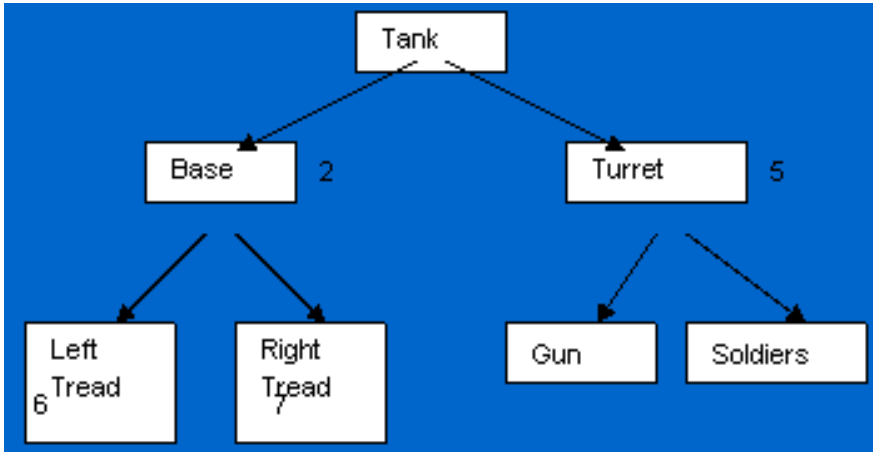
Image Processing

Based on materials from http://www.openscenegraph.org/

Scene Graphs

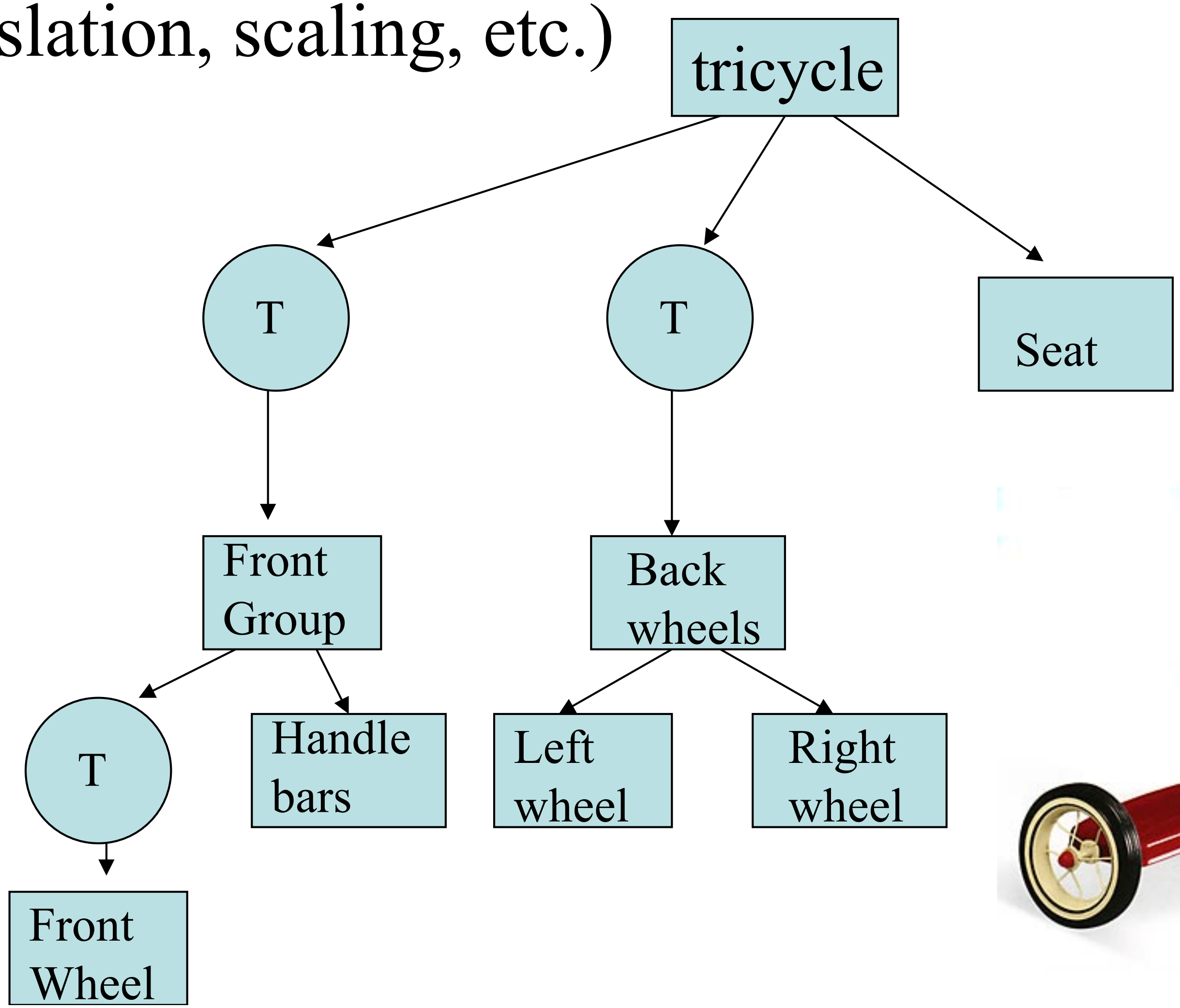
- Data structure: Directed Acyclic Graph (DAG)
  - Usually a tree (only one parent per node)
  - Represents object-based hierarchy of geometry
- Leaves contains geometry (triangles, etc.)
- Each node holds pointers to children
- Children can be
  - Group
  - Geometry
  - Matrix transform
  - Others…





Scene Graphs

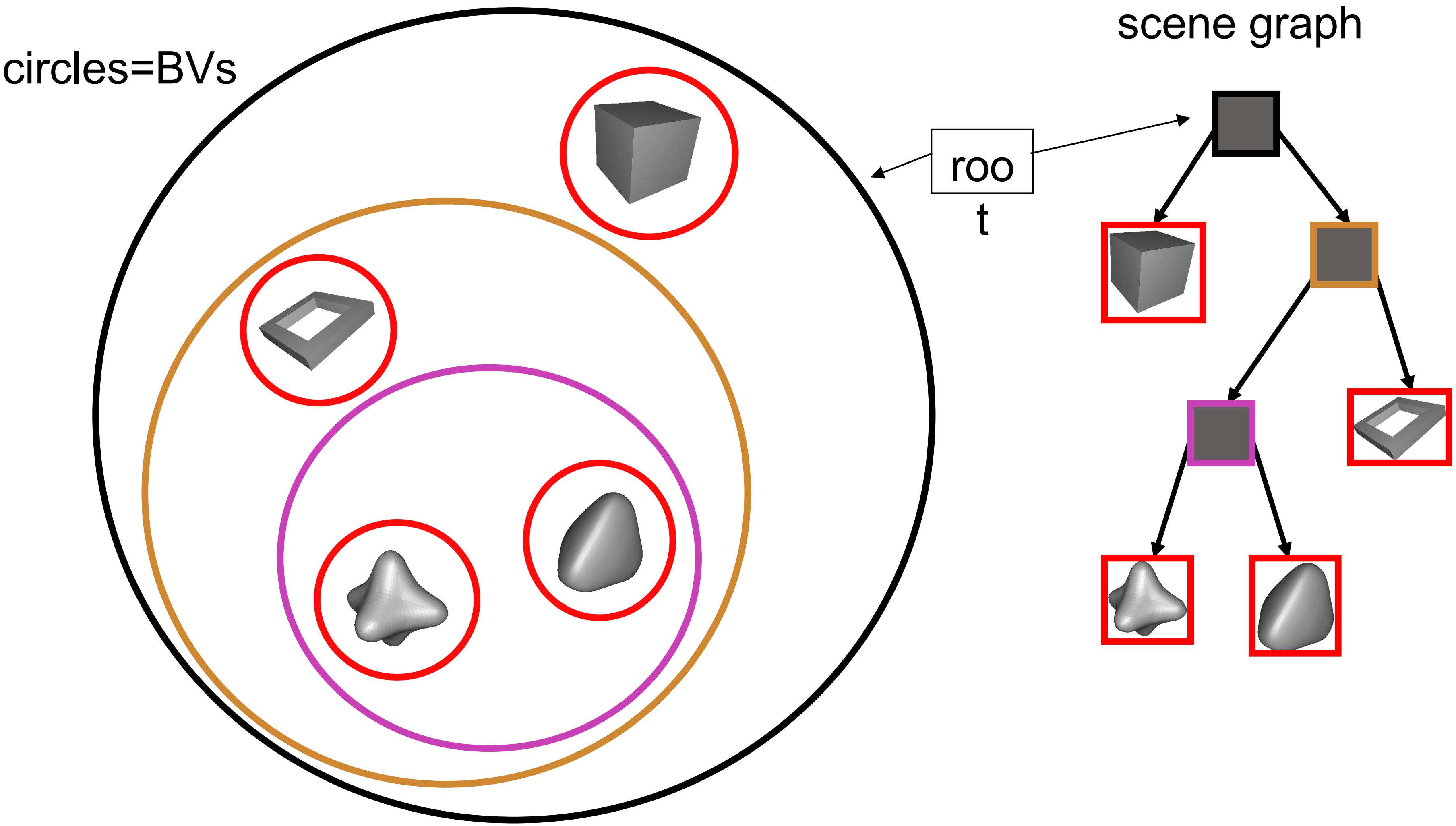
- Spatial transforms represented as graph nodes (rotation, translation, scaling, etc.)



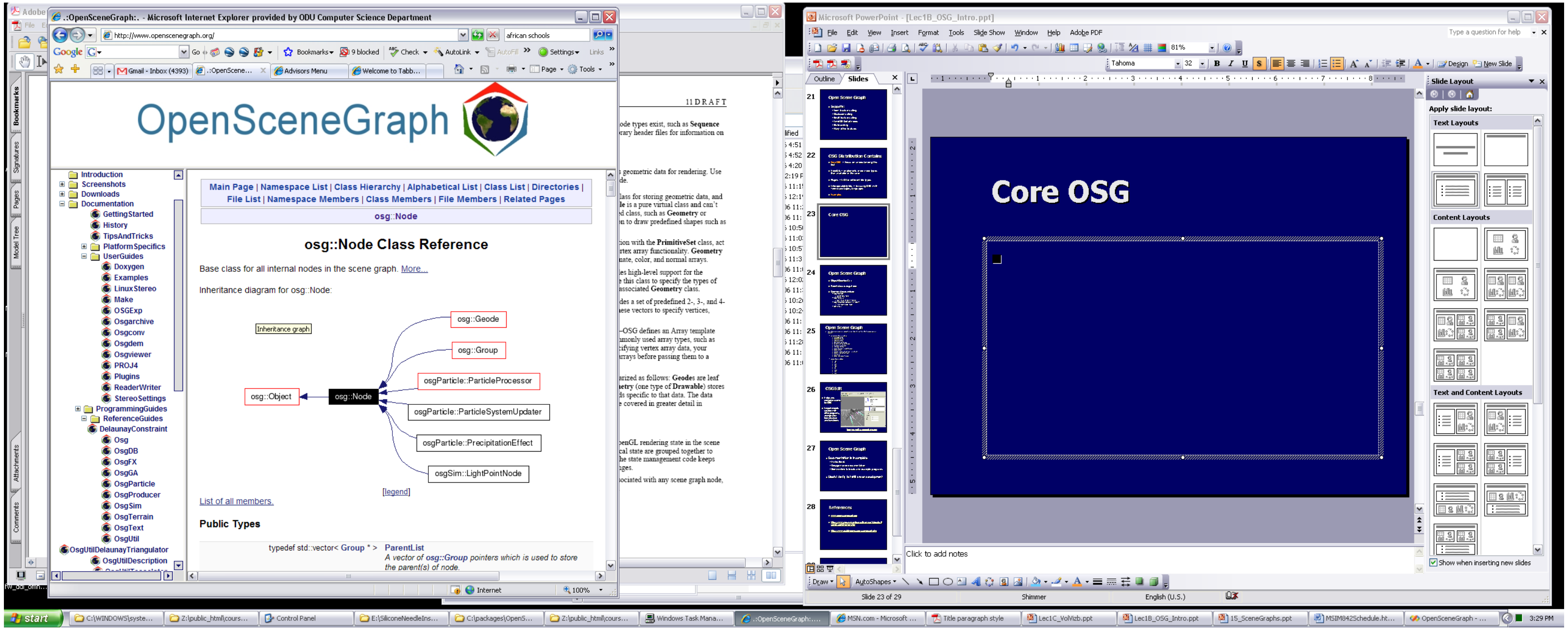
Scene Graphs & Bounding Volumes

- Basic idea:
  - Augment scene graphs with bounding volume data (spheres or blocks) ***at each node***
    - Sometimes called “Bounding Volume Hierarchy” (BVH)
  - By applying clipping/culling tests to the bounding volumes, prune entire branches of the tree and possibly avoid processing many triangles

Scene graph example



Core OSG



Main Page | Namespace List | Class Hierarchy | Alphabetical List | Class List | Directories | File List | Namespace Members | Class Members | File Members | Related Pages

osg::Node

osg::Node Class Reference

Base class for all internal nodes in the scene graph. More...

Inheritance diagram for osg::Node:

Inheritance graph
osg::Geode
osg::Group
osgParticle::ParticleProcessor
osg::Object
osg::Node
osgParticle::ParticleSystemUpdater
osgParticle::PrecipitationEffect
osgSim::LightPointNode
[legend]

List of all members.

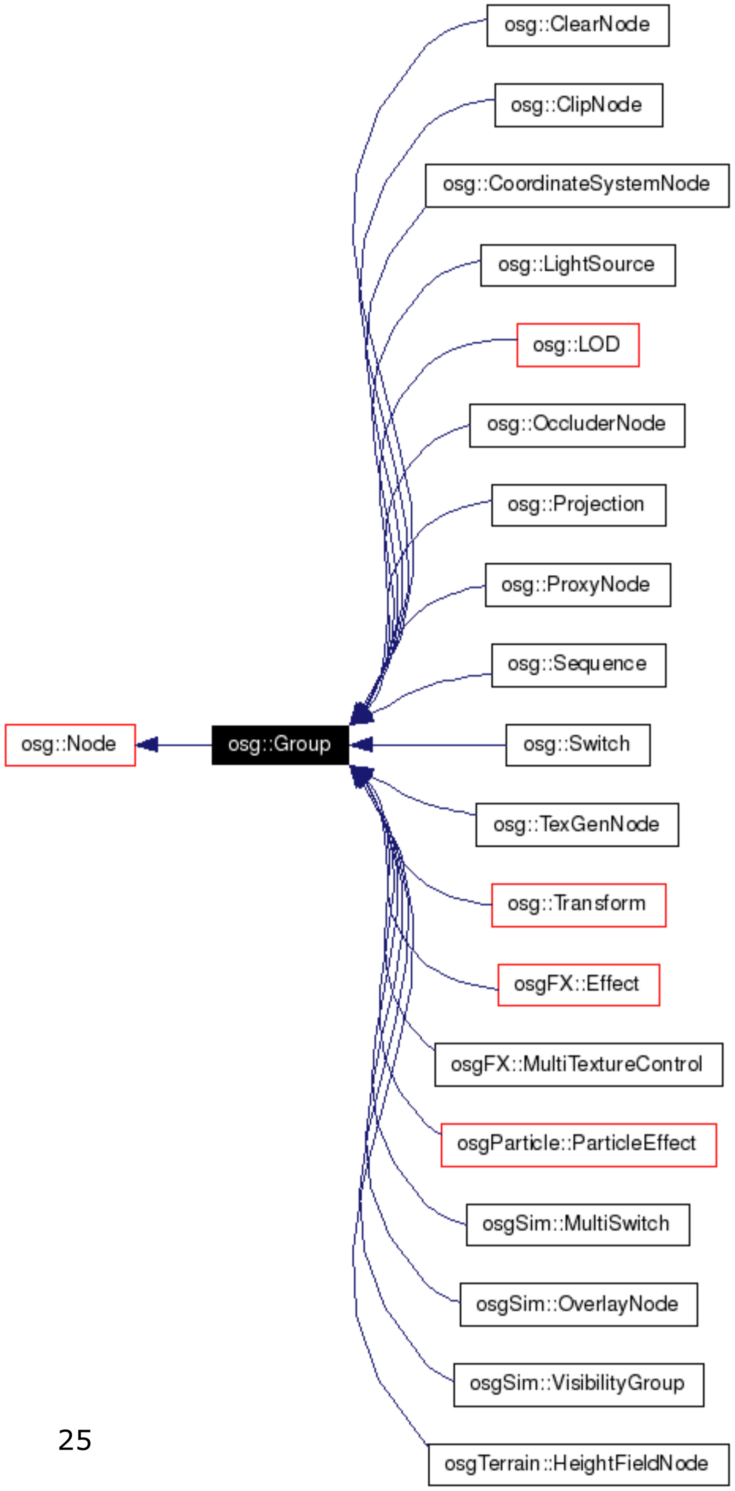
Public Types

typedef std::vector< Group * > ParentList
A vector of osg::Group pointers which is used to store the parent(s) of node.

osg::Node - Base class for all nodes in the scene graph.

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.

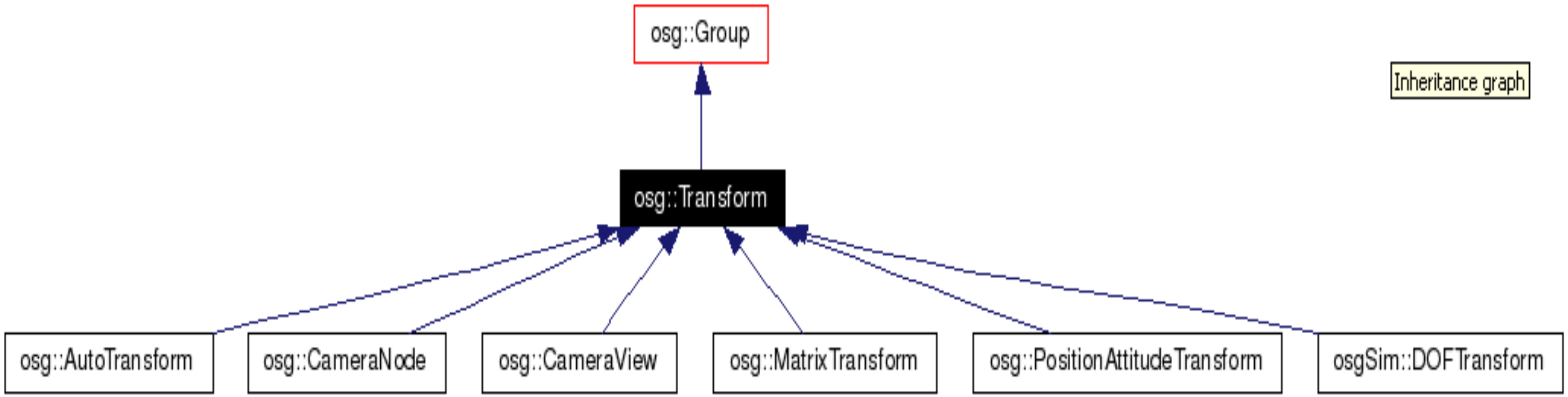


osg::Group - General group node which maintains a list of children.

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

Transform Nodes



osg::Transform - A **Transform** is a group node for which all children are transformed by a 4x4 matrix. It is often used for positioning objects within a scene, producing trackball functionality or for animation.

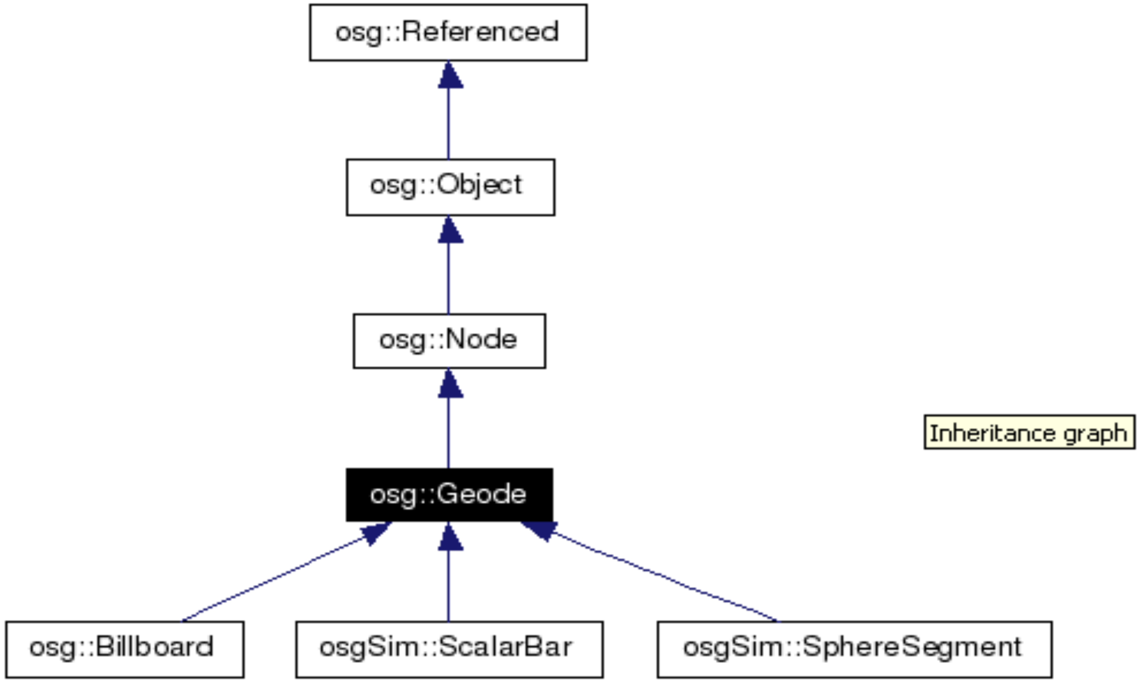
Transformations

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

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.

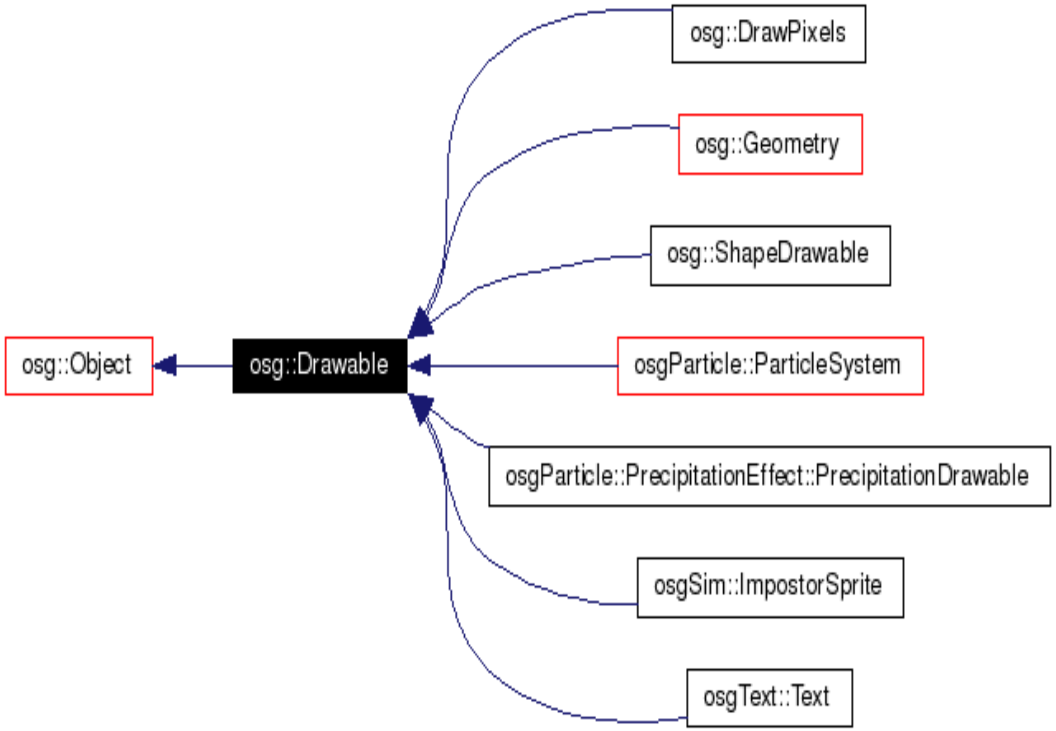
Geometry Nodes



osg::Geode - A **Geode** is a "geometry node", that is, a leaf node on the scene graph that can have "renderable things" attached to it.

Renderable things are represented by objects from the **Drawable** class, so a **Geode** is a **Node** whose purpose is grouping **Drawable**s.

Drawables



Pure virtual base class for drawable geometry.

Everything that can be rendered is implemented as a class derived from **Drawable**.

A **Drawable** is not a **Node**, and therefore it cannot be directly added to a scene graph. Instead, **Drawable**s are attached to **Geode**s, which are scene graph nodes.

The OpenGL state that must be used when rendering a **Drawable** is represented by a **StateSet**.

Drawables can also be shared between different **Geode**s, so that the same geometry (loaded to memory just once) can be used in different parts of the scene graph.

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–singlepixels
- osgParticle::ParticleSystem–allows to draw a particle system
- osgText::Text–drawable true type text

Plugins for file I/O

- Has plugins to support reading/writing lots of graphics file formats and 3D models:
  - 3D database loaders include
    - OpenFlight (.flt)
    - TerraPage (.txp)
    - LightWave (.lwo)
    - Alias Wavefront (.obj)
    - Carbon Graphics GEO (.geo)
    - 3D Studio MAX (.3ds)
    - Peformer (.pfb)
    - Quake Character Models (.md2)
    - Direct X (.x)
    - Inventor Ascii 2.0 (.iv)/ VRML 1.0 (.wrl)
    - Designer Workshop (.dw)
    - AC3D (.ac)
    - native .osg ASCII format.
  - Image loaders include
    - .rgb
    - .gif
    - .jpg
    - .png
    - .tiff
    - .pic
    - .bmp
    - .dds
    - .tga

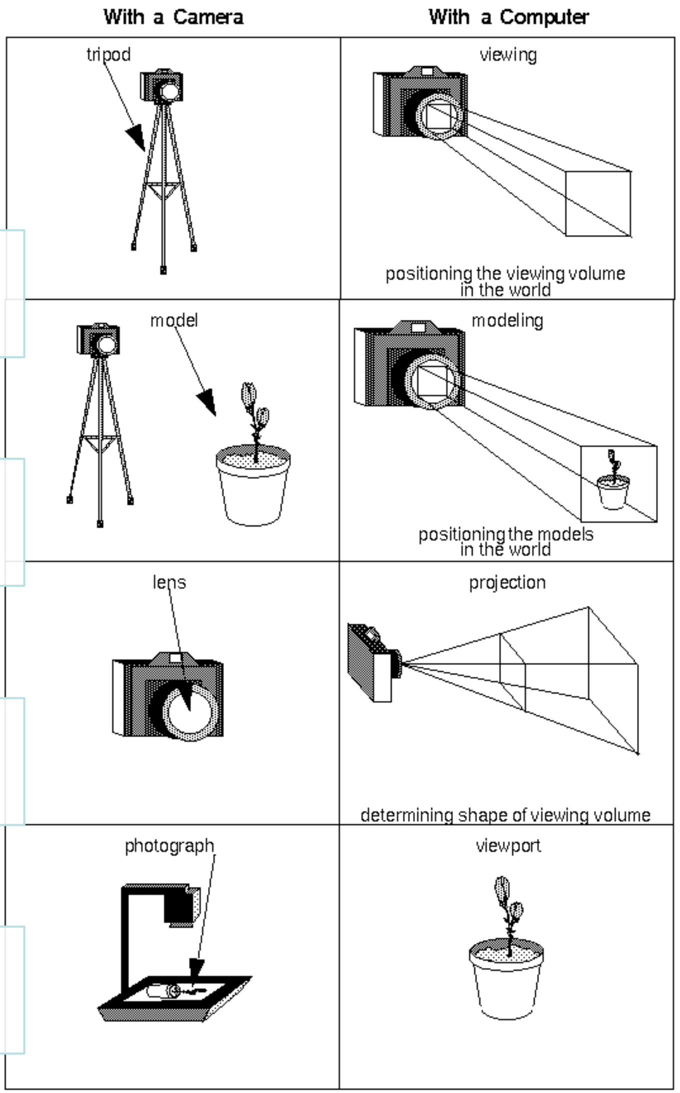
Viewing: Camera Analogy

Positioning the Camera

Positioning the Model

Choose a camera lens and adjust zoom

Mapping to screen


With a Camera
With a Computer
tripod
viewing
positioning the viewing volume in the world
model
modeling
positioning the models in the world
lens
projection
determining shape of viewing volume
photograph
viewport

Viewing Transformation

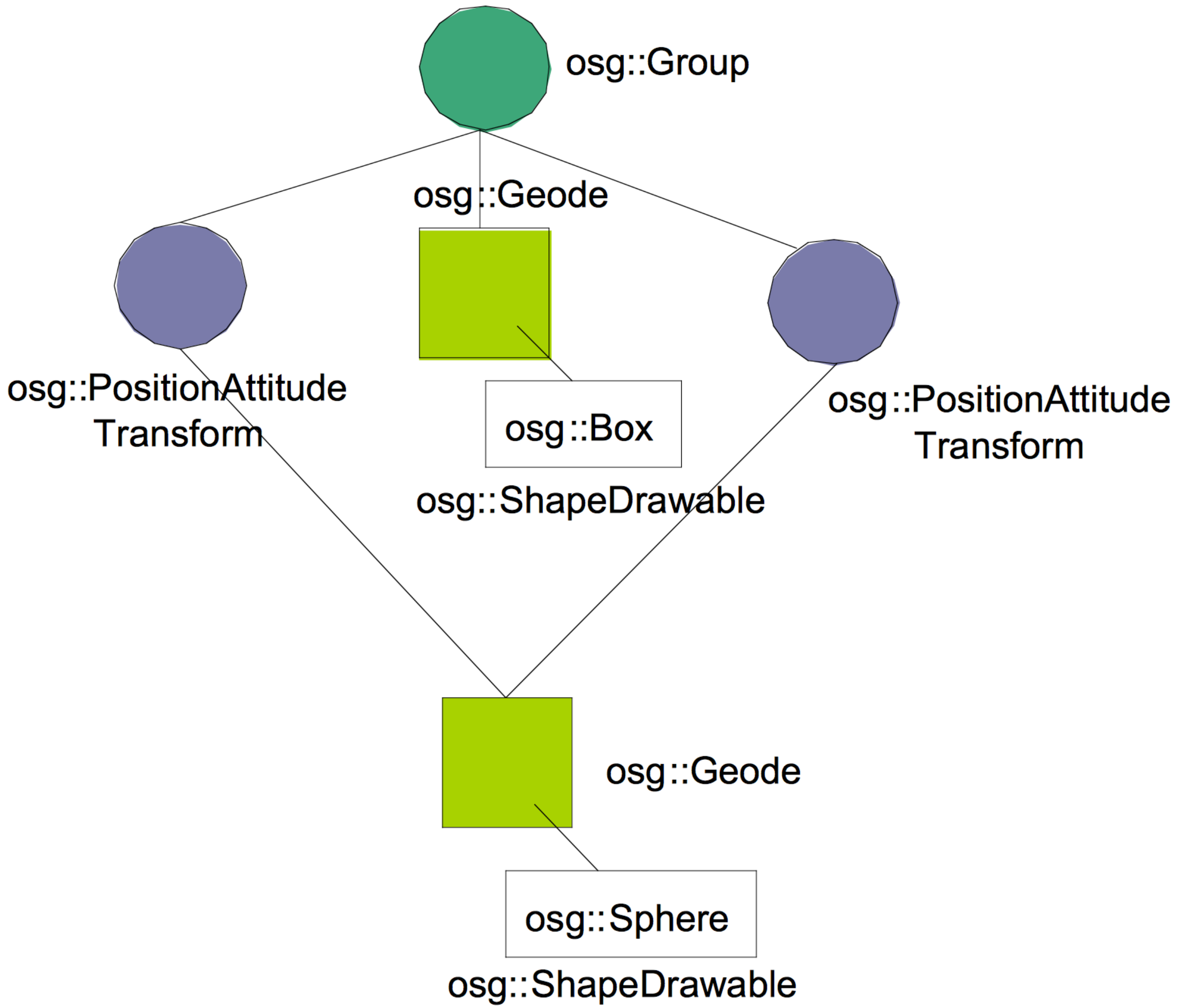
Modeling Transformation

Projection Transformation

Viewport Transformation

A simple example scene graph

- One box and two spheres



Standard steps

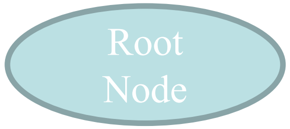
- 1. Create a Producer based viewer
- 2. configure the viewer
- 3. Load or create a scene graph, and associate its top node with the viewer
- 4. (optional) optimize the scene graph
- 5. update the scene
- 6. draw the scene
- 7. Create the simulation loop, which loops between 5. and 6.

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

Building first OSG program

- ex_simple_viewer.cpp


Root Node

```
// load the nodes from the command line arguments.
   osg::Node* model = osgDB::readNodeFile(argv[1]);

// initialize the viewer and set the scene to render
   osgViewer::Viewer viewer;
   viewer.setSceneData(model);
   viewer.setCameraManipulator(new osgGA::TrackballManipulator());

   // normal viewer usage.
   return viewer.run();
```

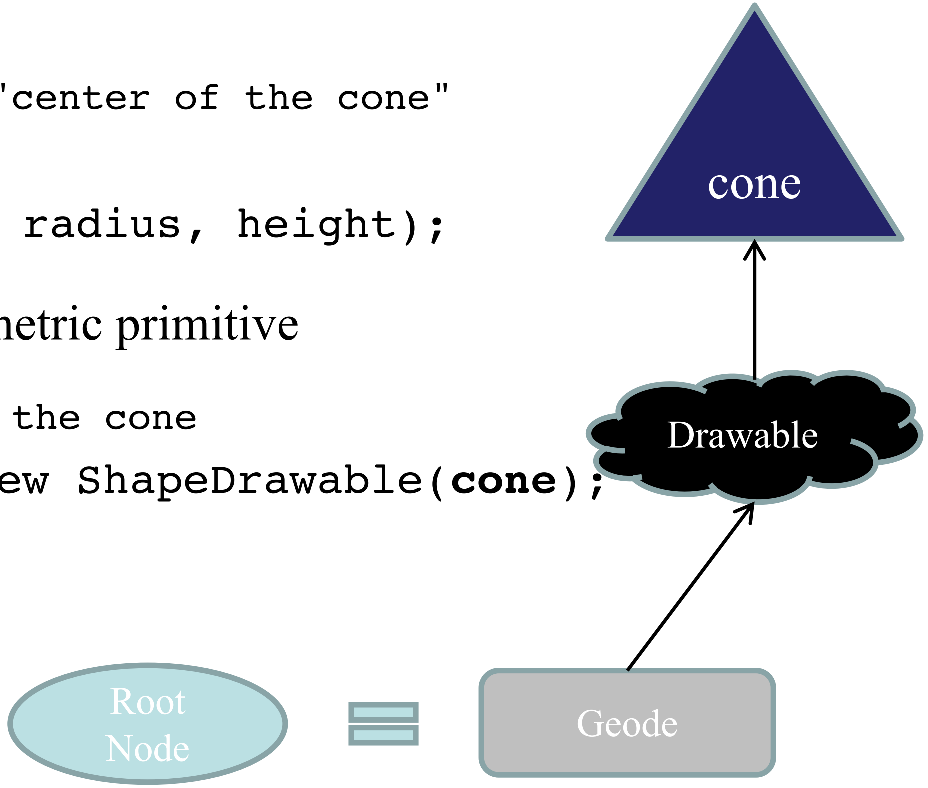
Add geometric primitive

```
// Create a vector to represent the "center of the cone"
Vec3 vcen(xcen, ycen, zcen);
osg::Cone* cone = new Cone(vcen, radius, height);
```

Add geometric primitive

```
// Create a drawable object based on the cone
osg::ShapeDrawable *drawable = new ShapeDrawable(cone);
```

```
// create a new geode (root node)
osg::Geode* geode = new Geode();
geode->addDrawable(drawable);
```



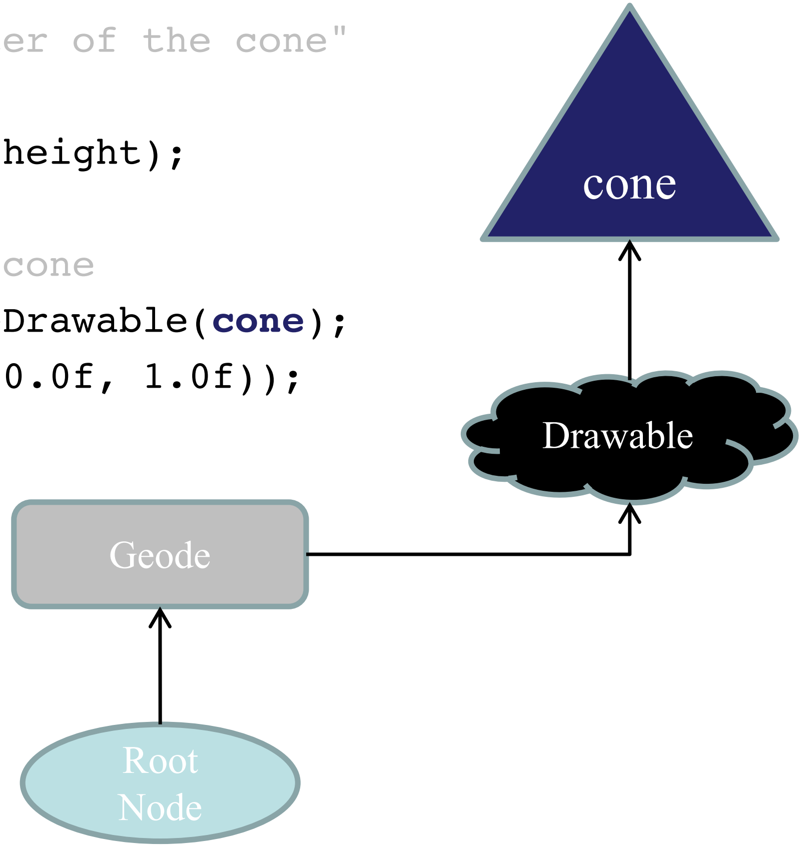
Improving Example

```
// Create a vector to represent the "center of the cone"
osg:: Vec3 vcen(xcen, ycen, zcen);
osg::Cone* cone = new Cone(vcen, radius, height);

// Create a drawable object based on the cone
osg:: ShapeDrawable *drawable = new ShapeDrawable(cone);
drawable->setColor(osg::Vec4(1.0f, 0.0f, 0.0f, 1.0f));

// create a new geode
osg:: Geode* geode = new Geode();
geode->addDrawable(drawable);

// create a root node
osg::Group *root = new osg::Group();
root->addChild(geode);
```



Primitives

OSG comes with a number of primitives

- *Box*
- *Sphere*
- *Cone*
- *Cylinder*
- *Capsule*
- *Special shapes (e.g. InfinitePlane)*

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