Virtual Reality Technology and Programming
TNM053: Lecture 7: Scene Graphs

OpenGL and VR
- OpenGL (+GLUT)
  - Basic primitives
  - Complex transformations
  - No visible interrelations
  - No interactors
- Hard to manage complex scenes
- Hard to manage interaction

Example: A car
- Define (display lists) for
  - Body shape
  - Wheel
  - Wheel 'nut'
- In model space:
  - Perform the transforms
  - Insert the geometries

Process pseudocode
Draw body
For i=1,4
  Push matrix
  Transform to wheel i
  Draw wheel
  For j=1,n
    Push matrix
    Transform to nut j
    Draw nut
    Pop matrix
  Pop matrix

OpenGL and VR (2)
- VR
  - Complex scenes
  - Many related and connected objects
  - Complex interactors
- OpenGL cannot easily give us this
- Scene Graphs can help

Scene Graph and OpenGL
Software: API’s

- OpenInventor - http://www.sgi.com/software/inventor/
- Maverik - http://aig.cs.man.ac.uk/systems/Maverik/
- OpenRM - http://www.r3vis.com/
- Open Scene Graph - http://www.openscenegraph.org/
- OSG - http://www.osg.org/

Software: Applications

- Ray Tracers
- Modellers / Animation Packages
- Virtual Reality Markup Language (VRML)
- Games
- Pretty much *any* high performance computer graphics system

S.G. software provides...

- Wide selection of primitives
- Wide selection of materials
- Lots of interactors
- Lighting objects
- Cameras
- Viewers
- But that’s not the point...

The Scene Graph *itself* provides...

- High performance rendering for less effort
- Easy management of large numbers of objects in scene
- Ease of management of interaction
- Ease of management of modification

What is a Scene Graph (1)

- Directed Acyclic Graph of objects.

What is a Scene Graph (2)

- Often spatial ordering of objects
  - Helps with depth-sorting
  - Important for multipass rendering
- Moves the structure of the scene from the program into a data structure.
- Scene Graph API provides mechanism to manage this
Why use a Scene Graph?
- Models often hierarchical in nature.
- Management of details.
- Convenience
- Facilitate rapid application development.
- Exploit reuse

What's in a Scene Graph?
- A root
- Nodes
- Directed edges (parent to child)

Nodes
- 3 Basic types:
  - Shape or geometry
  - Material or Property
  - Group
- Each has attached qualities (fields)
- Other control nodes
  - System dependent

Shape/ Geometry
- Defines a 3D geometric entity:
  - Point
  - Vector
  - Polygon ...
  - Other primitive:
    - Sphere
    - Cone
    - Cylinder
    - Curves (NURBS)

Property
- Colour qualities:
  - Ambient
  - Specular
- Shininess
- Transparency
- Textures
- Transformations:
  - Scale
  - Rotation
  - Translation

Group
- Node used to combine a sub-graph
Ordering and Traversal

- Layout is less abstract than it seems!
- Traversal occurs according to rules:
  - Typically left-to-right
  - Typically ‘bottom-up’

Water Molecule example

(Open Inventor) water molecule in C++

- Groups:
  - SoGroup *waterMol = new SoGroup;
  - SoGroup *oxygen = new SoGroup;
  - SoGroup *hydrogen1 = new SoGroup;
  - SoGroup *hydrogen2 = new SoGroup;

- Shapes:
  - SoSphere *sphere1 = new SoSphere;
  - SoSphere *sphere2 = new SoSphere;
  - SoSphere *sphere3 = new SoSphere;

Open Inventor water molecule in C++ (2)

- Materials:
  - SoMaterial *redplastic = new SoMaterial;
  - SoMaterial *whiteplastic = new SoMaterial;

- Transforms:
  - SoTransform *Xform1 = new SoTransform;
  - SoTransform *Xform2 = new SoTransform;

Open Inventor water molecule in C++ (3)

- Fill in fields for the Materials:
  - redplastic->ambientColor.setValue(1.0,0.0,0.0);
  - redplastic->diffuseColor.setValue(1.0,0.0,0.0);
  - redplastic->specularColor.setValue(0.5,0.5,0.5);
  - redplastic->shininess = 0.5;

- Similar for whiteplastic

Open Inventor water molecule in C++ (4)

- Fill in fields for the transforms:
  - Xform1->scaleFactor.setValue(0.75,0.75,0.75);
  - Xform1->translation.setValue(0.0,-1.2,0.0);
  - Xform2->translation.setValue(1.185,1.388,0.0);

- Leaves us with a bunch of nodes...
Open Inventor water molecule in C++ (5)

- Build the graph:
  - water->addChild(oxygen);
  - water->addChild(hydrogen1);
  - water->addChild(hydrogen2);
  - oxygen->addChild(redplastic);
  - oxygen->addChild(sphere1);
  - hydrogen1->addChild(whiteplastic);
  - hydrogen1->addChild(Xform1);
  - hydrogen1->addChild(sphere2);
  - hydrogen2->addChild(Xform2);
  - hydrogen2->addChild(sphere3);

Open Inventor water molecule in C++ (6)

- Add some initialization code
- Add a few more instances of water
- Run a molecular simulation to -10°C
- Add some code to trigger rendering
- And...

...Ice! (unit cell)

Points to note

- Why doesn’t Xform2 have a scale field?
- Why doesn’t hydrogen2 have a child “whiteplastic”?
- Traversal!
  - Traversal carries state
  - State is overwritten by nodes

Traversing the graph

Managing state

- Need to manage state:
  - in OpenGL use push and pop of matrix
  - In scene graphs use ‘separator’ nodes
- Separator node causes:
  - push entire state on entry
  - pop on exit
- Separator is derived from group
Managing state

Shared nodes

- Nodes can be shared.
- Saves space
- Reduces number of objects to be specified
- Makes updates easier and faster

Shared nodes (2)

- Shape node shared
- Traversal unaffected
- State management !!

Other nodes

- Camera nodes
- Lights
- Control nodes:
  - Usually classed as ‘group’ nodes
- Manipulators/interactors

Camera nodes

- Derived from shape node
- Orthographic/Perspective
- Fields:
  - Position
  - Orientation
  - Viewport (angles/size)
  - Near/far clipping distances
  - Etc.
- Managed and traversed just like all nodes

Positioning cameras
**Lights**

- Types:
  - Point
  - Directional
  - Spot
- Fields:
  - Position
  - On/off
  - Colour
  - Intensity
  - Direction
  - Etc.

**Control nodes**

- Control the scene graph through defining branches of traversal
- In Inventor:
  - SoSwitch
  - SoBlinker
  - SoSelection
  - SoLevelOfDetail

**SoSwitch**

- One or more children
- A field determines which *one* to visit

**SoBlinker**

- Time-based
- Selects *one* of it's children
- Use to define simple animations
**Control nodes:**

**SoLevelOfDetail**
- Left to right traversal of children
- Most detailed to least detailed
- Draw first whose *projected* 3D bounding box fits the specified range

**Control nodes:**

**SoSelection**
- Node inserted into graph to manage object selection
- Determines which objects can be selected
- Permits child objects to be selected
- Determines mode of selection:
  - single
    - clear list each time an object is selected
  - multi
    - add each selected object to list
    - Might have to use special key: shift key?

**Manipulators/Interactors**
- **Draggers**
  - E.g. scaler, rotator, translator etc.
  - Add to a group
  - Connect output field to (e.g.) shape fields
  - Scale, rotate or translate object
  - Can build in your own callbacks
- **Manipulators**
  - Subclass of other nodes (e.g. transform)
  - Include ‘hidden’ dragger

**Manipulator: Trivial Example**
**Other, other nodes**

- Many many types:
  - Culling controls
  - Transform controls
  - Texture controls
  - Transparency controls
  - Picking controls
  - Callback controls
- Typically system dependent

**‘Sensors’ and ‘Engines’**

- Sensors respond to changes in the DB
  - Changes in values etc
- Engines define relationships between fields.
  - Animation, constraints

**Sensors**

- Respond to DB changes
  - Value changes
  - Timer events
- Invoke user-defined callbacks
  - Do stuff - whatever you want

**Sensor example**

```c
SoCamera *camera = myViewer->getCamera;
SoFieldSensor *mySensor = new SoFieldSensor(camerachangeCB,camera);
mySensor->attach(&camera->position);
```

**‘Engines’**

- Connect fields together within the scene graph
- Engines are ‘filters’ with inputs and output(s) to control the transfer of data
- Many types:
  - Boolean, Calculator, Compose matrix (transforms), gates
  - Other - define your own

**Sensors and Engines**

- Using Sensors and engines you can:
  - Build in and control animation
  - Build in responses to user stimuli
  - Add complex callbacks to carry out user-defined functions like:
    - Molecular simulation
    - Haptic interaction
### Benefits of scene graph based methods

- Fast, precise rendering
- Fast, precise culling
- Fast (easy) application updates
- Manageable interactors

### Benefits: Rendering

- Scene graph describes whole, consistent state of the objects in the scene
- Scene graph traversal allows:
  - System knows position/size of each object in scene simultaneously
  - Perhaps uses bounding box information to speed up
  - System can order rendering correctly
  - Can apply multi-pass rendering for transparent objects

### Benefits: Culling

- Scene graph describes whole, consistent state of the objects in the scene
- Scene graph traversal allows:
  - System knows position/size of each object in scene simultaneously
  - Can pre-cull occluded objects correctly
  - Can pre-cull back-face polygons
  - Can pre-calculate clipping

### Benefits: Updates

- Scene graph knows about relations:
  - What is connected to what
  - What occludes what
  - What is where in the rendered image
- Updates:
  - Changes propagate to the final images
  - Minimize re-rendering
  - Maximize re-use

### Benefits: Interactors

- OpenGL/GLUT:
  - 2D information comes back
  - Ok for 2D image
  - Not for 3D - hard to locate objects
- Scene graph:
  - Knows about objects in 3D scene
  - Can use 2d information to backtrack
  - Locates 3D object which is selected
  - Also allows use of 3D interaction within scene

### Summary

- Scene graphs move objects from programmer control to database
- Saves a lot of programming
- Allows:
  - Easier definition of complex scenes
  - Easier definition of interaction
- Exploits:
  - Scene relationships of objects
Summary (2)

- Manage state carefully and...
- Use sensors and 'engines' to connect node fields:
  - complex animation effects
  - Little effort required