Graphical Objects and Scene Graphs

Objectives

- Introduce graphical objects
- Generalize the notion of objects to include lights, cameras, attributes
- Introduce scene graphs

Limitations of Immediate Mode Graphics

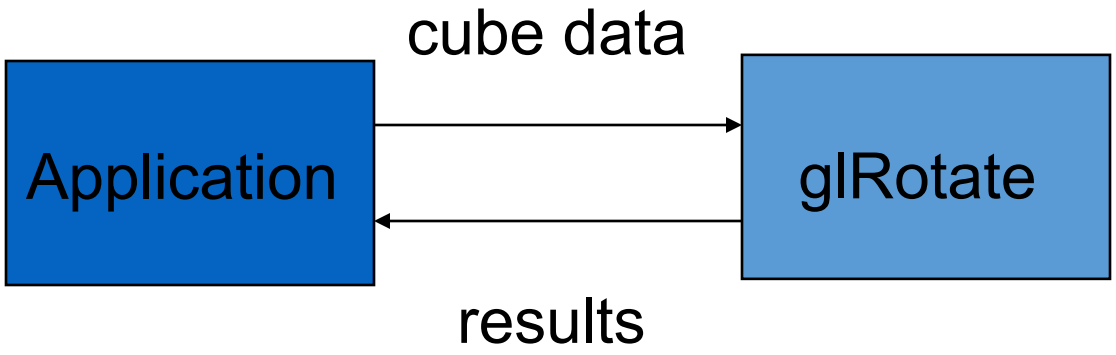
- When we define a geometric object in an application, upon execution of the code the object is passed through the pipeline
- It then disappears from the graphical system
- To redraw the object, either changed or the same, we must reexecute the code
- Display lists provide only a partial solution to this problem

OpenGL and Objects

- OpenGL lacks an object orientation
- Consider, for example, a green sphere
  - We can model the sphere with polygons or use OpenGL quadrics
  - Its color is determined by the OpenGL state and is not a property of the object
- Defies our notion of a physical object
- We can try to build better objects in code using object-oriented languages/techniques

Imperative Programming Model

- Example: rotate a cube



- The rotation function must know how the cube is represented
  - Vertex list
  - Edge list

Object-Oriented Programming Model

- In this model, the representation is stored with the object

- The application sends a *message* to the object
- The object contains functions (*methods*) which allow it to transform itself

C/C++

- Can try to use C structs to build objects
- C++ provides better support
  - Use class construct
  - Can hide implementation using public, private, and protected members in a class
  - Can also use friend designation to allow classes to access each other

Cube Object

- Suppose that we want to create a simple cube object that we can scale, orient, position and set its color directly through code such as

```
cube mycube;
mycube.color[0]=1.0;
mycube.color[1]= mycube.color[2]=0.0;
mycube.matrix[0][0]=……
```

Cube Object Functions

- We would also like to have functions that act on the cube such as
  - `mycube.translate(1.0, 0.0,0.0);`
  - `mycube.rotate(theta, 1.0, 0.0, 0.0);`
  - `setcolor(mycube, 1.0, 0.0, 0.0);`
- We also need a way of displaying the cube
  - `mycube.render();`

Building the Cube Object

```
class cube {
   public:
      float color[3];
      float matrix[4][4];
   // public methods

   private:

   // implementation

}
```

The Implementation

- Can use any implementation in the private part such as a vertex list
- The private part has access to public members and the implementation of class methods can use any implementation without making it visible
- Render method is tricky but it will invoke the standard OpenGL drawing functions such as **`glVertex`**

Other Objects

- Other objects have geometric aspects
  - Cameras
  - Light sources
- But we should be able to have nongeometric objects too
  - Materials
  - Colors
  - Transformations (matrices)

Application Code

```
cube mycube;
material plastic;
mycube.setMaterial(plastic);

camera frontView;
frontView.position(x ,y, z);
```

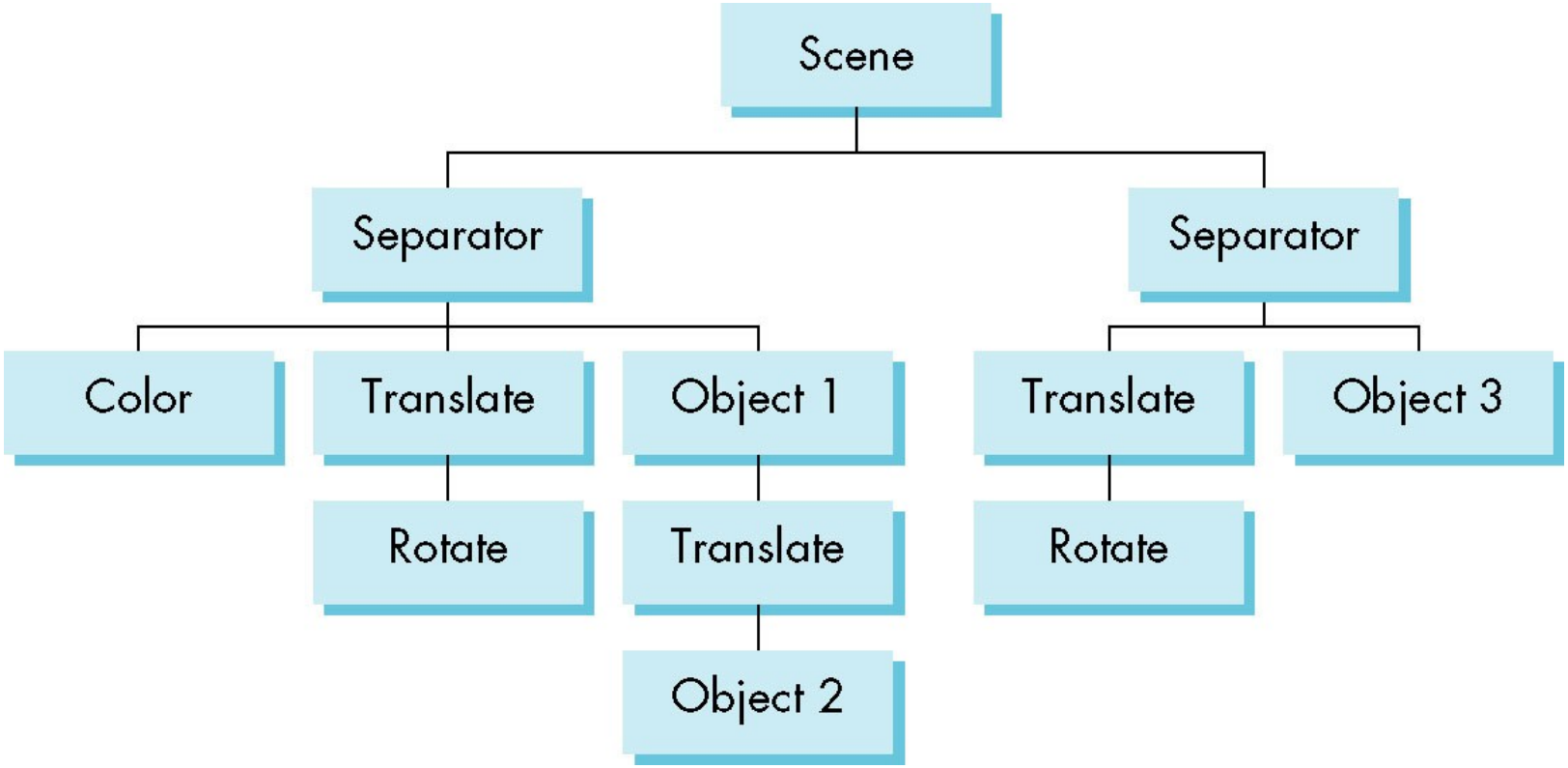
Light Object

```
class light {     // match Phong model
  public:
   boolean type; //ortho or perspective
   boolean near;
   float position[3];
   float orientation[3];
   float specular[3];
   float diffuse[3];
   float ambient[3];
}
```

Scene Descriptions

- If we recall figure model, we saw that
  - We could describe model either by tree or by equivalent code
  - We could write a generic traversal to display
- If we can represent all the elements of a scene (cameras, lights,materials, geometry) as C++ objects, we should be able to show them in a tree
  - Render scene by traversing this tree

Scene Graph


Scene
Separator
Separator
Color
Translate
Object 1
Translate
Object 3
Rotate
Translate
Rotate
Object 2

Preorder Traversal

```
glPushAttrib
glPushMatrix
glColor
glTranslate
glRotate
Object1
glTranslate
Object2
glPopMatrix
glPopAttrib
```

…

Group Nodes

- Necessary to isolate state chages
  - Equivalent to OpenGL Push/Pop
- Note that as with the figure model
  - We can write a universal traversal algorithm
  - The order of traversal can matter
    - If we do not use the group node, state changes can persist

Inventor and Java3D

- Inventor and Java3D provide a scene graph API
- Scene graphs can also be described by a file (text or binary)
  - Implementation independent way of transporting scenes
  - Supported by scene graph APIs
- However, primitives supported should match capabilities of graphics systems
  - Hence most scene graph APIs are built on top of OpenGL or DirectX (for PCs)

VRML

- Want to have a scene graph that can be used over the World Wide Web
- Need links to other sites to support distributed data bases
- Virtual Reality Markup Language
  - Based on Inventor data base
  - Implemented with OpenGL

Open Scene Graph

- Supports very complex geometries by adding occulusion culling in first path
- Supports translucently through a second pass that sorts the geometry
- First two passes yield a geometry list that is rendered by the pipeline in a third pass