OpenSceneGraph

Callbacks and animation, models and materials
Overview

• Callbacks
• Animation
• Lightning
• Materials
Callbacks in OSG

• Callbacks are user-defined functions executed upon tree traversal (e.g. when updating, drawing or culling)
  – Can associate a callback with a certain node
  – Upon traversal, every node with a callback gets it executed
Callbacks in OSG

• Remember this code from the event loop?
  
  `viewer.sync();`
  
  `viewer.update();`
  
  `viewer.frame();`

• Invokes any update callbacks for nodes in the tree
  
  – Suitable place for animating an object
Callbacks in OSG

• Why not do it like this?
  – `viewer.sync();`
  – `viewer.update();`
  – `... put animation code in here ...`
  – `viewer.frame();`

• The effect will be the same
  – But callbacks provide a better interface
  – simpler, cleaner, less cluttered, easier to maintain
Callbacks in OSG

• In the "Node" class
  – setCullCallback(…)
  – setEventCallback(…)
  – setUpdateCallback(…)

• NodeCallback, the base class for callbacks
  – The callback function is named
    • operatorMethod(Node, NodeVisitor)
  – Override this function with your own code
Callbacks in OSG

• A callback can access user data via
  – node.setUserDate()
  – node.getUserData()
• The user data can be used to store node-specific information, such as the rotation of a node
  – User data should extend the Referenced class
    • (Reference counted class for garbage collection)
Example – adding a callback to a node

```java
public class MyNodeCallback extends NodeCallback {
    public void operatorMethod(Node node, NodeVisitor nv) {
        ...
    }
}
...
node.setUpdateCallback(new MyNodeCallback());
```
Animation

• “A motion picture made by photographing successive positions of inanimate objects (as puppets or mechanical parts)”
  ...from the Merriam-Webster dictionary.

• I.e. changing an object’s position over time
Cartoon vs. Computer Animation

• Cartoon/computer animation has common features
  – Both work on a frame per frame basis
  – Each frame holds a snapshot of the scene
• In the next frame, the scene has changed slightly
  – The result?
  – The viewer gets the impression of movement
Cartoon vs. Computer Animation

• Framerate, number of frames per second (fps)
  – E.g. 60 fps in computer games, smooth scrolling/updates
  – Film projection, 24 fps, enough to trick the viewer
  – TV: 25 or 29.9997 fps (50/60 fields per second)

• Quality of animations
  – Some cartoons look really good, some others don’t
  – Why?
    • Manual animation by hand drawing is expensive!
      – 24 fps, 90 minute movie, results in 129600 frames
      – One way to make it cheaper is to reduce the number of frames via the framerate
      – Some low budget TV cartoons use e.g. 12 or 6 fps
Keyframe Animation

• Keyframe
  – A frame where objects are
  – Positioned
  – Oriented
  – Scaled
    ...in a predefined way
  – Based on time (normally)
Keyframe Animation

• Interpolating in between keyframes
  – You don’t want to specify every single frame (Like we did earlier)

• Easier to set-up keyframes
  – Let the computer interpolate
  – Comparison with cartoon animations

• Some hand animators also work like this
  – Artists draw the important keyframes
  – Animators draws the frames in between
  – a.k.a. ”inbetweening”, “tweening”
The AnimationPath in OSG

• Defines a path by which an object should move
  – Control points at specific times (seconds, real-world clock)
    • Position
    • Rotation
    • Scaling
  – Interpolates the frames in between the keyframes
    • Can also retrieve points at an arbitrary time, if needed
The AnimationPath in OSG

• Applied via e.g. an AnimationPathCallback
  – transform.setUpdateCallback()
  – Called at the event-loop’s viewer.update()

• Can also be used to record the Viewer’s motion
Animation Path – an example
Animation Path – attitudes

time, \((x_t, y_t, z_t), \text{ attitude}\)
Animation Path

• Loop/repeat mode
  – Looping
    • Repeat forever

• No loop
  – Just play once

• Swinging
  – Play once, then
    – play backwards,
    – then repeat
Example – moving in a square

AnimationPath ap = new AnimationPath();
// add positions for given times
ap.insert(0.0, new ANIMATIONPATHControlPoint(    
    new Vec3dReference(40,40,0)));
ap.insert(2.0, new ANIMATIONPATHControlPoint(    
    new Vec3dReference(-40,40,0)));
ap.insert(4.0, new ANIMATIONPATHControlPoint(    
    new Vec3dReference(-40,-40,0)));
ap.insert(6.0, new ANIMATIONPATHControlPoint(    
    new Vec3dReference(-40,-40,0)));
ap.insert(8.0, new ANIMATIONPATHControlPoint(    
    new Vec3dReference(40,-40,0)));
ap.setLoopMode(ANIMATIONPATHLoopMode.LOOP);

For loops, make sure start/end is identical
Example – penguin moving in a square (AnimDemo.java)

```java
PositionAttitudeTransform xform =
    new PositionAttitudeTransform();
xform.setUpdateCallback(
    new AnimationPathCallback(
        new AnimationPathCallback(ap));
Node node =
    osgDBNamespace.readNodeFile("penguin.obj");
xform.addChild(node);
root.addChild(xform);
```
Example – moving and rotating
(AnimDemo2.java)

Quat q = new Quat();
q.setAxis(Vec3d.ZAxis);
q.setAngle(Math.PI/2);
ap.insert(15,
    new ANIMATIONPATHControlPoint(
        new Vec3dReference(10,20,30), q));
AnimationPath – time-related functions

- `getInterpolatedControlPoint(time, controlPoint)`
  - Return an interpolated point for an arbitrary time
- `getFirstTime()`, `getLastTime()`, `getPeriod()`
- `getMatrix(time, matrix)`
  - Return the transformation matrix for an arbitrary time
Creating Animations with A.P.

- Animations are easy to implement in OSG
  1. Create an AnimationPath
  2. Add control points (keyframes) for specified times
     - Positions, attitudes, scaling
  3. Apply an update callback for the path
     - Apply on a transform node
     - Add geodes etc. to the transform
Other ways of animating objects

• Do everything in display loop…
• Add code in event handlers that manipulate transform nodes
  – If the user can directly manipulate an object, the animation occurs as feedback to the user’s actions
• Add your own update callback
  – Extend e.g. the NodeCallback or AnimationPathCallback
    • Implement/override the operatorMethod()
    • Compute the interpolated positions and rotations
  – More work in terms of code, but also more flexibility
Animating the viewer

• The viewer itself has no update callback
• Can put code between update() / frame()
  
    viewer.sync();
    viewer.update();

    … animate viewer
    viewer.frame();

• And/or use a MatrixManipulator
Matrix manipulators

• The ”MatrixManipulator” class
  – Abstract base class extending the standard GUIEventHandler
  – Defines interface for controlling cameras in response to GUI events
  – Subclasses (test with ”1”, ”2”, ”3” in standard Viewer)
    • TrackballManipulator
      – The standard viewer mode
    • DriveManipulator
      – Left button accelerates, right breaks
    • FlightManipulator
      – Works like a flight simulator
Matrix manipulators, more subclasses

- **AnimationPathManipulator**
  - Follow an animation path
- **KeySwitchMatrixManipulator**
  - Allows for easily changing among multiple manipulators
  - E.g. make a certain key "k" switch to another manipulator
- **NodeTrackerManipulator**
  - Make viewer track a node with/without rotations
- **TerrainManipulator**
  - Make viewer follow the terrain (for rolling hills, etc)
How to apply

• `viewer.addCameraManipulator(…)`
• `Remove others from viewer.setUpViewer()`
Animation

- Using time to decide movement
- Same movement regardless of CPU and GPU speed, etc.

- Problem:
  while(…) {
    animate();
    draw();
  }
Better solution

```java
while(…) {
    time = getTimeElapsed();
    if (moving) {
        distance = 20.0*time;
    }
}
```
Timings

• Initial approach
• In display loop:
  
  time = System.currentTimeMillis() – lastTime;
  // … use time for animations etc.
  lastTime = time

• Problem?
Timings

• 60-100 frames per second?
• 10-16.7 ms per frame
• `System.currentTimeMillis()` resolution?

```java
long ct1 = System.currentTimeMillis();
while((System.currentTimeMillis() - ct1) < 100)
    System.out.println((System.currentTimeMillis() - ct1));
```

Output?
Timings

• 0
• 0
• 0
• ...
• 10
• 10
• 10
• ...
• 20
• 20
• 20
• ...
• 30
• 30
• 30
• ...
• ...
Better solutions

- High resolution timers
- `System.nanoTime()` (Since jdk1.5.0)
- Use the average over the last few values
public static void main(String args[]) {
    long nt1 = System.nanoTime();
    long nt2 = nt1;
    long ct1 = System.currentTimeMillis();
    long ct2 = ct1;
    int i = 100;
    ATimer timer = new ATimer();
    while (i-- > 0) {
        timer.mark();
        nt2 = System.nanoTime();
        ct2 = System.currentTimeMillis();
        System.out.println(((nt2 - nt1) / 1000000) + " " + (ct2 - ct1) + " " + timer.getElapsedTime());
        ct1 = ct2;
        nt1 = nt2;
        try { Thread.sleep(15); } catch (Exception e) {}; 
    }
}
class ATimer {
    long v[] = new long[5];

    public ATimer() {
        for(int i = 0; i < v.length; i++)
            v[i] = System.currentTimeMillis();
    }

    public void mark() {
        for(int i = 0; i < v.length - 1; i++)
            v[i] = v[i + 1];
        v[v.length - 1] = System.currentTimeMillis();
    }

    public long getElapsedTime() {
        return (v[v.length - 1] - v[0])/(v.length-1);
    }
}
Output

0 0 0
14 10 2
15 20 7
15 10 10
15 20 10
17 10 15
15 20 15
15 10 15
16 20 15
14 20 17
18 10 15
14 20 17
15 10 15
20 20 15
14 20 17
15 10 15
14 20 17
15 10 15
15 20 15
15 20 15
14 10 15
Animating avatars - Skeletons and morph target

- Skeleton based animation
  - Scene graph
  - Joints have position and orientation
  - Manipulating joints make the object move
  - One root node
    - E.g. pelvis
Skinning

- Skinning gives a hull to the skeleton
- Vertexes are connected to joints
  - Will be transformed together with their joints
- **Hard skinning**
  - Vertex is connected to only one joint
  - Faster
  - Hard edges
  - Example: counter strike
- **Soft skinning**
  - Vertex is connected to more than one joint
  - More computations
  - Softer edges
  - Example: Unreal Tournament 2004
Keyframes and skeletons

• Define keyframes for joints in a skeleton object
  – To create poses
• Tweening for interpolation between poses
• One set of “hull vertexes” – the mesh
• Keyframes affect the whole mesh
Morph Target

• Linear interpolation of vertex positions in a mesh
• No skeleton
  – Save whole mesh for each keyframe
• Useful for surface changes
  – Where most vertexes are uniquely transformed – facial expressions
• Cheap computational
• Memory demanding
• Example: Quake III Arena
Modeling in OSG
Loading models, setting materials, etc.

- More realistic look
- Lightsources
  - Can change e.g. the "mood" of a scene via the lighting
- Materials
  - Diffuse colour, the "base colour"
  - Specularity/hardness, making plastics, metals, etc...
  - Reflections, (not in the GL pipeline, but in e.g. raytracing)
  - Emission, making objects glow or shine as if by light
- Textures
  - Image mapping
  - Bump mapping, normal mapping, displacement mapping, etc.
Creating models

• Modeling via code
  – Easy for simple models
  – But, requires a fair bit of imagination

• For example:
  • `Sphere(0,-0.2,0,2);`
  • `Sphere(1.3,1.4,0,1);`
  • `Sphere(-1.3,1.4,0,1);`
  • `Sphere(0,0,1.9,0.3);`
Lights and Lightsources

• LightSource
  – Extends the Group node
    • addChild() still possible
  – Contains one Light
    • setLight()
    • getLight()
The Light

• Encapsulates OpenGL glLight() functionality
  – setLightNum(), sets which GL light to operate on
• setAmbient(), ambient colour of the light
• setDiffuse(), diffuse colour component
• setSpecular(), specular component
The Light

• Light can be point source or spotlight
  – setPosition(), positions the light
  – setDirection(), aims the spotlight in the given direction

• Many functions to tweak the lighting
  – setConstantAttenuation(), also Linear and Quadratic

• More spotlight functions
  – setSpotCutoff(distance)
  – setSpotExponent(exp)
The Light – shadows?

• OSG does not support real time shadows by default
• Can be added by user code
Lightsources – example code

Light light1 = new Light();
l light1.setLightNum(0);
light1.setDiffuse(
    new Vec4fReference(1f,0f,1f, 1f));
light1.setPosition(
    new Vec4fReference(x, y, z, 1f));
Lightsources – example code

// create lightsource and set its light
LightSource ls = new LightSource();
ls.setLight(light1);

// turn on the lightsource
ls.setLocalStateSetModes(
STATEATTRIBUTEValues.ON_Val);
Lightsources – example code

// create a “stateset” for the root node
StateSet stateset = new StateSet();
root.setStateSet(stateset);
// turn the lightsource on for this stateset
ls.setStateSetModes(stateset,
STATEATTRIBUTVALUES.ON_Val);

Lightsources – example code

// then add the lightsource to the scene
root.addChild(ls);

• Repeat the process if you need more lightsources
  – You should reuse the same StateSet however

• Keep in mind you can alter light properties during run-time
  – Colour, position, direction, etc...
  – Turn lights on and off, etc...
Summary – how to add a lightsource

1. Create a new LightSource
2. Create a new Light and associate with a GL light
3. Adjust the light parameters; colours, attenuation...
4. Make the lightsource use your light (via setLight())
5. Create a stateset for the part of the scenegraph tree
6. that should be illuminated by the lightsource
7. Turn on the lightsource in the stateset
8. Add the lightsource to the scene
What is a StateSet?

• It encapsulates OpenGL state modes and attributes
• Used for textures, lightsources, materials, transparency...
• E.g. specifies textures for Drawables
  – Drawables can share a single StateSet
• Sharing statesets is recommended! Why?
  – It minimizes state changes in the GL pipeline
  – Changing the state in the pipeline is expensive!
  – Thus, sharing the same StateSet is a good thing
Materials in OSG

• The "Material" class
  – Encapsulates OpenGL’s glMaterial state

• Usage
  – Create a new Material and set its properties
  – Apply the material to a StateSet via
    • stateset.setAttribute(material);
  – Apply the StateSet to e.g. a Geode
    • geode.setStateSet(stateset);
The Material class – some functions

setDiffuse(MATERIALFace, Vec4fReference)
setSpecular(...)
setAmbient(...)
setEmission(...)
setShininess(...)
setAlpha/setTransparency(...)
Example – creating an orange plastic ball

shape = new ShapeDrawable(
    new Sphere(new Vec3fReference(0f, 0f, 0f), 1f));
geode = new Geode();
stateset = new StateSet();
material = new Material();
material.setDiffuse(MATERIALFace.FRONT_AND_BACK,
    new Vec4fReference(1.0f, 0.5f, 0.1f, 1f));
material.setSpecular(MATERIALFace.FRONT_AND_BACK,new
    Vec4fReference(1.0f, 1.0f, 1.0f, 1f));
material.setShininess(MATERIALFace.FRONT_AND_BACK, 63f);
stateset.setAttribute(material);
geode.setStateSet(stateset);
geode.addDrawable(shape);
Example – a window

shape = new ShapeDrawable(new Box(new Vec3fReference(-1f, -1f, 0f), 1f, 0.2f, 2f));
geode = new Geode();
stateset = new StateSet();
material = new Material();
// set the diffuse colour component to a lightblue nuance
material.setDiffuse(MATERIALFace.FRONT_AND_BACK, new Vec4fReference(0.4f, 0.8f, 1f, 1f));
stateset.setMode(GL.GL_BLEND, STATEATTRIBUTEValues.ON_Val);
stateset.setRenderingHint(STATESETRenderingHint.TRANSPARENT_BIN_Val);
material.setTransparency(MATERIALFace.FRONT_AND_BACK, 0.6f);
stateset.setAttribute(material);
geode.setStateSet(stateset);
geode.addDrawable(shape);
root.addChild(geode);
Creating models with modeling programs

- No need to do things by code…
- Many programs and file formats available
- In the labs: MilkShape 3D
Introduction to MilkShape 3D

• A basic low-polygon modeler
• Originally designed for a game (Half-Life)
• Supports basic operations
  – Select, move, rotate, scale, extrude, etc
  – Low-level editing of vertices and faces
  – Primitives (spheres, boxes, cylinders…)
• Skeletal animation capabilities
• Supports many file formats from many games/engines
Introduction to MilkShape 3D

• Why MilkShape
  – Relatively easy to use and learn (compared to other tools…)
  – It’s powerful enough for this course
  – Handles a lot of different formats
  – Allows for 30-day free trial
  – It’s fairly cheap

• If you have access to other tools that OSG have loaders for – use them
MilkShape 3D

• Short demo
Exporting

- Autodesk 3DS
- LightWave 6.5x LWO
- Wavefront OBJ
  - Materials in an extra MTL file
- 3DS or LWO recommended
Loading objects – code example

Import
    openscenegraph osgDB osgDBNamespace;
Node node =
    osgDBNamespace.readNodeFile("somefile.lwo");
scene.addChild(node);
What to think about for external models

• Retrieving the dimensions for the model
  – Getting the BoundingSphere via node.getBounds()
    • getRadius()
  – E.g. for ”normalizing” the size of an unknown object
Skeleton animations?

- Supported by MilkShape but -
- No support by default in OSG…
Next time

• Next time
  – Buffers
  – Textures
  – Aliasing and anti-aliasing

• Then:
  – Pipeline implementation
    • Basics, clipping, scan conversion, shaders, etc
  – Example exam questions

• No lecture tomorrow
• One next week