Java and OpenSceneGraph

Overview

• Introduction
• Scene graph structure
  – Node types
• Example
• Vectors, Matrices and Transformations
• Events
• Picking

Introduction

• A scene graph system
• Used for
  – Visual simulations, scientific modeling, games virtual reality, etc
• Examples
  – Boeing – Flight simulation
  – NASA – Earth visualization
  – Indra – Train simulation
  – AutoSim – Driving simulator
  – Veriden’s IOBC – Army soldier training
  – See http://www.openscenegraph.org/

OpenSceneGraph

• Tree structure (Directed Acyclic graph)
• Scene management
• Optimizing graphics rendering
  – Culling
  – Sorting
  – Levels of details
OpenSceneGraph

- C++ API built on OpenGL
- Cross platform
  - Supports most platforms having OpenGL and C++

Java OSG

- Bindings for accessing OSG from Java
  - Native libraries accessing OSG
  - Jar and class files for Java
- C++ and Java code is similar – but not identical…
- JavaOSG is under development
- Not all of OSG is available
- Javadoc is sometimes a bit lacking…
- No tutorials…
  - Some C++ tutorials on the openscenegraph homepage
- 25-30 examples on course home page

Layers

- Applications
- OpenSceneGraph
- OpenGL
- Graphics Hardware

Nodes in the scene graph tree

- Node – base class
  - Group – holds a set of child nodes
  - Transform – transforms all children
  - LightSource – leaf node
  - Switch – switches between children
  - LOD – Level of Detail – switch depends on distance to viewer
- Geode – Leaf node for grouping Drawable:s
  - Billboard – orients drawables to always face the viewer
Nodes in the scene graph tree

- Drawable – abstract base class for drawable graphics
  - Geometry – Holds vertices
  - Text – Text
  - DrawPixels – drawing images using glDrawPixels
  - ShapeDrawable – primitives
  - Particle
- StateSet – encapsulates OpenGL states and attributes
- Texture – encapsulates OpenGL textures

Example

```
Group
  Transform
    Matrix
  Geode
    Drawable
  Geode
    Drawable
```

Root of the scene

Defines the transformation

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

```
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);
```
HelloWorld.java

- Three rotating boxes

Create scene graph and window

```java
Group root = new Group();
Geode cubeGeode1 = new Geode();
... // create cube
cubeGeode1.addDrawable(cubeSD1);
... // create transforms
xform1.addChild(cubeGeode1);
xform.addChild(xform1);
root.addChild(xform);
viewer.setSceneData(root);
viewer.realize();
```

Important parts

- Set up a standard viewer:
  ```java
  Viewer viewer = new Viewer();
  viewer.setUpViewer(VIEWERViewerOptions.STANDARD_SETTINGS.Val);
  ```
- Configures the viewer with "standard settings"
  - Mouse handlers for rotating and moving around the scene
  - A keyboard handler with some useful key mappings, for example
    - 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
Event loop

```java
while(!viewer.done()) {
    viewer.sync();
    viewer.update();
    // Animate, etc...
    viewer.frame();
}
viewer.sync();
System.exit(0);
```

Transformations

- Default coordinate system differs from OpenGL
  - Z up, X right, Y towards screen
- Transformations apply to all child nodes in the tree
  - Allows hierarchies
- Transform nodes:
  - MatrixTransform
    - Has 4x4 matrix (RefMatrixd)
  - PositionAttitudeTransform
    - Has a 3d position (Vec3d) and a Quat attitude
  - AutoTransform
    - Automatically aligns children with screen coordinates

PositionAttitudeTransform

```java
PositionAttitudeTransform xform1 = new PositionAttitudeTransform();
xform1.setPosition(new Vec3dReference(-5,0,0));
xform1.addChild(cubeGeode1);
xform.addChild(xform1);
Quat rot1 = new Quat();
...rot1.setAngle(i/32d); // (rotation)
rot1.setAxis(Vec3d.XAxis);
xform1.setAttitude(rot1);
```

MatrixTransform

```java
MatrixTransform mt = new MatrixTransform();
RefMatrixd matrix = new RefMatrixd();
matrix.makeIdentity();
matrix.makeTranslate(0, 0, 4); // can also do
  makeRotate, makeScale, etc..
mt.setMatrix(matrix);

• preMult()/postMult() for multiplying matrices
```
Transforming vectors

// Rotate defForw around the Z axis

Quat rotLeft = new Quat();
rotLeft.setAxis(Vec3d.ZAxis);
rotLeft.setAngle(angle);
Vec3d forw =
    RefMatrixd.transform3x3(RefMatrixd.rotate(rotLeft),
        defForw);

Vectors and operations

Vec3dReference forw = new Vec3dReference(0, -1, 0);
Vec3d move = new Vec3d(0, 0, 0);
if (moveUp) {
    move = up.mult(-0.4);
} …else if (moveRight) {
    move = forw.mult(0.4).crossProduct(up);
}
pos.addInPlace(move);
transformation.setPosition(pos);

The viewer

• The viewer
  – viewer.setUpViewer(
    VIEWERViewerOptions.STANDARD_SETTINGS_Val);
  – Configures the viewer with “standard settings”
• If you do not want standard handlers, consider e.g.
  – 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

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 that invokes all node
    update and animation callbacks
• viewer.frame();
  – Start traversing the scene for drawing and culling
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();
}

Camera

Camera camera = new Camera();
camera.getRenderSurface().useCursor(false);
camera.getRenderSurface().fullScreen(false);

CameraConfig cameraConfig = new CameraConfig();
cameraConfig.addCamera(new String("myCamera"), camera);

// define what area of the screen should be drawn by this camera;
// the following settings tell it to use the entire screen estate
camera.setProjectionRectangle(0f, 1f, 0f, 1f);
Viewer viewer = new Viewer(cameraConfig);

... camera.setViewByLookat()
camera.setViewByMatrix()
camera.setLensPerspective()

Examples

• 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
Events

- Event handlers for user input
- Extend the GUIEventHandler class
  - Implement your own handle() function
  - Invoked upon keyboard and mouse events

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

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)

Registering the handler

```java
class MyHandler extends GUIEventHandler {...}
MyHandler handler = new MyHandler();
// get the viewer's event handler list
VIEWEREventHandlerList ehl = viewer.getEventHandlerList();
// add to list
ehl.push_front(handler);
```

Handling events

- Keyboard events
  - KEYDOWN, when a key is pushed down
  - KEYUP, when the key is released
- Mouse events
  - MOVE, when moved
  - DRAG, when dragged
  - PUSH / RELEASE, for mouse button events
- Look at the javadocs for full details
Handling events

```java
GUIEVENTADAPTEREventType type = event.getEventType();
if (type == GUIEVENTADAPTEREventType.KEYDOWN) {
    switch (event.getKey()) {
        case (GUIEVENTADAPTERKeySymbol.KEY_Up_Val):
            rotateUp = true;
            break;
    }
}
```

• Mouse events
• `getX/Y()`, return the X/Y coords of the mouse pointer
• `getX/Ynormalized()`, return normalized X/Y coords
  – (-1, -1) at bottom left, (+1, +1) at top right
  – (0,0) is in the center of the screen
• `getButton()`, return the mouse button pushed
• `UserInput.java`

Handling events

• `public boolean handle(...) { ... }`
• Return value?
  – True: no other handlers will be invoked

Picking

```java
INTERSECTVISITORHitList hitlist =
    new INTERSECTVISITORHitList();
if (viewer.computeIntersections(
    event.getXnormalized(),
    event.getYnormalized(),
    0, hitlist)) {
    // 0 – camera number
    ...
}
```
Picking

• Go through the list

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

• Using the geode?
  – pg.setUserData, pg.getUserData
  – More object oriented:
    • Subclass Geode

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

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

Picking

• Can be used to find general intersections as well
• 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

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

// 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");

// SCREEN – face the viewer
label.setAlignment(TEXTAxisAlignment.XY_PLANE);  
label.setAxisAlignment(TEXTAxisAlignment.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);

Examples online

• http://www.sm.luth.se/csee/courses/smd/171/example/
• Some comments in code
• Read the Javadoc...

Next time

– More OpenSceneGraph...
  • Animations
  • Textures
  • Lighting
  • Loading models
    – Milkshape