CS4620/5620: Lecture 7

Scene Graphs

Announcements

• HW 1 out

• PA 1 will be out on Wed

• Next week practicum will have an office hour type session on Open GL
Example

• Can represent drawing with flat list
  – but editing operations require updating many transforms

Groups of objects

• Treat a set of objects as one
• Introduce new object type: group
  – contains list of references to member objects

• This makes the model into a tree
  – interior nodes = groups
  – leaf nodes = objects
  – edges = membership of object in group
Example

• Add group as a new object type
  – lets the data structure reflect the drawing structure
  – enables high-level editing by changing just one node

The Scene Graph (tree)

• A name given to various kinds of graph structures (nodes connected together) used to represent scenes
• Simplest form: tree
  – just saw this
  – every node has one parent
  – leaf nodes are identified with objects in the scene
Concatenation and hierarchy

• Transforms associated with nodes or edges
• Each transform applies to all geometry below it
  – want group transform to transform each member
  – members already transformed—concatenate
• Frame transform for object is product of all matrices along path from root
  – each object’s transform describes relationship between its local coordinates and its group’s coordinates
  – frame-to-canonical transform is the result of repeatedly changing coordinates from group to containing group

Instances

• Simple idea: allow an object to be a member of more than one group at once
  – transform different in each case
  – leads to linked copies
  – single editing operation changes all instances
Example

- Allow multiple references to nodes
  - allows editing of repeated parts in one operation

The Scene Graph (with instances)

- With instances, there is no more tree
  - an object that is instanced multiple times has more than one parent
- Transform tree becomes DAG
  - directed acyclic graph
  - group is not allowed to contain itself, even indirectly
- Transforms still accumulate along path from root
  - now paths from root to leaves are identified with scene objects
Implementing a hierarchy

• Object-oriented language is convenient
  – define shapes and groups as derived from single class

  abstract class Shape {
    void draw();
  }

  class Square extends Shape {
    void draw() {
      // draw unit square
    }
  }

  class Circle extends Shape {
    void draw() {
      // draw unit circle
    }
  }

Implementing traversal

• Pass a transform down the hierarchy
  – before drawing, concatenate

  abstract class Shape {
    void draw(Transform t_c);
  }

  class Square extends Shape {
    void draw(Transform t_c) {
      // draw t_c * unit square
    }
  }

  class Circle extends Shape {
    void draw(Transform t_c) {
      // draw t_c * unit circle
    }
  }

  class Group extends Shape {
    Transform t;
    ShapeList members;
    void draw(Transform t_c) {
      for (m in members) {
        m.draw(t_c * t);
      }
    }
  }
Basic Scene Graph operations

• Editing a transformation
• Getting transform of object in canonical (world) frame
  -- traverse path from root to leaf
• Grouping and ungrouping
  -- can do these operations without moving anything
  -- group: insert identity node
  -- ungroup: remove node, push transform to children
• Reparenting
  -- move node from one parent to another
  -- can do without altering position

Adding more than geometry

• Objects have properties besides shape
  -- color, shading parameters
  -- approximation parameters (e.g. precision of subdividing curved surfaces into triangles)
  -- behavior in response to user input
  -- ...
• Setting properties for entire groups is useful
  -- paint entire window green
• Many systems include some kind of property nodes
  -- in traversal they are read as, e.g., “set current color”
Scene Graph variations

- Where transforms go
  - in every node
  - on edges
  - in group nodes only
  - in special Transform nodes
- Tree vs. DAG
- Nodes for cameras and lights?

Transformations in OpenGL

```c
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
{
  // Draw something:
  glPushMatrix();
  glTranslatef(...);
  glRotatef(15f, ...);
  // set color and draw simplices
  glBegin(GL_TRIANGLES);
  glColor3f(...);
  glVertex3f(...);
  glVertex3f(...);
  glVertex3f(...);
  glEnd();
  glPopMatrix(); // toss old transform
}
{
  // Draw something else:
  glPushMatrix();
  ...
  glPopMatrix(); // toss old transform
}
```
Transformations in OpenGL

- Stack-based manipulation of model-view transformation, $M$
  - `glMatrixMode(GL_MODELVIEW)` Specifies model-view matrix
  - `glLoadIdentity()` $M \leftarrow 4\times4$ identity
  - `glTranslatef(float ux, float uy, float uz)` $M \leftarrow M T$
  - `glRotatef(float theta, float ux, float uy, float uz)` $M \leftarrow M R$
  - `glScalef(float sx, float sy, float sz)` $M \leftarrow M S$
  - `glLoadMatrixf(float[] A)` $M \leftarrow A$ (Note: column major)
  - `glMultMatrixf(float[] A)` $M \leftarrow MA$ (Note: column major)
- Manipulate matrix stack using:
  - `glPushMatrix()`
  - `glPopMatrix()`

Where are we?

- Transforms
  - 2D
  - Affine
  - 3D including rotation
  - Principles of Perspective vs. Orthographic
  - Hierarchical Transforms
- Input: Scene Graphs
- Now, how to produce a picture
  - Viewing Transforms (including perspective)
  - Hardware pipeline
  - Programmable Shading