OpenSceneGraph Tutorial

Mikael Drugge

Virtual Environments II

Spring 2006

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

Agenda

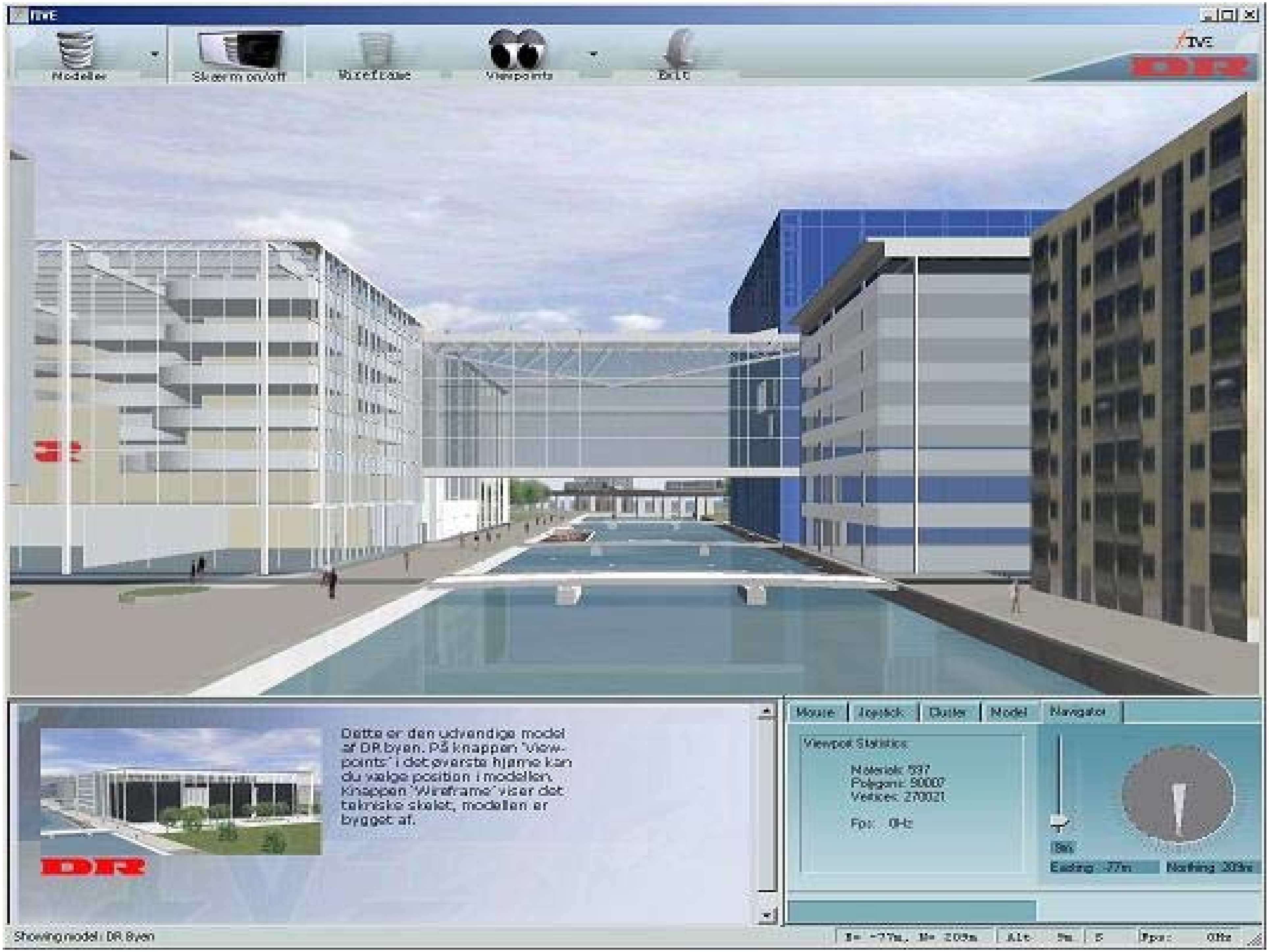
- Introduction to OpenSceneGraph (OSG)
- Overview of core classes
- Creating primitives
- Transformations
- Configuring the viewer
- Event handlers
- Special nodes
- Picking objects

Introduction to OpenSceneGraph

What is OpenSceneGraph?

- A scenegraph system
  - One of the largest in the open source community
- Used for
  - Visual simulation, scientific modeling, games, training, virtual reality, etc...
- Some examples...

PUMA



Dette er den udvendige model af DR byen. På knappen "view-points" i det øverste hjørne kan du vælge position i modellen. Knappen "Wireframe" viser det tekniske skelet, modellen er bygget af.

Viewpoint Statistics:

Materials: 597
Polygons: 90007
Vertices: 270021

Fps: 0Hz

Easting: -77m
Northing: 209m

Showing model: DR Byen

E= -77m, N= 209m | Alt 9m | 5 | Fps: 0Hz

POLICE STATION

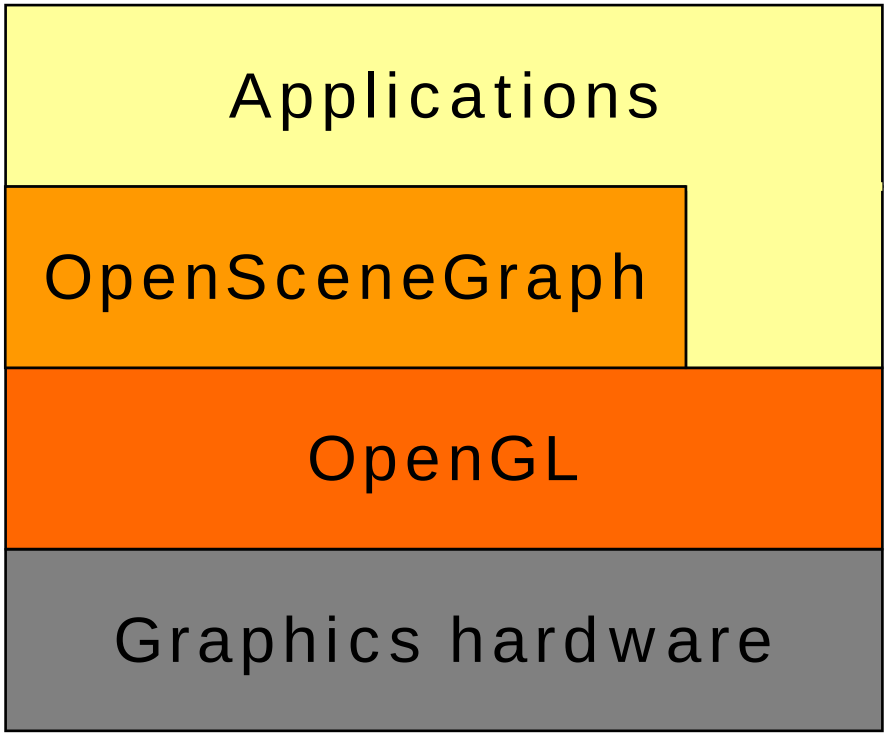
What is OpenSceneGraph?

- Tree structure (Directed Acyclic Graph)
- Scene management
  - Object oriented approach to graphics
  - Defines and manages a set of objects in a 3D world
    - E.g. airports, offices, solar systems, etc...
  - Hierarchical structures
    - E.g. cars, humans, robotic arms...
- Optimizing graphics rendering
  - Culling, sorting, level of detail, ...

Implementation of OpenSceneGraph

- C++ API built on OpenGL
- Cross-platform
  - Supports all platforms having OpenGL and C++
    - E.g. Windows, MacOSX, BSD, Linux, Solaris, IRIX, Sony Playstation, etc…
    - (Not the XBox though.)

Layers



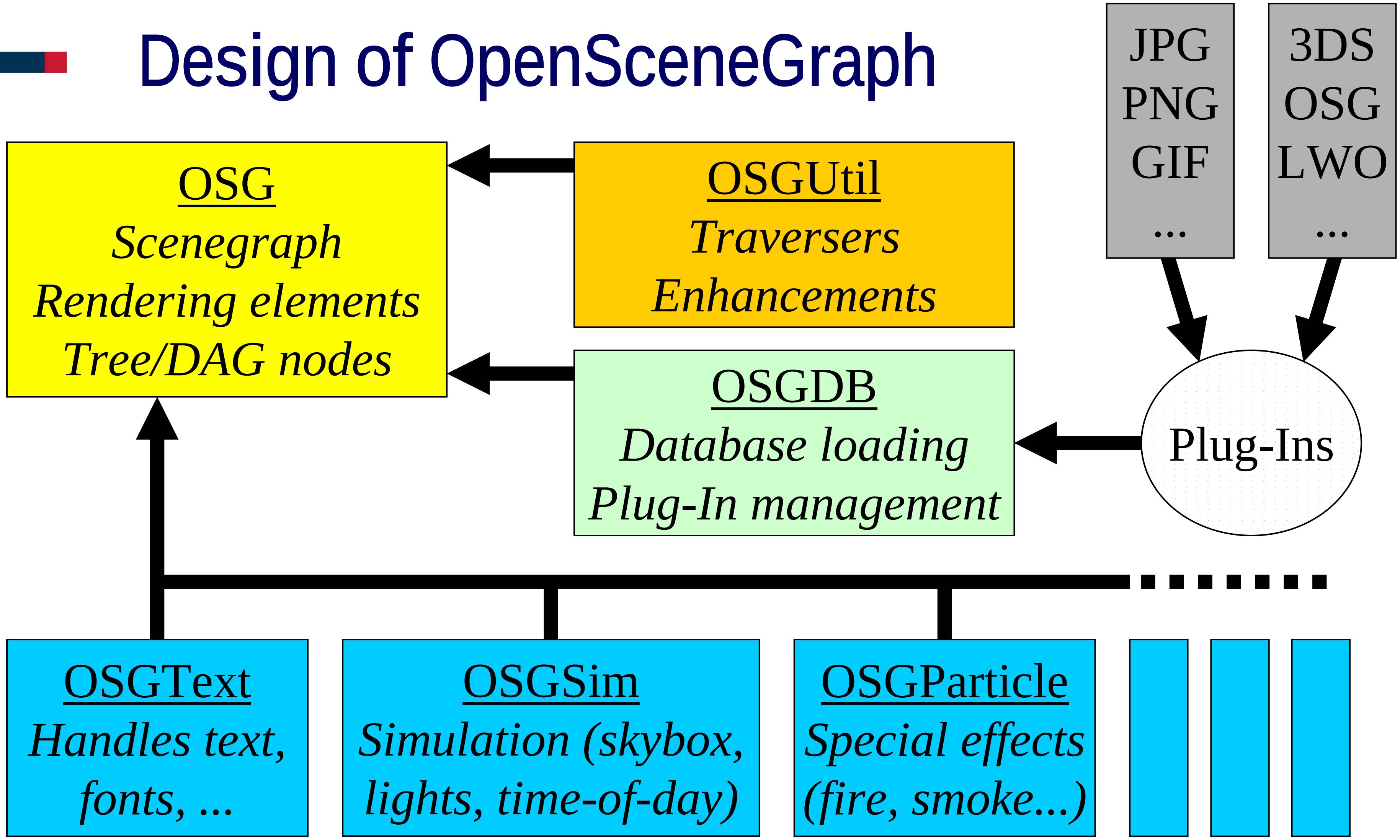
Applications

OpenSceneGraph

OpenGL

Graphics hardware

Design of OpenSceneGraph



OSG

Scenegraph

Rendering elements

Tree/DAG nodes

OSGUtil

Traversers

Enhancements

OSGDB

Database loading

Plug-In management

Plug-Ins

JPG
PNG
GIF
…

3DS
OSG
LWO
…

OSGText

Handles text, fonts, …

OSGSim

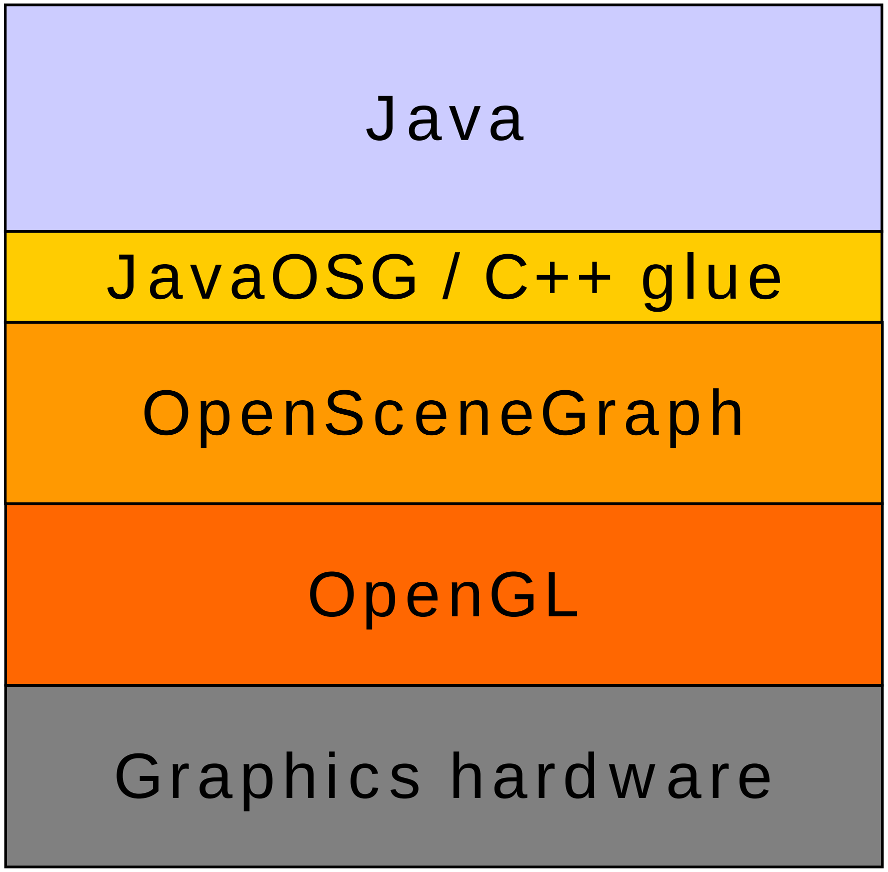
Simulation (skybox, lights, time-of-day)

OSGParticle

Special effects (fire, smoke…)

Accessing OSG from Java

JavaOSG



Java

JavaOSG / C++ glue

OpenSceneGraph

OpenGL

Graphics hardware

JavaOSG – Introduction

- Bindings for accessing OSG from Java
  - A set of native libraries (DLL:s in Windows)
  - A set of corresponding jar- and class-files for Java
  - Cross-platform
- C++ and Java code is similar, but not identical
  - Read the Javadocs (local copy on the course web)
  - Examples should help you get started
- JavaOSG webpage
  - http://www.noodleheaven.net/JavaOSG/javaosg.html

JavaOSG – Caveats

- JavaOSG is under development
  - ”Beta”, but stable enough
  - Not all of OSG is available
    - Some features/functionality not yet implemented
    - A few problems caused by language differences, e.g. multiple inheritance allowed in C++ but not in Java
  - API may change from one version to the next

    → Use JavaOSG version ***0.3.3*** for consistency

JavaOSG – Some known problems

- Java-specific documentation lacking
  - Still better than the alternatives =/
  - Examples, slides, javadocs, and OSG C++ code
- Compiling is a bit slow
  - Yes it is, the jar-files are quite large
- Callbacks
  - Some callbacks are troublesome and aren’t called
- Picking external models
  - Tricky, but there are some ways around this
- Learn the principles, not the tool

Installing

- OpenSceneGraph (use version 0.9.8)
  - http://www.openscenegraph.org/
  - Pre-compiled binaries with installer for Windows
- JavaOSG (use version 0.3.3)
  - http://www.noodleheaven.net/JavaOSG/javaosg.html
  - Pre-compiled libraries and jar-files for Windows
- Instructions
  - http://www.sm.luth.se/csee/courses/smm/009/osg_install.html

An overview of some OSG core classes

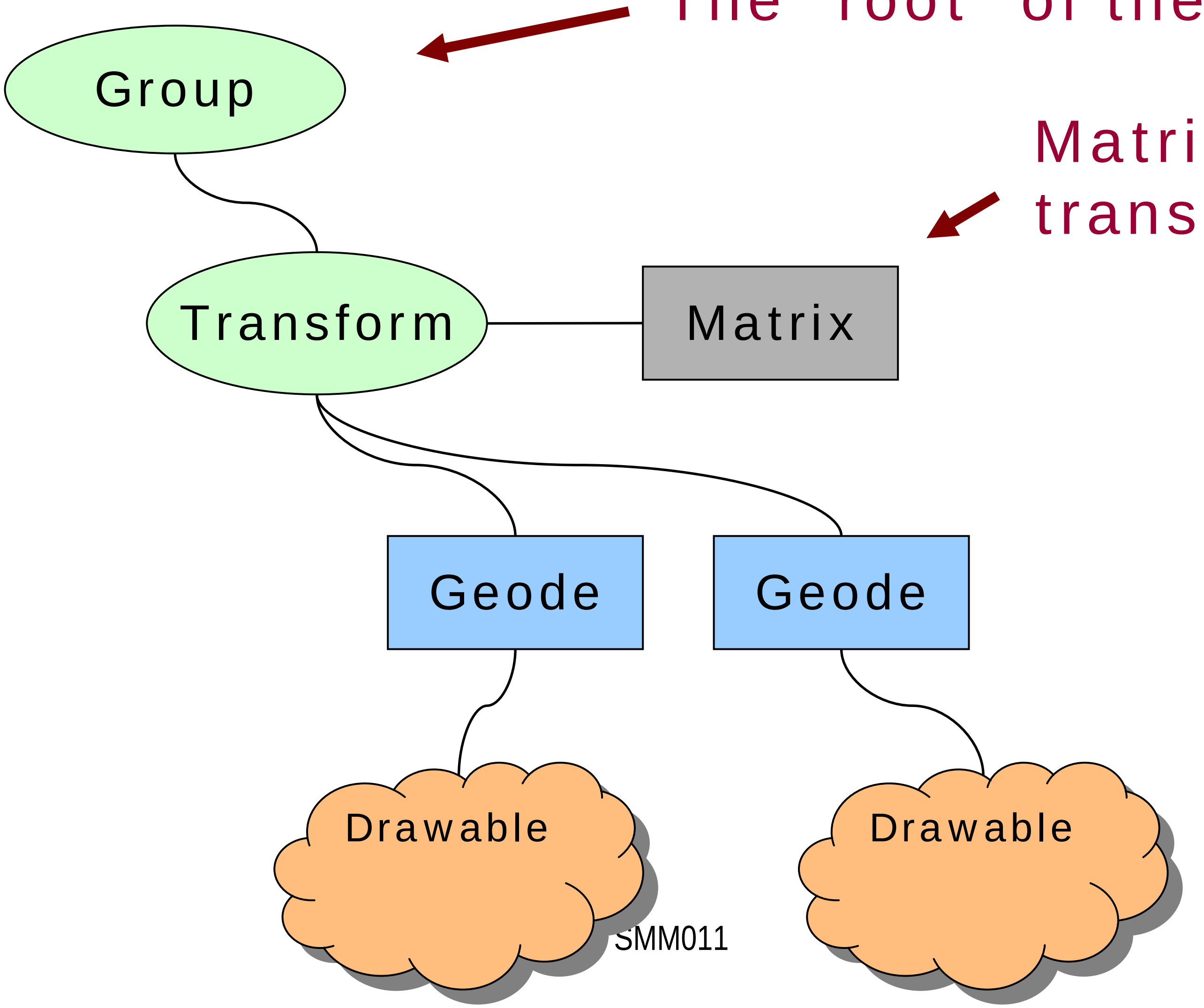
Nodes in the scenegraph tree (a subset)

- ”Node”, the base class
  - ”Group”, holds a set of child nodes
    - ”Transform”, transforms all children by a 4x4 matrix
    - ”Switch”, switches between children, e.g. traffic lights
    - ”LOD”, level of detail, switch based on distance to viewer
    - ”LightSource”, leaf node defining a light in the scene
  - ”Geode”, leaf node for grouping Drawables
    - ”Billboard”, orients Drawables to always face the viewer

Nodes in the scenegraph tree (a subset)

- ”Drawable”, abstract base class for drawable graphics
  - ”Geometry”, holds vertices, normals, faces, texture coords, ...
  - ”Text”, for drawing text
  - ”DrawPixels”, encapsulates drawing images using glDrawPixels
- ”StateSet”, encapsulates OpenGL states and attributes
- ”Texture”, encapsulates OpenGL texture functionality

Example

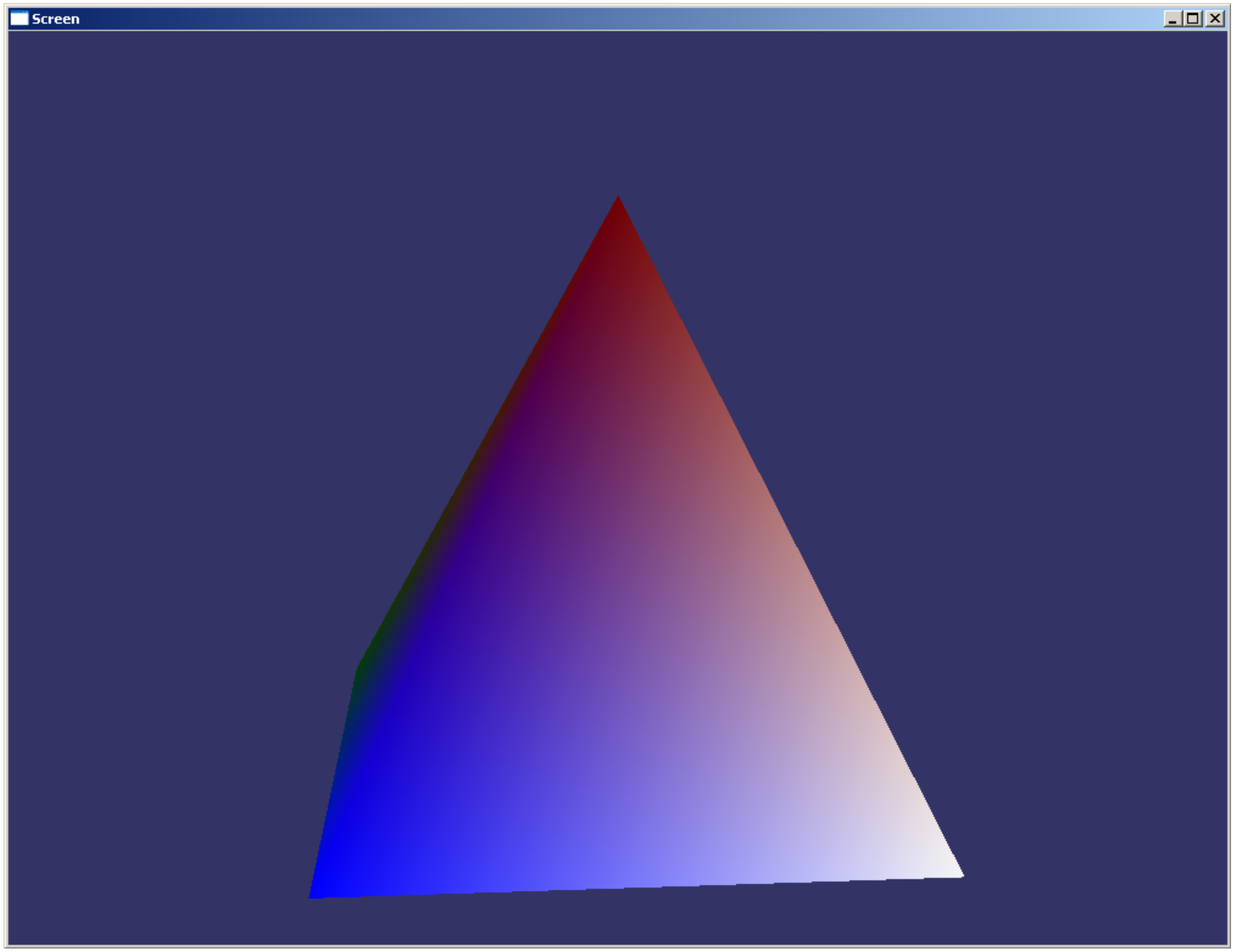


The ”root” of the scene

Matrix defines transformation

Creating OpenGL primitives

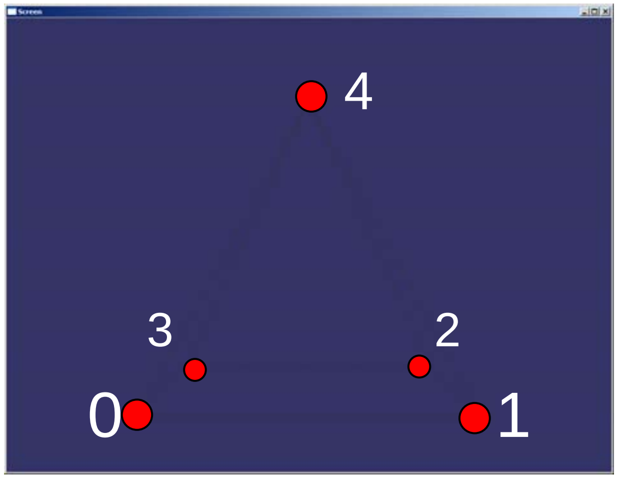
How do we create this colour pyramid?



Creating the viewer...

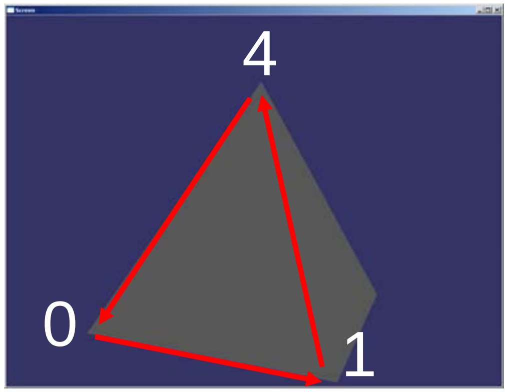
```
import openscenegraph.osg.*;
import openscenegraph.osgProducer.*;
import openscenegraph.osgDB.osgDBNamespace;
import noodle.noodleGlue.ShortPointer;
import noodle.noodleGlue.IntReference;

public class PrimitiveGL {

 public static void main(String[] args) {

   //
   // create viewer
   //
   Viewer viewer = new Viewer();

   viewer.setUpViewer(VIEWERViewerOptions.STANDARD_SETTINGS_Val);
```

Creating the geometry...



```
//
// create the model
//
Group root = new Group();

// create a geometry for holding OpenGL primitives...
Geometry pyramidGeometry = new Geometry();

// ..this will be a pyramid, so we specify the vertices
Vec3Array pyramidVertices = new Vec3Array();
pyramidVertices.push_back(new Vec3fReference(-5, -5,  0));  // left front  (0)
pyramidVertices.push_back(new Vec3fReference( 5, -5,  0));  // right front (1)
pyramidVertices.push_back(new Vec3fReference( 5,  5,  0));  // right back  (2)
pyramidVertices.push_back(new Vec3fReference(-5,  5,  0));  // left back   (3)
pyramidVertices.push_back(new Vec3fReference( 0,  0, 10));  // peak        (4)

// ..then add the vertices to the geometry
pyramidGeometry.setVertexArray(pyramidVertices);
```

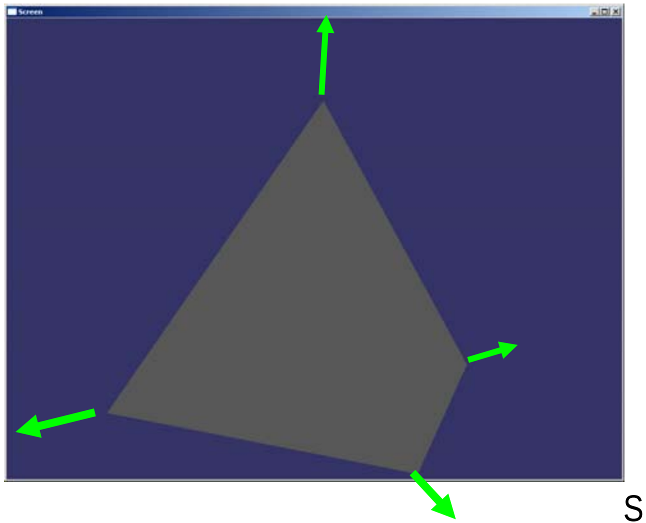
Specifying the faces...

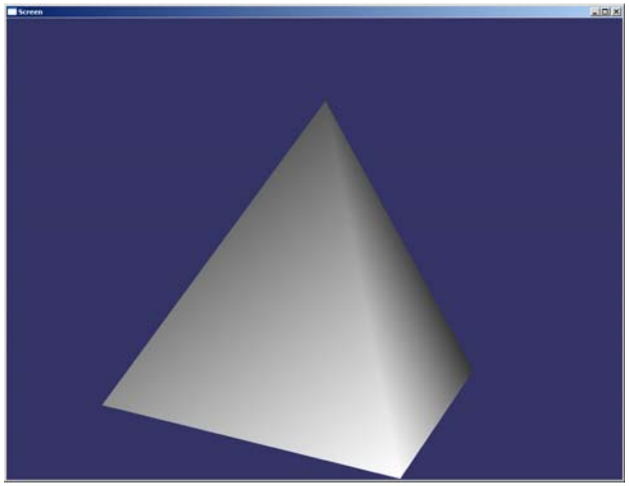


```
// next, we need to specify the 5 faces of the pyramid
// (4 triangular sides, 1 quadratic base), like this...

{ // base
    short indices[] = {3, 2, 1, 0};
    ShortPointer indices_ptr = new ShortPointer(indices);
    pyramidGeometry.addPrimitiveSet(
      new DrawElementsUShort(PRIMITIVESETMode.QUADS_Val,
                             indices.length,
                             indices_ptr));
}

{ // side 1
    short indices[] = {0,1,4};
    ShortPointer indices_ptr = new ShortPointer(indices);
    pyramidGeometry.addPrimitiveSet(
      new DrawElementsUShort(PRIMITIVESETMode.TRIANGLES_Val,
                             indices.length,
                             indices_ptr));
}

// side 2, 3 and 4 are designed in a similar fashion
```

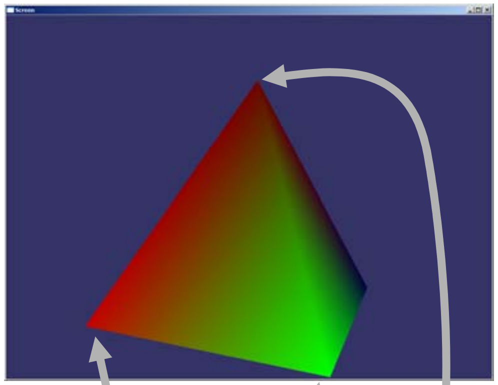
Defining the normals...

```
Vec3Array normals = new Vec3Array();
normals.push_back(new Vec3fReference(-1f,-1f, 0f));  // left front
normals.push_back(new Vec3fReference( 1f,-1f, 0f));  // right front
normals.push_back(new Vec3fReference( 1f, 1f, 0f));  // right back
normals.push_back(new Vec3fReference(-1f, 1f, 0f));  // left back
normals.push_back(new Vec3fReference( 0f, 0f, 1f));  // peak
pyramidGeometry.setNormalArray(normals);
pyramidGeometry.setNormalBinding(
     GEOMETRYAttributeBinding.BIND_PER_VERTEX);
```





Colouring the pyramid...



```
// create an array of colours
Vec4Array colors = new Vec4Array();
colors.push_back(new Vec4fReference(1f,0f,0f, 1f));  // red
colors.push_back(new Vec4fReference(0f,1f,0f, 1f));  // green
colors.push_back(new Vec4fReference(0f,0f,1f, 1f));  // blue
colors.push_back(new Vec4fReference(1f,1f,1f, 1f));  // white

// declare a variable matching vertex array elements to colour array elements
UIntArray colorIndexArray = new UIntArray();
IntReference intref;
intref = new IntReference(); intref.setValue(0); colorIndexArray.push_back(intref);
intref = new IntReference(); intref.setValue(1); colorIndexArray.push_back(intref);
intref = new IntReference(); intref.setValue(2); colorIndexArray.push_back(intref);
intref = new IntReference(); intref.setValue(3); colorIndexArray.push_back(intref);
intref = new IntReference(); intref.setValue(0); colorIndexArray.push_back(intref);

// associate the array of colors with the geometry
pyramidGeometry.setColorArray(colors);
pyramidGeometry.setColorIndices(colorIndexArray);
pyramidGeometry.setColorBinding(GEOMETRYAttributeBinding.BIND_PER_VERTEX);
```

Adding the geometry to the scene...

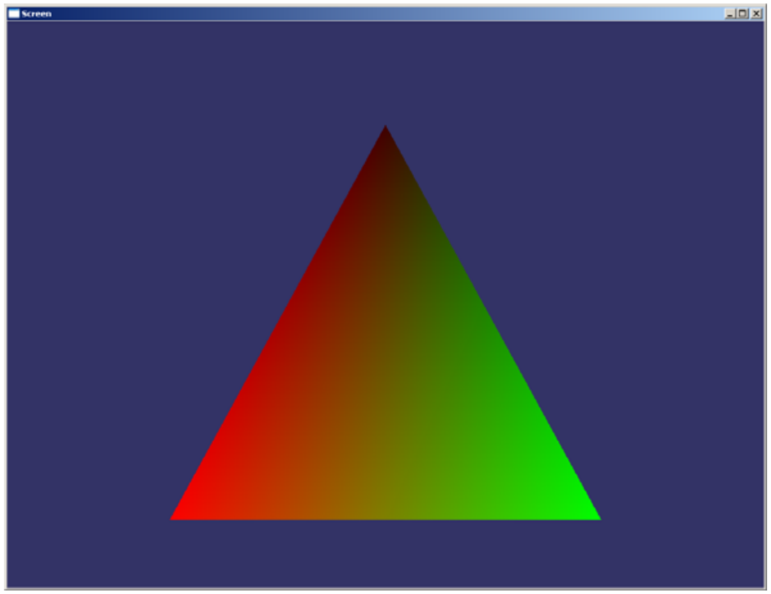
```
// create a geode (geometry node) holding the pyramid geometry
Geode pyramidGeode = new Geode();
pyramidGeode.addDrawable(pyramidGeometry);

// add geode to our model
root.addChild(pyramidGeode);

// add model to viewer
viewer.setSceneData(root);
```

Start running the viewer...

```
   //
   // create windows and start running thread
   //
   viewer.realize();

   //
   // event loop
   //
   while(!viewer.done()) {
       //
       // the drawing process
       //
       viewer.sync();
       viewer.update();
       viewer.frame();
   }
}
```

Done! Compile and run...

- `javac PrimitiveGL.java`
- `java PrimitiveGL`



A closer look at some of the code

The Viewer

```
viewer.setUpViewer(
    VIEWERViewerOptions.STANDARD_SETTINGS_Val);
```

- Configures the viewer with ”standard settings”
  - Mouse handler for rotating and moving around the scene
  - Keyboard handler with some useful key mappings
    - Esc - set the viewer.done() flag, e.g. to exit from the event loop
    - F - toggle full screen / window
    - L - toggle lighting
    - S - statistics about graphics performance
    - W - toggle solid / wireframe / vertices

The Viewer

- Other options

NO_EVENT_HANDLERS	**-**	no handlers installed
ESCAPE_SETS_DONE	**-**	exit by pressing Escape
HEAD_LIGHT_SOURCE	**-**	add a lightsource in front

- Settings can be combined (or'ed together), e.g.

```
viewer.setUpViewer(
    VIEWERViewerOptions.ESCAPE_SETS_DONE_Val |
    VIEWERViewerOptions.HEAD_LIGHT_SOURCE_Val
);
```

The Viewer’s event loop

```
while (!viewer.done()) {
   viewer.sync();
   viewer.update();
   viewer.frame();
}
```

`viewer.sync();`

- Waits for all draw and cull threads to complete

`viewer.update();`

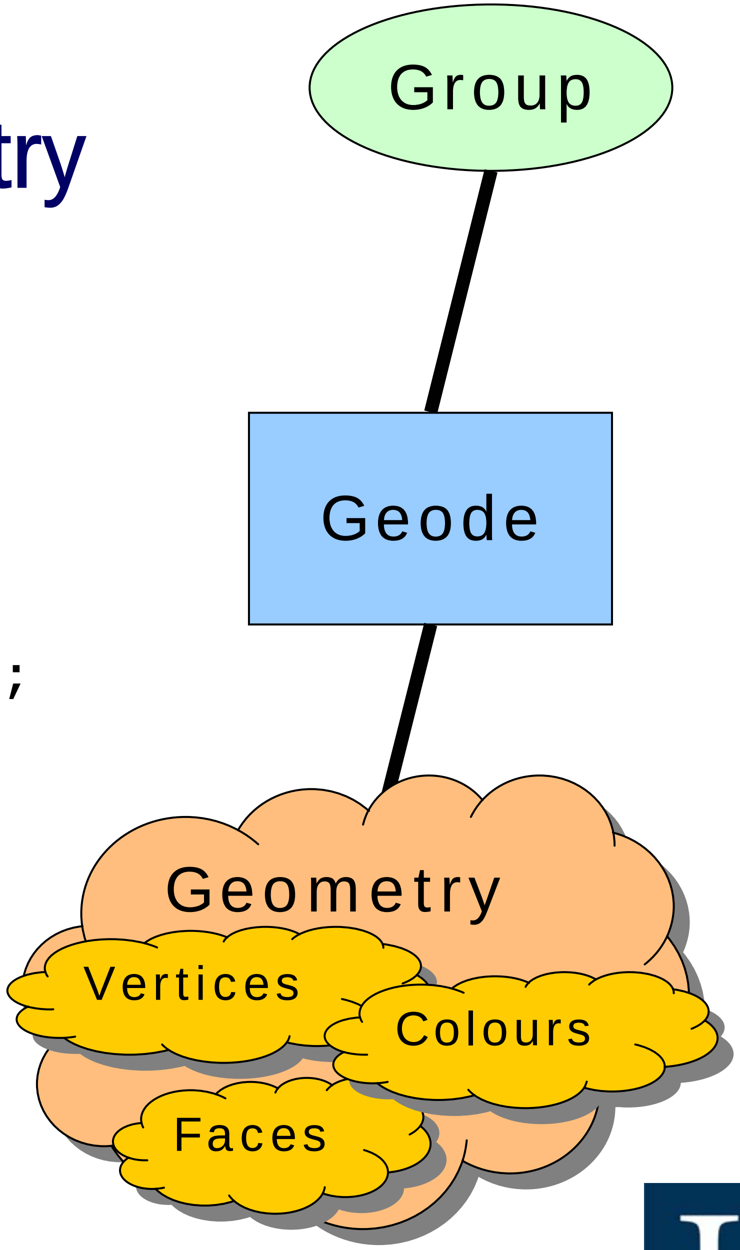
- Traverse the scene with an update visitor invoking node update and animation callbacks

`viewer.frame();`

- Start traversing the scene for drawing and culling

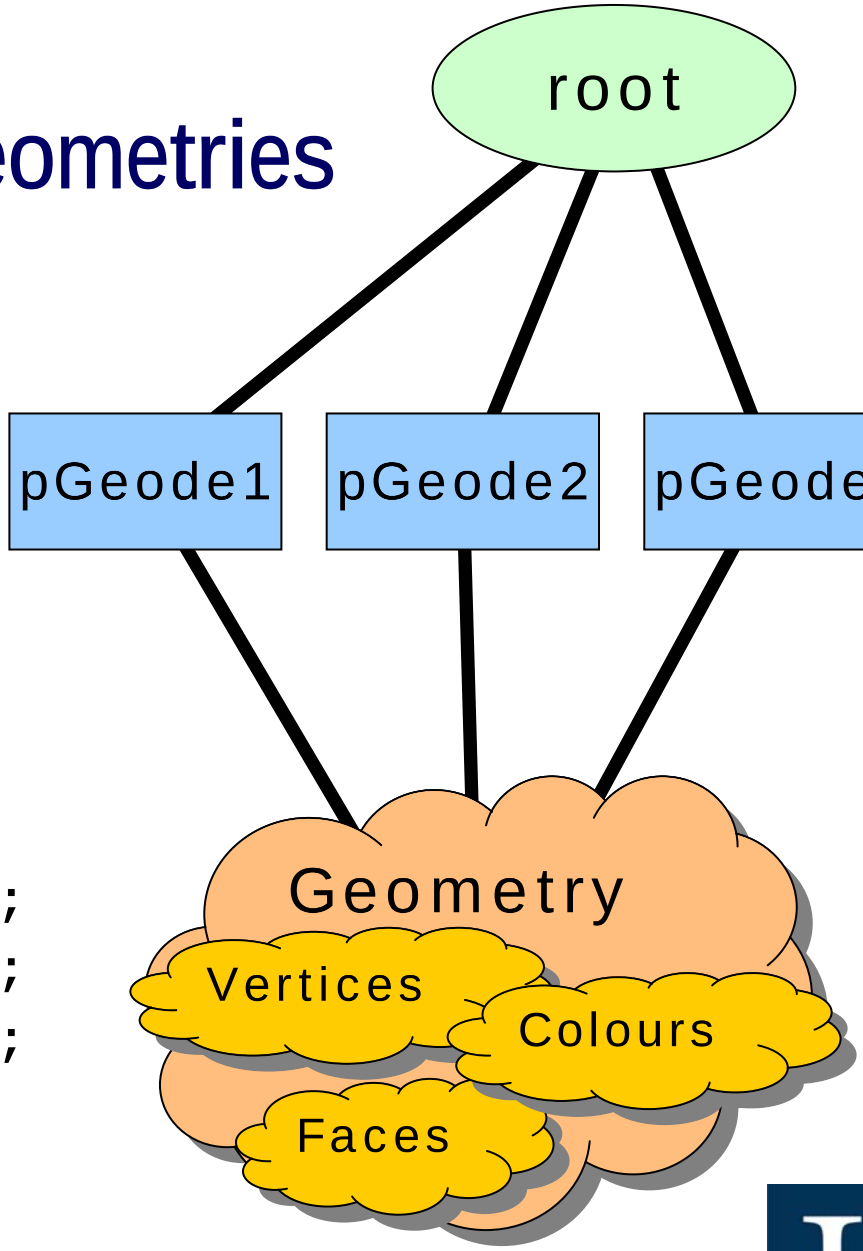
Group, Geode, Geometry

```
Group root = new Group();
...
Geode pGeode = new Geode();
root.addChild(pGeode);
...
Geometry pGeometry = new Geometry();
...
pGeode.addDrawable(pGeometry);
...
viewer.setSceneData(root);
```



Multiple Geodes/Geometries

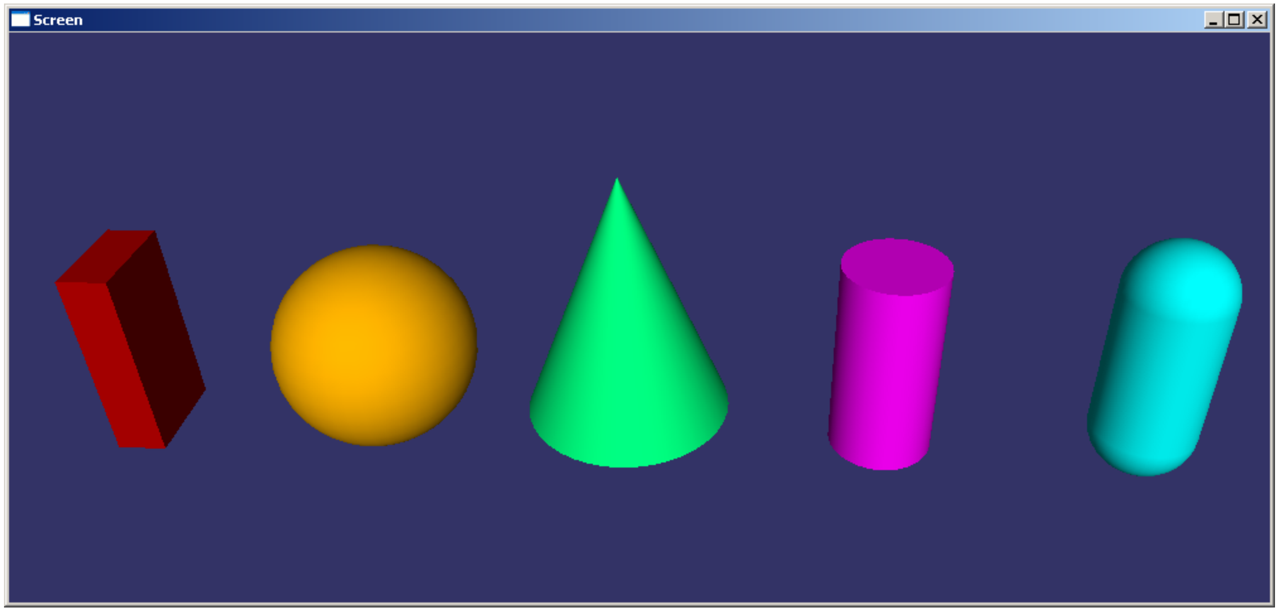
```
Group root = new Group();
...
root.addChild(pGeode1);
root.addChild(pGeode2);
root.addChild(pGeode3);

...
// sharing geometry
pGeode1.addDrawable(pGeometry);
pGeode2.addDrawable(pGeometry);
pGeode3.addDrawable(pGeometry);
```



Using the OSG built-in primitives

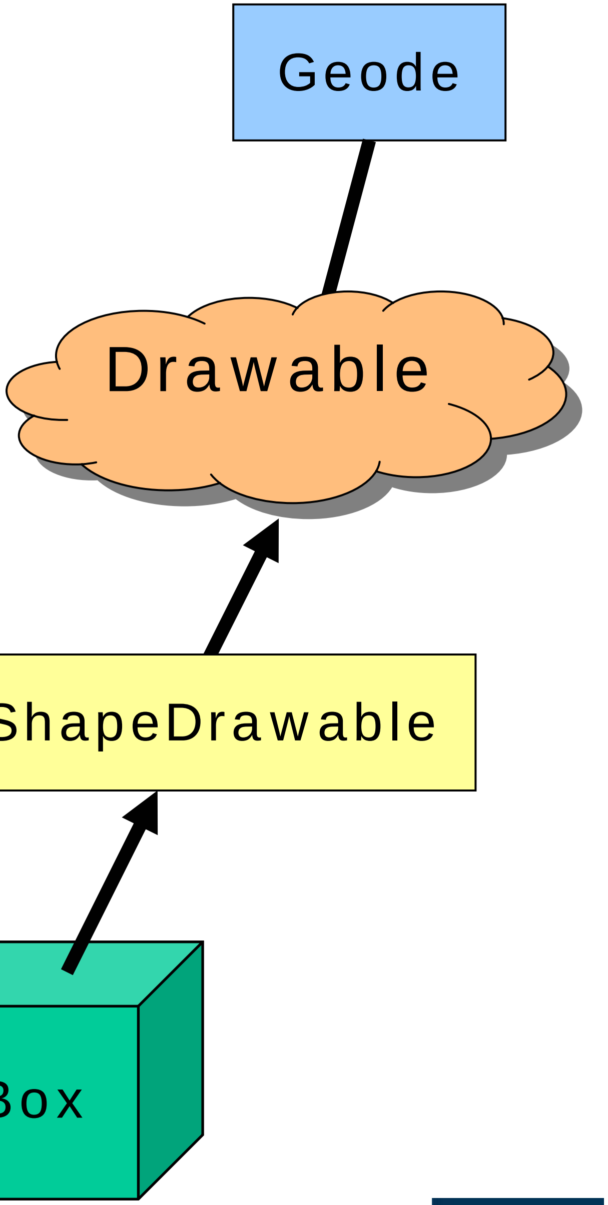
Primitive shapes

- OSG comes with a number of primitive shapes
  - Box, Sphere, Cone, Cylinder, Capsule
  - Plus some special shapes, e.g. InfinitePlane...



Using the primitive shapes

```
myGeode.addDrawable(
  new ShapeDrawable(
  new Shape(...)));
```



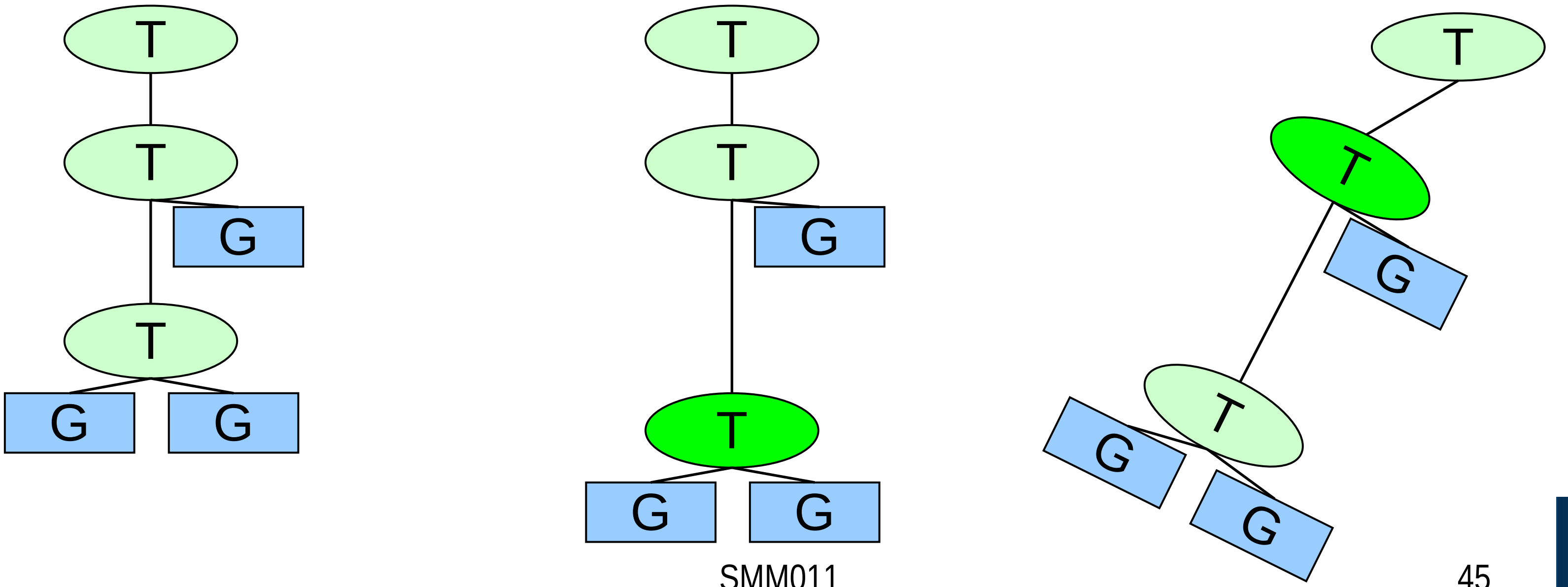
Example – Creating a cylinder

```
Geode cylGeode = new Geode();

ShapeDrawable cylShapeDrawable = new ShapeDrawable(
   new Cylinder(
        new Vec3fReference(0,0,0),    // center
        1,                            // radius
        4));                          // height

cylShapeDrawable.setColor(
   new Vec4fReference(1f, 0f, 1f, 1f) );     // magenta

cylGeode.addDrawable(cylShapeDrawable);
```

Transformations in OSG

Transformations

- Transformations apply to all child nodes in the tree
- Allows hierarchies, e.g. limbs on a human body



Transform nodes

- ”Transform”
  - ”MatrixTransform”
    - Has a 4x4 matrix (RefMatrixd) representing a transformation
  - ”PositionAttitudeTransform”
    - Sets transform via Vec3 position and Quat attitude
  - ”AutoTransform”
    - Automatically aligns children with screen coordinates

MatrixTransform

- Contains a 4x4 matrix
  - With JavaOSG, the matrix is a ”RefMatrixd”
  - getMatrix()
  - setMatrix()
- Matrix operations
  - makeIdentity()
  - makeTranslate(x, y, z)
  - makeRotate(angle, x, y, z,)
  - makeScale(x, y, z)
  - preMult() / postMult() for multiplying matrices

Example – MatrixTransform

```
MatrixTransform mt = new MatrixTransform();

// getting and translating the matrix
RefMatrixd matrix = mt.getMatrix();
matrix.makeTranslate(x,y,z);

// directly setting the matrix
mt.setMatrix(RefMatrixd.translate(x,y,z));

// adding child nodes (e.g. a geode)
mt.addChild(...);
```

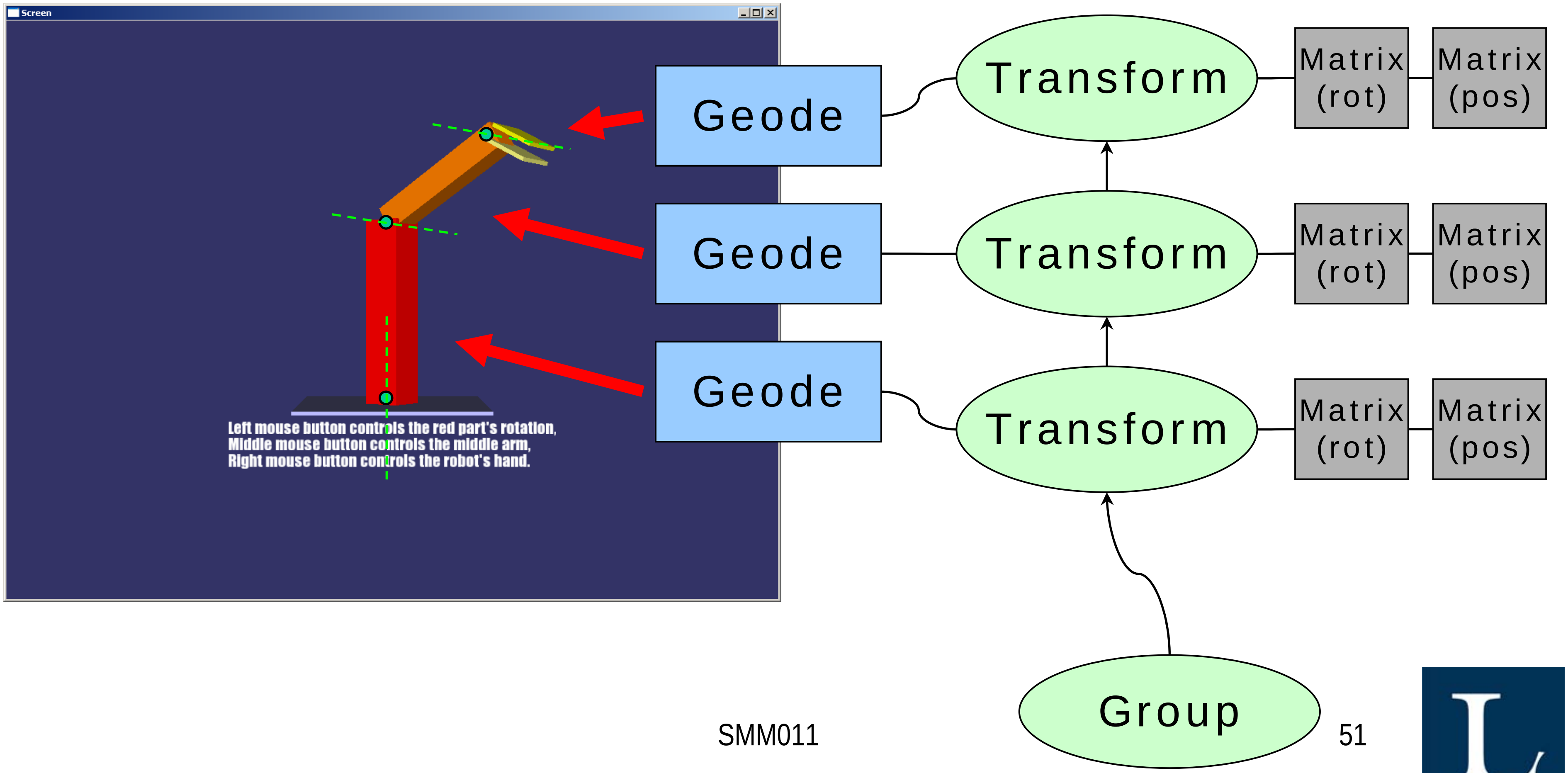
Example – Multiplying matrices

```
RefMatrixd matrix = new RefMatrixd();

// multiplying matrices
matrix.makeIdentity();
matrix.preMult(positionMatrix);
matrix.preMult(rotationMatrix);

// setting the transform’s matrix
mt.setMatrix(matrix)
```

Example – PositionAttitudeTransform

```
PositionAttitudeTransform pat =
  new PositionAttitudeTransform();

// positioning
Vec3dReference pos = new Vec3dReference(x,y,z);
pat.setPosition(pos);

// rotating
Quat rot = new Quat();
rot.setAxis(Vec3d.ZAxis);
rot.setAngle(rotation);
pat.setAttitude(rot);
```

Example – RobotArm.java



Left mouse button controls the red part's rotation,
Middle mouse button controls the middle arm,
Right mouse button controls the robot's hand.

Break...

Questions so far?

Agenda for 2nd half

- Configuring the viewer
- Event handlers
- Special nodes
- Picking objects

Configuring the viewer and camera

Viewer – using a matrix to set the view

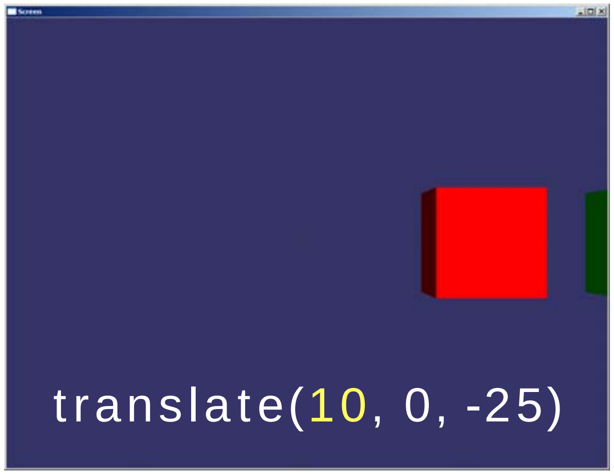
```
Matrix matrix;

// examples
matrix.makeRotate(angle, x,y,z);
matrix.makeTranslate(x,y,z);
matrix.preMult(...)

// set the view
viewer.setViewByMatrix(matrix);
```

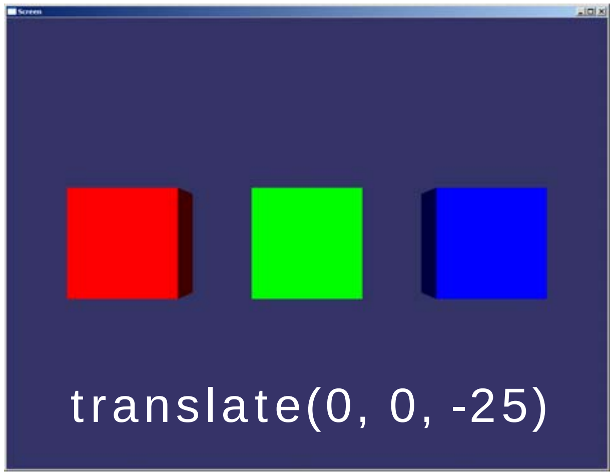
Viewer.setViewByMatrix()

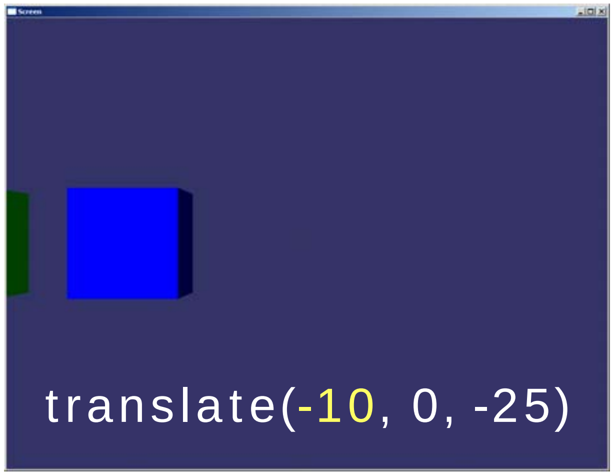
- Must be called between update() and frame()

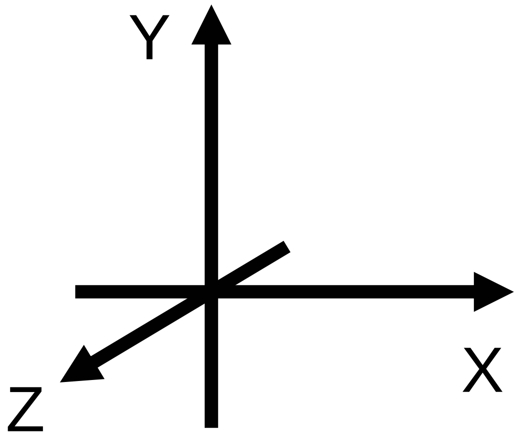
```
while (!viewer.done()) {
  viewer.sync();
  viewer.update();
  ...
  viewer.setViewByMatrix(matrix);
  ...
  viewer.frame();
}
```

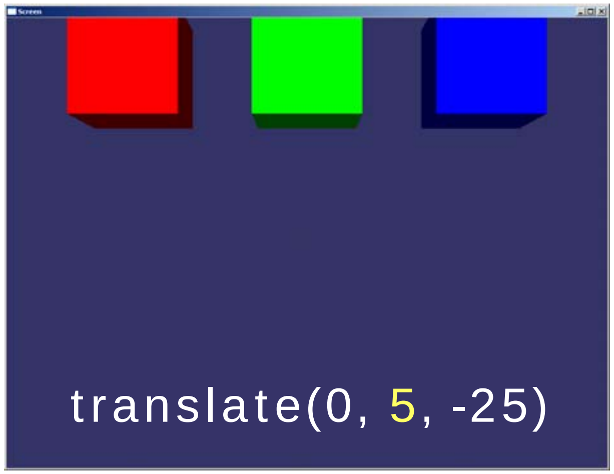
Viewer.setViewByMatrix() – translations


translate(10, 0, -25)


translate(0, 0, -25)

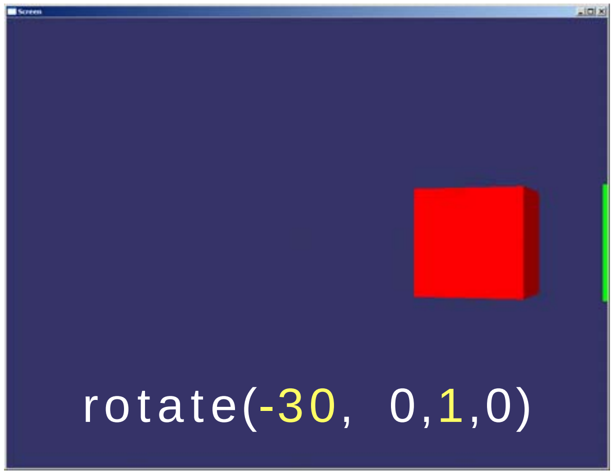

translate(-10, 0, -25)



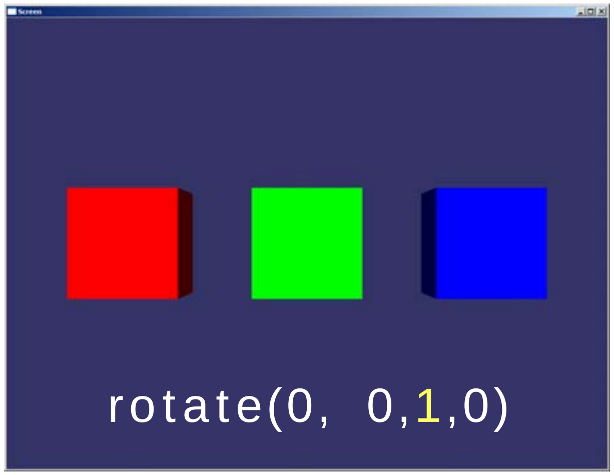

translate(0, 5, -25)

- Translations become ”inverted” compared to translating objects.
- You can think of it as ”translating the world”.

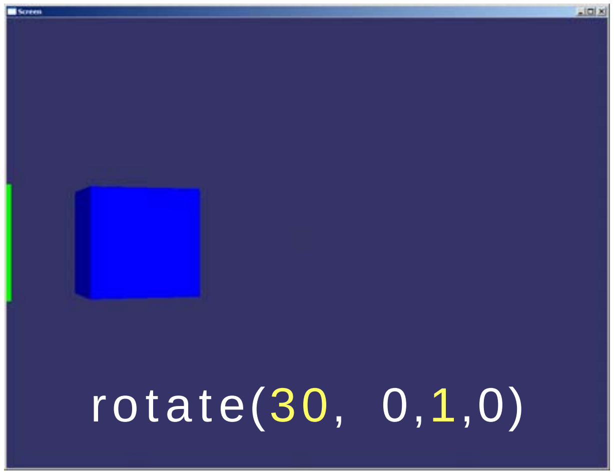
Viewer.setViewByMatrix() – rotations



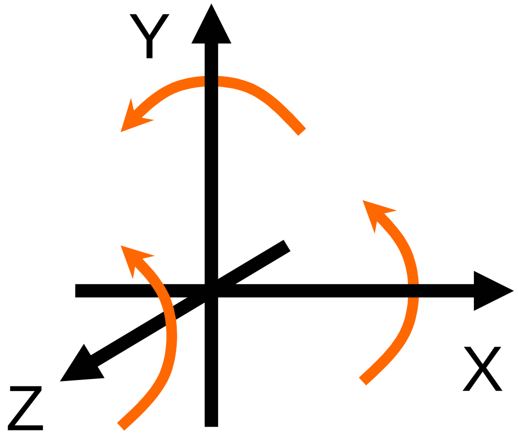
rotate(-30, 0,1,0)

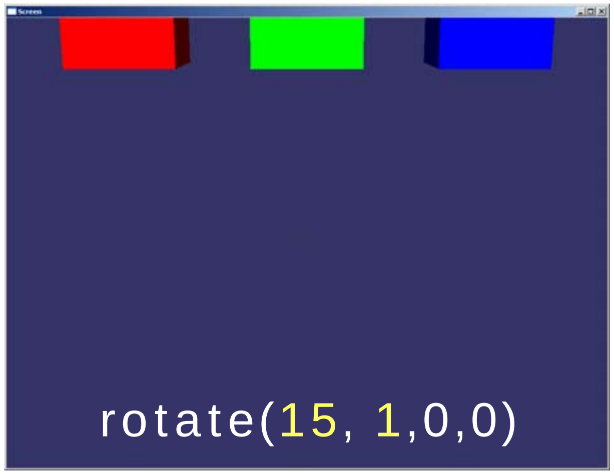


rotate(0, 0,1,0)



rotate(30, 0,1,0)

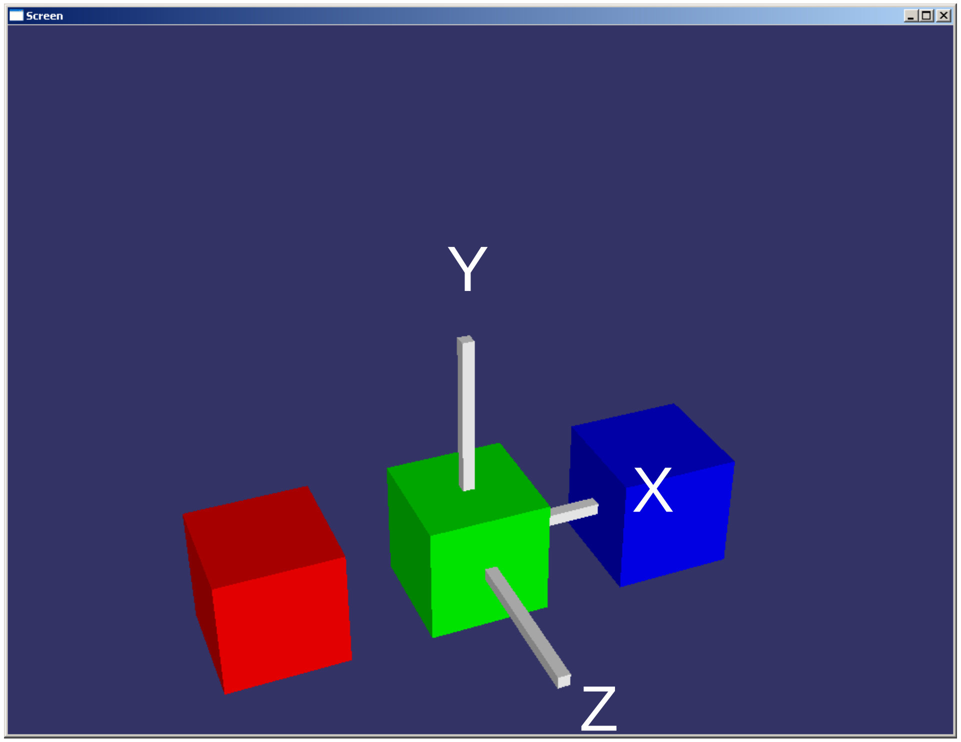




rotate(15, 1,0,0)

- Rotations become ”inverted” compared to rotating objects.
- You can think of it as ”rotating the world”.

Example – how do we obtain this view?

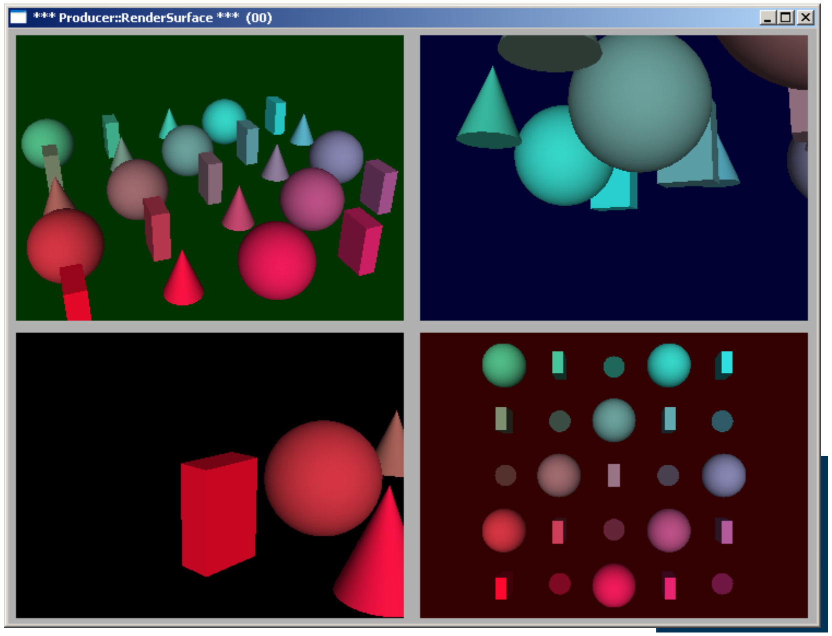


```
matrix.makeIdentity();

matrix.postMult(Matrix.translate(
        10,-20,-25));

matrix.postMult(Matrix.rotate(
        Math.PI/8, 0,1,0));

matrix.postMult(Matrix.rotate(
        Math.PI/6, 1,0,0));

viewer.setViewByMatrix(matrix);
```

Manip.
Viewer
Simple

Viewer vs. Camera

- Different ”interfaces” for roughly the same functionality
  - setViewByMatrix(), view determined by a 4x4 matrix
  - setViewByLookAt(), similar to gluLookAt() in OpenGL
  - setLensPerspective(), adjusts the lens
- Viewer can be constructed from a CameraConfig
  - CameraConfig can have one or more Cameras
    - Allows multiple views →



The Camera

- Contains a Lens
  - Lens gives control over the OpenGL PROJECTION matrix
  - (The OpenGL MODELVIEW matrix is controlled through the camera's position and attitude)

Relevant examples on the course web

- ManipulateViewerSimple.java
  - Static positioning of viewer via matrix operations
- ManipulateViewer.java
  - Viewer moves around in a circle
- MoveCamera.java
  - User moves camera
- MultiView.java
  - Multiple cameras viewing the same scene

Manip. Viewer

Move Camera

Multi View

Event handlers for user input

Creating the handler

- Extend the GUIEventHandler class
- Implement your own handle() function
  - Invoked upon keyboard and mouse events

```
class MyHandler extends GUIEventHandler {
   public boolean handle(GUIEventAdapter event,
                         GUIActionAdapter action) {
       ...
   }
}
```

Creating the handler

```
public boolean handle(GUIEventAdapter event,
                      GUIActionAdapter action) {
```

- event
  - Holds mouse button status, coordinates, key pressed, ...
- action
  - The Viewer implements the GUIActionAdapter interface
  - Access the Viewer from where the event originated
  - Can call useful functions in response to an event, e.g.
    - requestWarpPointer(x,y)
    - getSpeed()

Example – Handling events

```
public boolean handle(...) {
   if (event.getEventType == GUIEVENTADAPTEREventType.KEYDOWN) {
        switch(event.getKey()) {
                ...
        }
   }

   if (event.getEventType == GUIEVENTADAPTEREventType.DRAG) {
        float x = event.getX();
        float y = event.getY();
        ...
   }
   return true;
}
```

TRUE means no other handlers will be invoked, FALSE means they will be

User Input

Special nodes

Special nodes

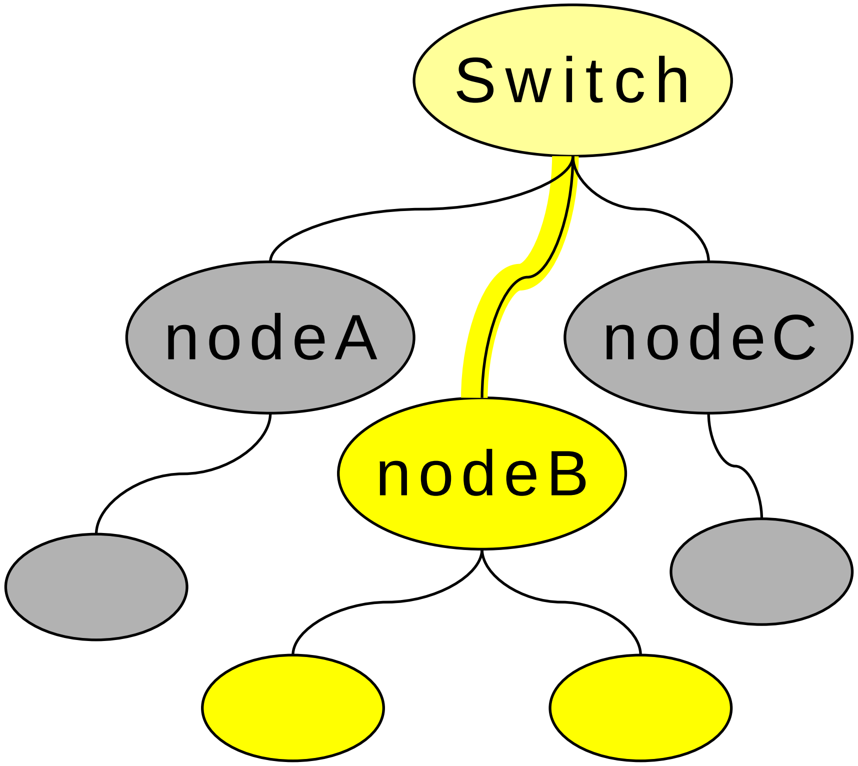
- Nodes for switching states for an object
  - The ”Switch” node
- Nodes for optimizing rendering performance
  - The ”LOD” node
  - The ”Billboard” node
- Nodes for presenting text
  - The ”Text” node

The ”Switch” node

- How to represent a traffic light
  - Can be either red or green
  - Add the different appearances to the Switch node
    - switchNode.addChild(trafficRed);
    - switchNode.addChild(trafficGreen);
    - Then add some logic to control the switch node
- What about a box that can be opened and closed
  - Has two different states, either opened or closed
  - Can be solved by rotating its lid, maybe that’s better?

The ”Switch” node – an example

```
Switch s = new Switch();
...
s.insertChild(0, nodeA);
s.insertChild(1, nodeB);
s.insertChild(2, nodeC);
...
```



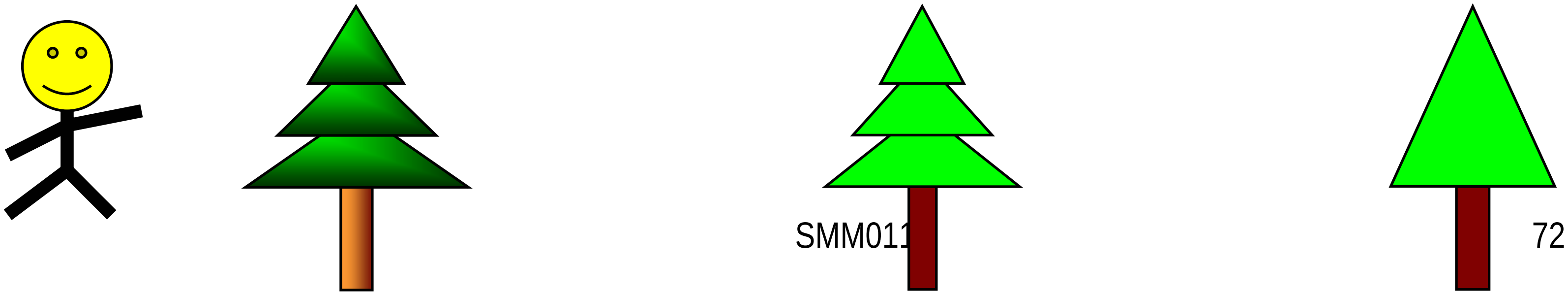
```
// in e.g. event handler
s.setSingleChildOn(1);
```

Level of detail – the problem

- Example: how to represent and draw a tree
  - A tree has many details (leaves, branches, textures...)
    - Requires probably a few millions of polygons
    - Easy if we only want to see one instance close-up
  - What happens when we want a whole forest?
    - Draw all trees? Billions of polygons? Not feasible!
    - Reduce polygon count? No, we still want details!

Level of detail – one solution

- Human vision and screen resolution are limited
  - Can you really make out a leaf on a tree ~1 km away?
  - Will a tree that far away occupy more than a few pixels?
- We can reduce details for objects far away
  - For close-ups, use a highly detailed object
  - For medium distances, some details are unimportant
  - For very long distances, we can use a very simple object



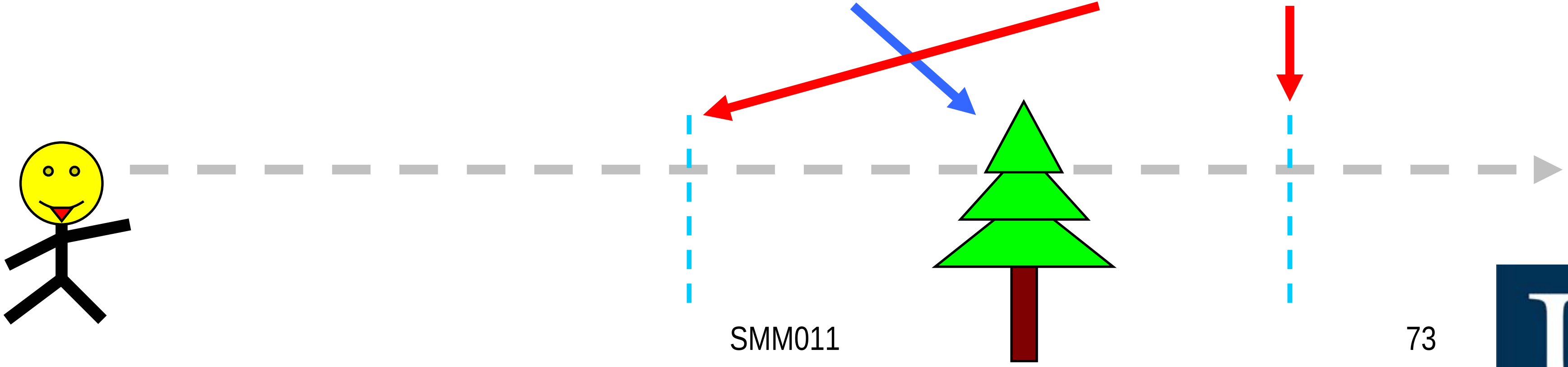
Level of detail – LOD in OSG

- ”Level of Detail” (LOD)
  - Like a Switch node but switches based on distance to viewer
- Works like a regular group node

```
lod.addChild(detailedNode);
```

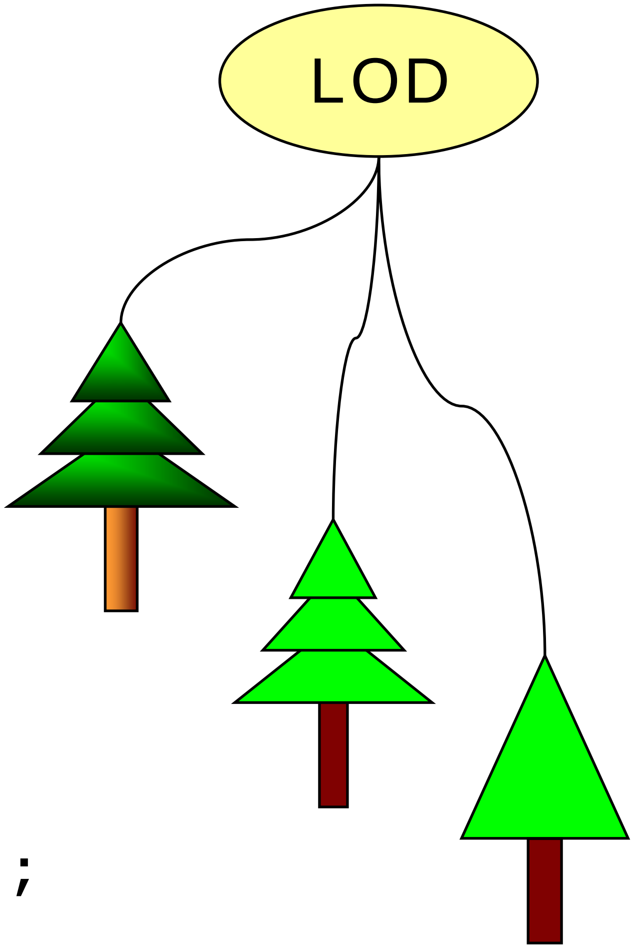
- Set visible range for each child (unique or overlapping)

```
lod.setRange(childNumber, near, far);
```



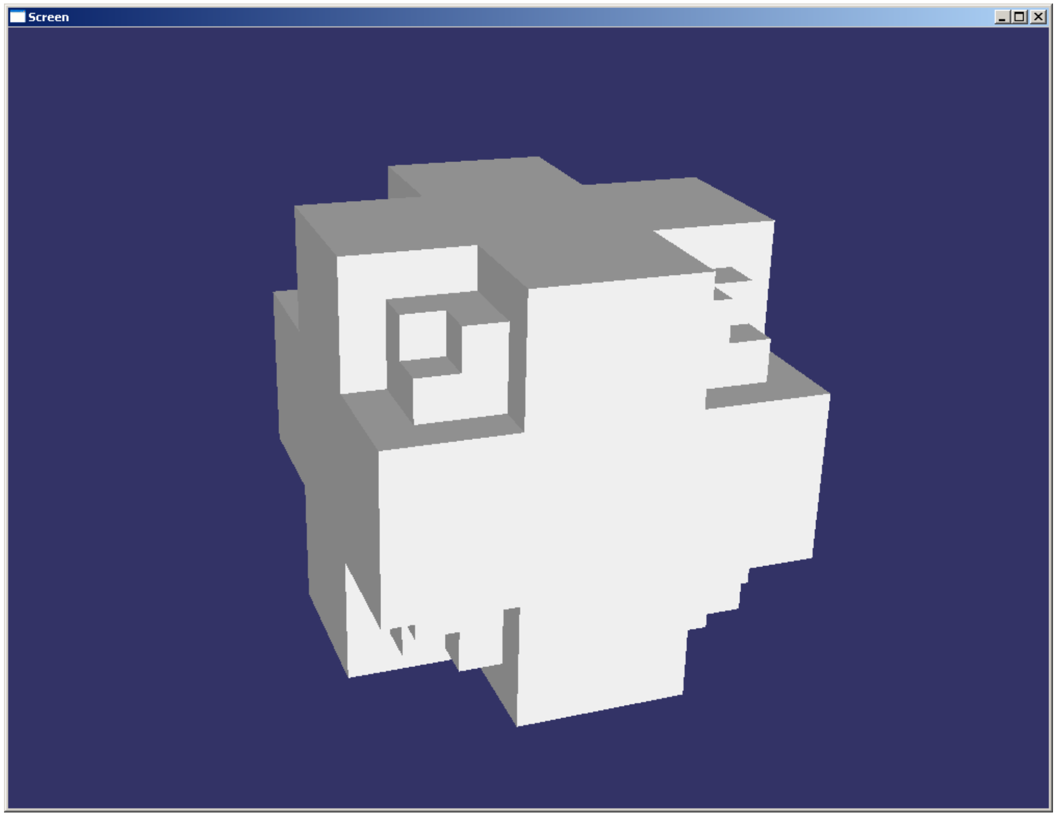
Level of detail – example

```
LOD lod = new LOD();
...

lod.addChild(detailedNode);
lod.setRange(0, 0, 10);

lod.addChild(notSoDetailedNode);
lod.setRange(1, 10, 100);

lod.addChild(noDetailsAtAllNode);
lod.setRange(2, 100, 25000);
```



Level of detail – example on the web

- Look at ”LODingNode.java”



LODing Node

Billboards

- The idea
  - – Instead of modeling a detailed object, use an image map
  - – Make the image always face the user
    - Images come in 2D, but we want a ”3D” object
    - Suitable for representing e.g. trees in a forest
- Benefits
  - – Cheaper to render (image map vs. numerous polygons)
  - – Natural things are difficult because they’re, well, natural

Billboard Node

Billboards – some issues to consider

- The object will look somewhat fake, it’s not really 3D
- Works best with viewer + objects on the same level
  - E.g. walking on the ground in a billboard forest looks good, but flying over the forest – looking down – causes problems
- Don’t reuse the same image again and again and again
  - Variation is the key among numerous billboards
- Maybe suitable as a LOD child node when far away

Billboard Tree

Billboards – the node in OSG

- Billboard is a subclass of Geode
  - addDrawable(...)
- Can change
  - The axis (or point) of rotation
  - The direction in which the billboard will face
    - setNormal(...)

Billboards – adding textures

- Textures are used to map an image to the billboard
- Typically, a 2D plane is used as a drawable
  - E.g. a GL primitive, rectangle or triangle
- Texture mapping is straightforward in this case
  - (Textures, both in OSG and in general, will be covered in future lectures, so this is just a primer to get you started)

Billboards – adding textures

Any file format supported by the plugins

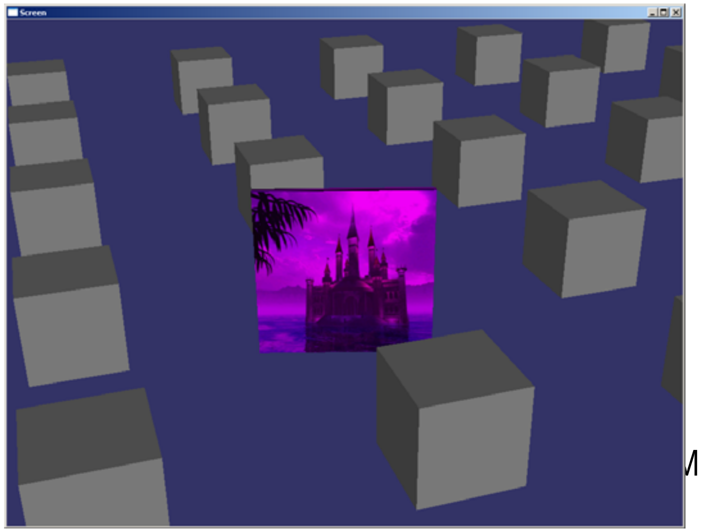
```
Image image =
  osgDBNamespace.readImageFile(”tree.jpg”);

Texture2D texture = new Texture2D();
texture.setImage(image);

StateSet stateset = new StateSet();
stateset.setTextureAttributeAndModes(0,
     texture, STATEATTRIBUTEValues.ON_Val);

bb.setStateSet(stateset);
```

Turns texture 0 ON and associates it with the billboard.

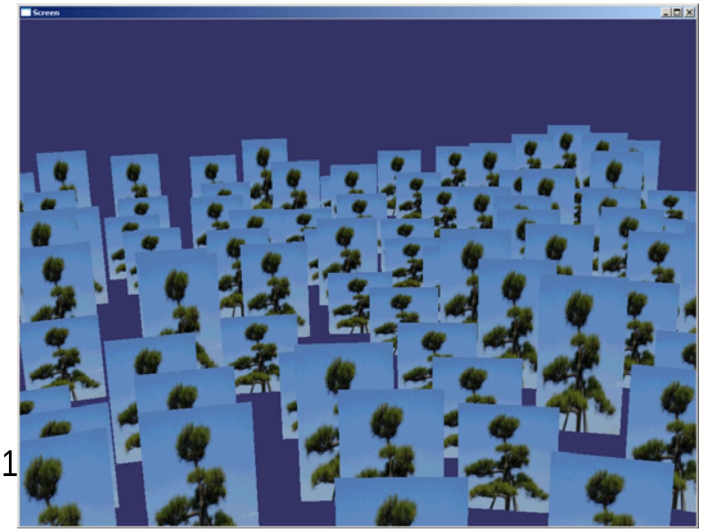
Billboards – an example

```
Billboard bb = new Billboard();
...
bb.addDrawable(...);
```

BillboardingNode.java



BillboardingTree



Text objects

- Labeling objects, augmenting them with information
- Text nodes behave just like any other node
  - Translates, rotates, scales, become occluded, etc...
  - E.g. add text next to a geode, and it will stay with it
- Can auto-align to always face the screen
  - Makes it easier to read
- Fonts
  - Can use standard system fonts (Arial, Courier, Times, ...)

Text objects – an example

```
import openscenegraph.osgText.*;
...

// create text object
Text label = new Text();

// set font size and colour
label.setCharacterSize(0.4f);
label.setFont("/fonts/arial.ttf");
label.setColor(new Vec4fReference(1f,1f,0f,1f));

// the text to display (changeable during run-time)
label.setText("Sphere");
```

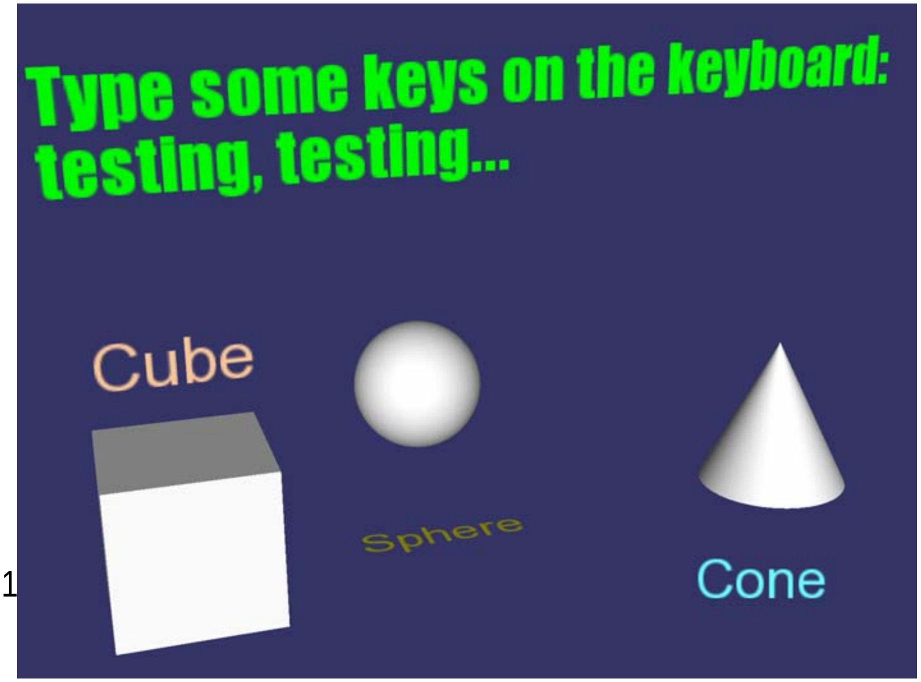
Set to SCREEN to face the viewer

Text objects – an example

```
label.setAxisAlignment(TEXTAxisAlignment.XY_PLANE);
label.setAlignment(TEXTAlignmentType.CENTER_TOP);
label.setDrawMode(TEXTDrawModeMask.TEXT_Val);
label.setPosition(new Vec3fReference(0,-0.5f,-1.0f));
```

- Text is a subclass of Drawable...

```
Geode geode = new Geode();
geode.addDrawable(label);
scene.addChild(geode);
```



Text Demo

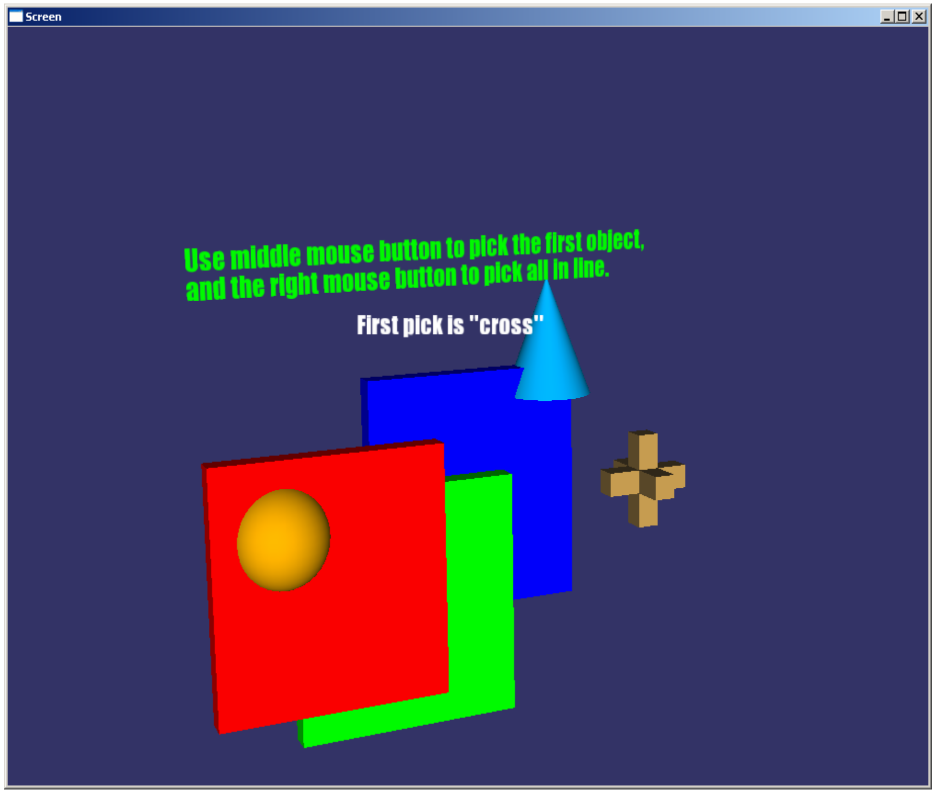
Picking

About picking

- The principle
  1. Project a ray from the mouse pointer into the screen
  2. Detect the surfaces which the ray intersects with
  3. Get hold of the corresponding nodes

Picking in OpenSceneGraph

- Example on the web
  - Picking.java
- Changes colour on the object(s) in the projected ray's path.
- The code?



Use middle mouse button to pick the first object, and the right mouse button to pick all in line.

First pick is "cross"

Picking in OpenSceneGraph – code

- Put this code in your event handler’s ”handle()” function

```
  INTERSECTVISITORHitList hitlist =
     new INTERSECTVISITORHitList();

  if (viewer.computeIntersections(
          event.getXnormalized(),
          event.getYnormalized(),
          0, hitlist)) {
          ...
}
```

Fills the hitlist with all intersections for the ray at (X,Y)

Picking in OpenSceneGraph – code

- ...then, go through the hitlist

Depth sorted – the first hit is also the foremost.

```
if ( ! hitlist.empty()) {
  for (int i=0; i<hitlist.size(); i++) {
     Hit hit = hitlist.at(i);

     Geode pg = (Geode)hit.getGeode();

     ...
  }
}
```

This function is not in the current javadocs, but you will need it!

Picking in OpenSceneGraph – code

- Can make it even more object oriented...
  - Create e.g. a ”PickableGeode”

```
private class PickableGeode extends Geode {
  public void pick() { /* do something */ }
}
```

E.g. setColour(rgb)

Picking in OpenSceneGraph – code

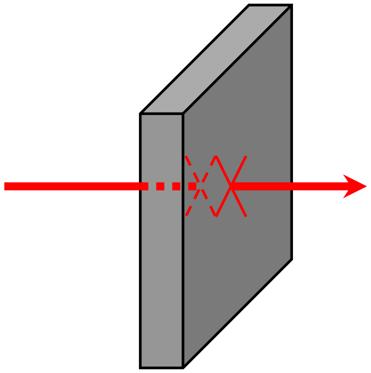
- ...then, in the event handler...

Remember to check this before casting if you mix different geodes

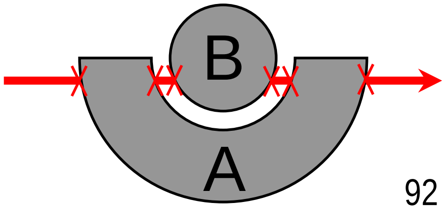
```
if (hit.getGeode() instanceof PickableGeode) {
  PickableGeode pg =
     (PickableGeode)hit.getGeode();

  pg.pick();
}
```

Perform the action on the object

Picking in OpenSceneGraph – hints



- Hitlist contains one hit for **each** intersection
  - Possible to get multiple hits for the same Geode
  - For example, a box will give 2 hits
    - 1 Hit for the front face + 1 Hit for the back face
  - Be careful with code that toggles some state
    - 1st hit toggles on, 2nd hit toggles it back off!
- Depth sorted hits
  - The geodes are not necessarily sorted
    - Consider e.g. A_1, A_2, B_3, B_4, A_5, A_6...

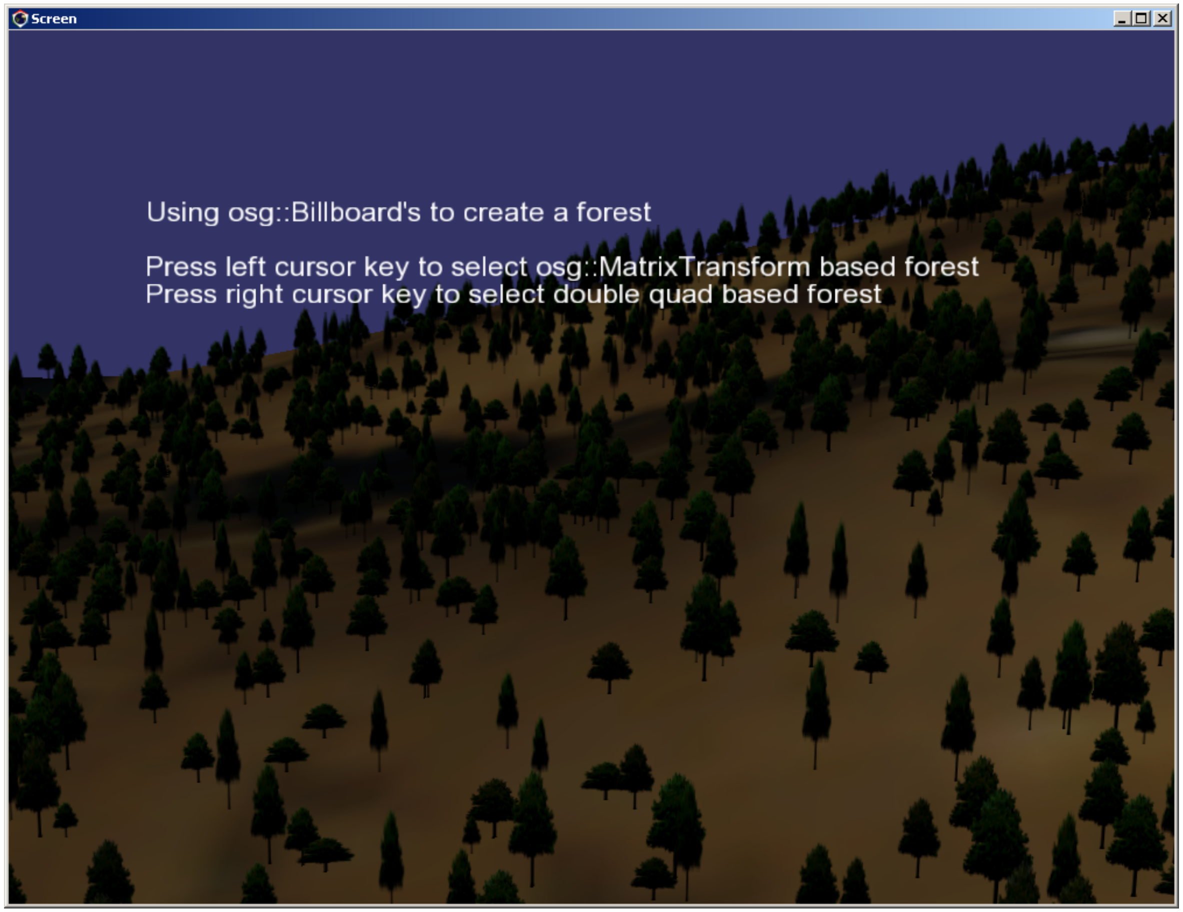


Beyond picking... Intersections!

- You can execute arbitrary intersection tests in a scene
  - Picking is just a specific case of such a test
- Workflow for a test
  - Create the ray as a LineSegment with start and stop coordinates
  - Create an IntersectVisitor
  - Add the LineSegment to the IntersectVisitor
  - Start a traversal with the IntersectVisitor at a start node (e.g. the root)
    - We use a hitlist and proceed just like when picking
  - Retrieve the resulting hits from the test
    - Get hold of the nodes (and coordinates) for the hit

Intersections – example

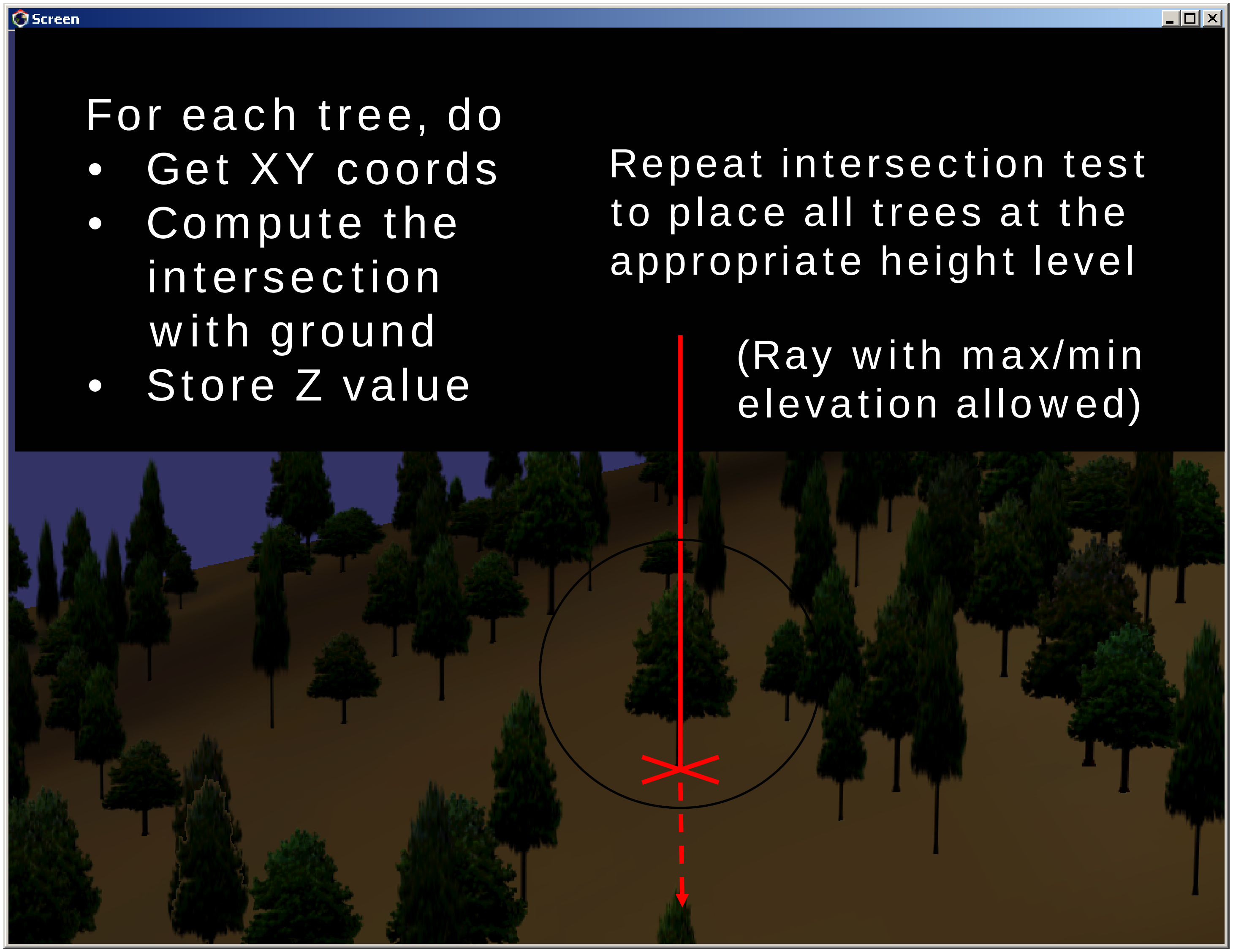
- How to create a forest covering rolling hills and stones?



Using osg::Billboard's to create a forest

Press left cursor key to select osg::MatrixTransform based forest
Press right cursor key to select double quad based forest

Intersections – example



For each tree, do

- Get XY coords
- Compute the intersection with ground
- Store Z value

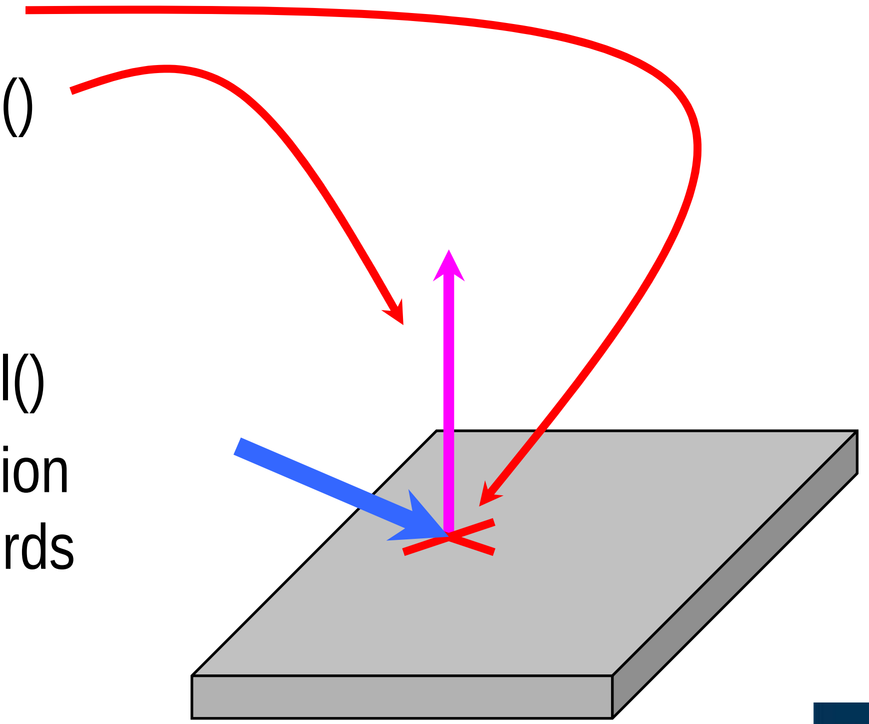
Repeat intersection test to place all trees at the appropriate height level

(Ray with max/min elevation allowed)

Intersections – more about the Hit

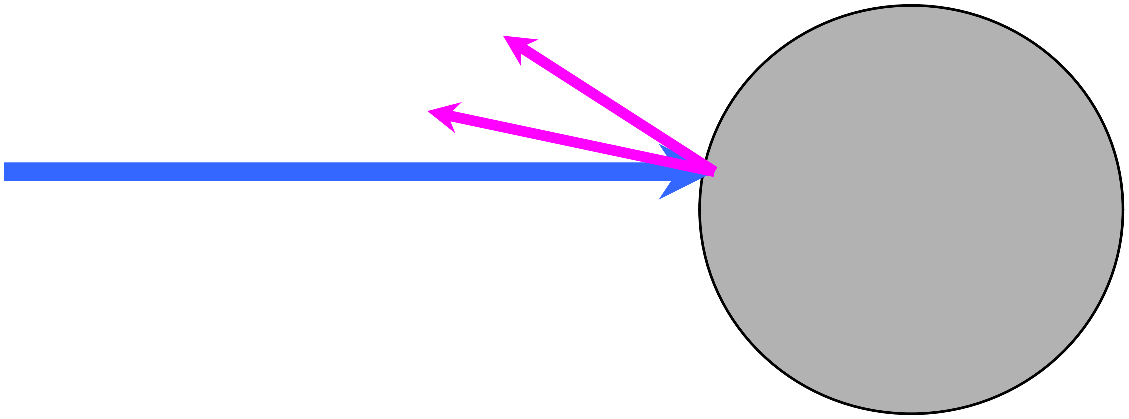
- Retrieving the coordinates for an intersection
  - getLocalIntersectPoint()
  - getLocalIntersectNormal()

  - getWorldIntersectPoint()
  - getWorldIntersectNormal()
    - Returns the intersection in world-oriented coords



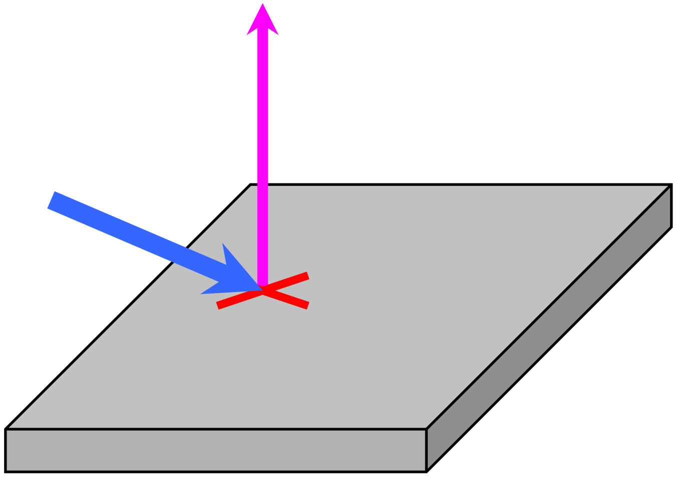
Intersections – more about the Hit

- Most intersections take place with tesselated objects
  - But this can be overridden by the intersected objects
  - E.g. a sphere can override its tesselated representation with a true sphere intersection calculation, giving more accuracy



Intersections – more about the Hit

- Some examples on what to use all the information in a Hit for...
  - Bouncing balls against objects
  - Implementing physics engines
    - Grounding avatars
    - Sliding down slopes
  - Computing damage based on the angle of impact
    - Punching objects
    - Holes vs ricochets



Intersections – end notes

- Intersection tests are very useful, not just in picking

Exercise assignment

Getting started...

- Exercise assignment to get you started with OSG
  - Download ”GetStarted.java” from the course web
  - Follow the instructions and hand in your solution
  - Complete the assignment *individually*
  - Deadline next friday

Summary

- Introduction to OpenSceneGraph
- Accessing OSG from Java
  - Done through the Java bindings
- Overview of core classes
  - Nodes, Groups, Geodes, Drawables, Transform nodes
- Creating OpenGL primitives
- Using built-in primitives
- Transformations

Summary

- Configuring the viewer
  - How to manipulate the viewer/camera
- Event handlers for user input
  - Creating and registering the handler
  - Handling keyboard and mouse events
- Special nodes
  - Switching object appearance during run-time (e.g. Switch)
  - Using nodes to optimize rendering performance (e.g. LOD)
  - Presenting text for the user
- Picking objects in 3D
  - Advantages of using a scenegraph
  - How to implement it, what to think about
  - Picking as an application of intersection tests in general

Questions?