CSC492/592
Introduction to Scene Graphs

Instructor: Dr. L. Miguel Encarnação
imedia
- The ICPNM Academy -
Objective

A discussion of the issues involved in designing a scene graph in general and OpenInventor in particular.
Motivation
Concept of Scene Graph

- Objects placed relative to one another
- Objects made of similar components
- Directed acyclic graph
Motivation
Use for Animation/Modeling
Motivation
Use for Animation/Modeling

- One object has a **local transformation** relative to its parent
  - shoulder is translation \((0\ 1\ 0)\) from base
  - upper arm is translation \((0\ 3\ 0)\) from shoulder
  - elbow is translation \((0\ 3\ 0)\) from upper arm
  - fore arm is rotation \(Z\) by -90 then translation \((0\ 2\ 0)\)
Sharing Nodes

- E.g. One table many places
- Table1 has CTM T1T0
- Table2 has CTM T2T0
## O-O vs. Scene Graph

<table>
<thead>
<tr>
<th></th>
<th>Non O-O</th>
<th>O-O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non Scene Graph</td>
<td>OpenGL</td>
<td>C++/Java binding of OpenGL, RenderWare</td>
</tr>
<tr>
<td>Scene Graph</td>
<td>PHIGS</td>
<td>OpenInventor, Java3D, VRML/X3D</td>
</tr>
</tbody>
</table>

OpenSG
Application Programmer Interface

- Classes and methods defined by a library
Hierarchical Organization
Example: Water Molecule
Organizing Scenes Hierarchically

Transformation
Shape

T0

T1

T2

S1

S2

T0 T1

T0 T2
Creating Scene Hierarchies

- Creating hierarchies procedurally
  - Sequence of commands
  - Commands repeatedly executed
- Creating hierarchy using objects
  - Set up data structures
  - Render by parsing data structures
Creating Scene Hierarchies

- Creating hierarchies procedurally
  - Sequence of commands
  - Commands repeatedly executed
- Creating hierarchies using objects
  - Set up data structures
  - Render by parsing data structures
Creating Hierarchies Procedurally

```
setModelingMatrix ( T0 )
```

Modeling Matrix Stack

```
T0 T1
T0 T2
```
Creating Hierarchies Procedurally

glMatrixMode(GL_MODELVIEW);
glPushMatrix();
glTranslated(-1, 0, 0);
Creating Hierarchies Procedurally

Modelling Matrix Stack

\[ \text{pushMatrix}( ) \]
\[ \text{multiMatrix}( T_1 ) \]
\[ \text{draw geometry} \]
Creating Hierarchies Procedurally

- `pushMatrix()`
- `multiMatrix(T_1)`
- `draw geometry`
- `popMatrix()`;

$T_0$, $T_1$, $T_2$, $T_{0T_1}$, $T_{0T_2}$
Creating Hierarchies Procedurally

Modeling Matrix Stack

\[
\begin{align*}
&\text{pushMatrix ( )} \\
&\text{multMatrix ( } T_2 \text{ )} \\
&\text{draw geometry} \\
&\text{popMatrix ( )};
\end{align*}
\]
Creating Scene Hierarchies

- Creating hierarchies procedurally
  - Sequence of commands
  - Commands repeatedly executed
- Creating hierarchies using objects
  - Set up data structures
  - Render by parsing data structures
Creating Hierarchies Using Objects

- Create transformation objects $T_0$, $T_1$, $T_2$
- Create two geometry objects
- Assemble in a suitable data structure
Nodes of the Graph

Nodes = \{T_0, T_1, T_2, S_1, S_2\}
Edges of the Graph

Nodes = \{T_0, T_1, T_2, S_1, S_2\}
Edges = \{(T0,T1), (T1, S_1), (T0,T2), (T2, S_2)\}

but edges do not need to be explicitly represented
A Binary Tree Implementation

class Node {
    Content* c;
    Node* left;
    Node* right;
    …
};

- Inefficient access to children
An N-ary Tree Implementation

class Node {
    Content* c;
    vector<Node*> children;
    ...
};

- Number of children
- Child deletion
Directed Acyclic Graphs

identical geometry
Traversing a Scene Graph
Render Visitors

- Transformation node
  - update transformation matrix
- Shape node
  - draw
Render Visitors::Transformations

- Must get object definitions in WC before passing to camera
- For object under Base
  - p.B is in WC
- “inherit” matrices down stack
- So for object under shoulder
  - p.SB is in WC
  - (p.S is in base coordinates)
Render Visitors::Transformations

- On traverse
  - “push” on graph descend
  - “pop” on graph ascend
- Combined matrix is current transform (CTM)
Creating Scene Hierarchies

Comparison

- Creating hierarchies procedurally
  - Immediate mode
  - Lower-level programming
  - Hardware optimization
  - Rendering of program-generated data (e.g., particles)
  - Less Memory, more speed

- Creating hierarchies using objects
  - SceneGraphs
  - Higher-level abstraction
  - Platform independence
  - Rendering of objects that map easily to traditional modeling paradigms (polygonal, iso-surfaces, ...)
  - Easy to extend
Object Hierarchy vs. Class Hierarchy
See you again …

On our Field Trip, 2/13!
Lights and Cameras
The Camera

- Non scene-graph issues
  - Specifying the camera
- Scene-graph issues
  - Relation of camera object to the scene graph?
  - Transformations affecting the camera?
The Camera

- Non scene-graph issues
  - Specifying the camera

- Scene-graph issues
  - Relation of camera object to the scene graph?
  - Transformations affecting the camera?
Non-Scene Graph Issue:
Specifying the Camera/View Frustum

view direction

view angle

viewpoint

near

up

view direction

far
Camera Object:
Single Class

<table>
<thead>
<tr>
<th>Camera</th>
</tr>
</thead>
<tbody>
<tr>
<td>viewpoint: Point3d</td>
</tr>
<tr>
<td>viewDirection: Direction3d</td>
</tr>
<tr>
<td>upDirection: Direction3d</td>
</tr>
<tr>
<td>near, far, angle: double</td>
</tr>
</tbody>
</table>
Camera Object: Parallel and Perspective Classes
The Camera

- Non scene-graph issues
  - Specifying the camera

- Scene-graph issues
  - Relation of camera object to the scene graph
  - Transformations affecting the camera
The Camera

- Non scene-graph issues
  - Specifying the camera
- Scene-graph issues
  - Relation of camera object to the scene graph
  - Transformations affecting the camera
Is the Camera a Node in the Scene Graph?

1. User is immersed in the virtual world
   - Camera is a node in the scene graph
2. User is looking (through a window) at the virtual world
   - Camera is not a node in the scene graph
OpenInventor: Camera Is Part of the Scene Graph
Java3D: Camera Is Not Part of the Scene Graph
The Camera

- Non scene-graph issues
  - Specifying the camera
- Scene-graph issues
  - Relation of camera object to the scene graph
  - Transformations affecting the camera
Transformations Affecting the Camera
Transformations Affecting the Camera
Light Sources

- Non scene-graph issues
  - Specifying the light source
- Scene-graph issues
  - Subset of scene graph illuminated
  - Transformations affecting the light
Light Sources

- Non scene-graph issues
  - Specifying the light source
- Scene-graph issues
  - Subset of scene graph illuminated
  - Transformations affecting the light
Non Scene-Graph Issue: Specifying the Light

- Light type:
  - Directional light:
    - Light at infinity
  - Point light:
    - Light at proximity
  - Spot light:
    - Specific direction and spread angle

- Light intensity
- Light color
Light Classes

```
Light
{abstract}

DirectionalLight
PointLight
SpotLight
```
Light Sources

- Non scene-graph issues
  - Specifying the light source
- Scene-graph issues
  - Subset of scene graph illuminated
  - Transformations affecting the light
Light Sources

- Non scene-graph issues
  - Specifying the light source
- Scene-graph issues
  - Subset of scene graph illuminated
  - Transformations affecting the light
Subset of Scene Graph Illuminated

1. Light illuminates nodes traversed after it:
   1. Scene Graph parsed sequentially
   2. Scene Graph parsed hierarchically
2. Light illuminates a region of influence
Subset of Scene Graph Illuminated

1. Light illuminates nodes traversed after it:
   1. Scene Graph parsed sequentially
   2. Scene Graph parsed hierarchically
2. Light illuminates a region of influence
Light Illuminates Geometry Traversed After It
Subset of Scene Graph Illuminated

1. Light illuminates nodes traversed after it:
   1. Scene Graph parsed sequentially
   2. Scene Graph parsed hierarchically

2. Light illuminates a region of influence
Light Illuminates Descendant Geometry
Subset of Scene Graph Illuminated

1. Light illuminates nodes traversed after it:
   1. Scene Graph parsed sequentially
   2. Scene Graph parsed hierarchically
2. Light illuminates a region of influence
Light Illuminates a Region of Influence

Light

setInfluencingBounds(BoundingBox bb)
Light Sources

- Non scene-graph issues
  - Specifying the light source
- Scene-graph issues
  - Subset of scene graph illuminated
  - Transformations affecting the light
Transformations Affecting Light
The End

Email: me@crvg.edu
Choice of a Node Class Hierarchy
Object Hierarchy vs. Class Hierarchy
Class Hierarchy

Class Graph

- Node
  - void addChild(Node n)
- Group
Possible Class Hierarchy (1)

- **SoNode**
  - Abstract
- **SoSeparator**
  - void addChild(Node n)
- **SoTransformation**
- **(Shape)**
Possible Class Hierarchy (2)

- SceneGraphObject
  - {abstract}
  - Group
    - TransformGroup
      - void addChild(SceneGraphObject n)
    - (Shape)
Children of Transform Groups

```
Group
  void addChild(Node n)

TransformGroup {abstract}

ScaleGroup
RotateGroup
TranslateGroup

Node
```
Group* root = new Group;
TranslateGroup* tgrp1 =
    new TranslateGroup(Vector3d(1,0,0));
TranslateGroup* tgrp2 =
    new TranslateGroup(Vector3d(-1,0,0));
Shape* cube = new Shape("cube.vpl");
Shape* cone = new Shape("cone.vpl");

root->addChild(tgrp1);
root->addChild(tgrp2);
tgrp1->addChild(cube);
Code Example (OpenInventor)

Group* root = new Group;
TranslateGroup* tgrp1 =
    new TranslateGroup(Vector3d(1,0,0));
TranslateGroup* tgrp2 =
    new TranslateGroup(Vector3d(-1,0,0));
Shape* cube = new Shape("cube.vpl");
Shape* cone = new Shape("cone.vpl");

root->addChild(tgrp1);
root->addChild(tgrp2);
tgrp1->addChild(cube);
tgrp2->addChild(cone);
Control & Event Handling
Programming the application
Read-evaluation loop

- The application has complete control over the events received

```plaintext
repeat
  read-event(myevent)
  case myevent.type
    type_1:
      do type_1 processing
    type_2:
      do type_2 processing
    ...
    type_n:
      do type_n processing
  end case
end repeat
```
Read-evaluation based UI (X11)

do {
    opt = getnextevent(&x, &y, &n);
    switch(opt) {
        /* what kind of event? */
        case ButtonPress:
            processbuttonpress(opt, x, y, n);
            break;
        case Expose:
            refreshwindow();
            break;
    }
} while (opt != QUIT);
Programming the application
Notification-based approach

- The process receiving the events resides outside the client application

![Diagram of application and notifier process flow]
Notification-based UI
(glut/OpenGL)

```c
void
display(void)
{
    glDrawBuffer(GL_BACK_LEFT);
    glClearColor(1.0, 0.0, 0.0, 1.0); /* red */
    glClear(GL_COLOR_BUFFER_BIT);
    glDrawBuffer(GL_BACK_RIGHT);
    glClearColor(0.0, 0.0, 1.0, 1.0); /* blue */
    glClear(GL_COLOR_BUFFER_BIT);
    glutSwapBuffers();
}

int
main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_STEREO);
    glutCreateWindow("stereo example");
    glutDisplayFunc(display);
    glutMainLoop();
    return 0; /* ANSI C requires main to return int. */
}
```
Notification-based UI
(OpenInventor)

```cpp
void main(int argc, char **argv)
{
    Widget myWindow = SoXt::init(argc[0]);
    if (myWindow == NULL) exit(1);

    SoSeparator *root = new SoSeparator;
    root->ref();
    SoMaterial *myMaterial = new SoMaterial;
    myMaterial->diffuseColor.setValue(1.0, 0.0, 0.0);
    root->addChild(myMaterial);
    root->addChild(new SoCone);

    // Set up viewer:
    SoXtExaminerViewer *myViewer =
        new SoXtExaminerViewer(myWindow);
    myViewer->setSceneGraph(root);
    myViewer->setTitle("Examiner Viewer");
    myViewer->show();
    SoXt::show(myWindow);
    SoXt::mainLoop();
}
```