Computer Graphics

Hierarchies of Objects

Modeling Complex Objects

- It is necessary to link or group pieces together into an object at times.
- An example of this would be a human limb. You rotate your shoulder and the arm rotates as well

How would you do this?

- **Shoulder**
  - **Translate, Rotate**
  - **push Transformation**
  - **Draw upper arm**
  - **Elbow**
    - **Rotate**
    - **push Transformation**
    - **Draw Lower Arm**
    - **Wrist**
      - **Rotate**
      - **push Transformation**
      - **Draw Hand**
      - **pop Tranformation**
    - **pop Transformation**
  - **pop Transformation**

Order is Important

- Note that we have arranged things so that the shoulder transformation stays in effect for the entire arm.
- The elbow transformation only affects the lower arm and hand
- The wrist transformation only affects the hand.
- We could continue on down to the individual fingers but you should get the idea.

A Simple Program To Demonstrate This

```
class Arm {
  private:
      double shoulderAngle, elbowAngle, wristAngle;
      unsigned int upperArm, lowerArm, hand;
  public:
      Arm( );
      void rotateShoulder ( double angle);
      void rotateElbow ( double angle);
      void rotateWrist ( double angle);
      void resetAngles( );
      void draw ( ) ;
};
```

```
Arm::Arm( ) {
      shoulderAngle = 0.0;
      elbowAngle = 0.0;
      wristAngle = 0.0;
  }
```

```
void Arm::draw ( )
  {
      glPushMatrix();
      /*
       * Moving the shoulder (rotation)
       */
            glRotated(shoulderAngle, 0.0, 0.0, 1.0);
            glColor3d(1.0, 0.0, 0.0);
            glBegin( GL_LINES );
                  glVertex3d( 0.0, 0.0, 0.0 );
                  glVertex3d( 0.0, 10.0, 0.0);
            glEnd( );
            glPushMatrix( );
```

```
      /* Moving the Elbow - the translation gets
       * the elbow out to the end of the "upper"
       * arm.
       */
            glTranslated ( 0.0, 10.0, 0.0 );
            glRotated(elbowAngle, 0.0, 0.0, 1.0);
            glColor3d(0.0, 1.0, 0.0);
            glBegin( GL_LINES );
                  glVertex3d(0.0, 0.0, 0.0);
                  glVertex3d(0.0, 7.0, 0.0);
            glEnd( );
            glPushMatrix( );
```

```
            /*
             * Moving the wrist -- The translation gets
             * the wrist out to the end of the lower arm.
             * The translates are cumulative so that the
             * hand is moved out 17 units on the y axis
             * (10 in the transform for the elbow and 7
             * more here)
             */
                  glTranslated ( 0.0, 7.0, 0.0 );
                  glRotated(wristAngle, 0.0, 0.0, 1.0);
                  glColor3d(0.0, 0.0, 1.0);
                  glBegin(GL_LINES);
                        glVertex3d(0.0, 0.0, 0.0);
                        glVertex3d(0.0, 3.5, 0.0);
                  glEnd( );
            glPopMatrix( );
      glPopMatrix( );
  glPopMatrix( );
}
```

```
void Arm::rotateShoulder( double angle ) {
  shoulderAngle += angle;
}

void Arm::rotateElbow( double angle ) {
  elbowAngle += angle;
}

void Arm::rotateWrist ( double angle ) {
  wristAngle += angle;
}
```

display( );

```
void display()
{
  // Display draws the arm with its hopefully
  revised position.
  glMatrixMode(GL_MODELVIEW);
  glClear(GL_COLOR_BUFFER_BIT |
  GL_DEPTH_BUFFER_BIT);

  glLoadIdentity();           // reset the transform
  glLineWidth(8.0);
  testObj.draw( );
  glFlush();
  return;
}
```

menuHandler( );

```
void menuHandler(int selection)
{
  // Handle the menu and update the angles
  // appropriately.  The step size for rotations
  // is fixed for my convenience.

  switch (selection) {
      case 1:
            testObj.rotateShoulder(-10.0);
            break;
      case 2:
            testObj.rotateShoulder(10.0);
            break;
      case 3:
            testObj.rotateElbow(-10.0);
            break;
```

```
        case 4:
                testObj.rotateElbow(10.0);
                break;
        case 5:
                testObj.rotateWrist(-10.0);
                break;
        case 6:
                testObj.rotateWrist(10.0);
                break;
        case 999:
                exit(0);
                break;
    }
    glutPostRedisplay();
    return;
  }
```

Let's build a System For Complex Objects

- What do I mean by a "complex object"
  - Multiple pieces (at least possible)
  - There are possible translations at each "joint" between sections
  - Different types of objects – not only predefined primitives and glut objects.
- The object is structured somewhat like a tree. A base object is the root, objects connected to the base object are the children, etc…

Using the STL

- STL – Standard Template Library
- Has common data structures implemented as templates and able to be used.
  - List, Stack, Queue, Vector, Deque
  - Set, Map, Multiset, Hash
- The basic operations are there and available.

Our Base Class

- Needs to allow transformations
- Should allow for a draw routine
- Should allow at least some attributes to be set.
- Should allow attached objects to be added (children)
- Should maintain state (pop/push)

Version 0.0

```
#ifndef __HIERARCHICALOBJECT_H__
#define __HIERARCHICALOBJECT_H__
#include "glut.h"
#include <list>    // STL list to hold subobjects
using namespace std;
class HierarchicalObject;

class HierarchicalObject {
protected:

public:
   HierarchicalObject();
   void translate ( double dx, double dy, double dz);
   void rotate( double angle, double vx, double vy, double vz);
   void scale ( double sx, double sy, double sz);
   virtual void draw() = 0;
   addSubObject(HierarchicalObject *obj);
};

#endif
```

Needed Yet..

- Routines for non-virtual functions
- Protected instance variables
- Attributes…

Protected Variables

```
double currentTransform[16];
list<hierarchicalObject *> subObjectList;
```

- The currentTransform will hold the current transformation (the result of all of the transforms on the current object) – We look at it as a 4x4 matrix but OpenGL stores it as a 16 element vector.
- subObjectList is a possibly empty list of objects that are connected to this object. I'm using pointers in this case to limit the problems with copying objects.

Constructor

```
HierarchicalObject::HierarchicalObject() {
  /* Sets the initial transform to the identity matrix.
   * In a finished version it would also initialize
   * any attributes.
   */
  int i;
  for (i = 0; i < 16; i++) {
      currentTransform[i] = 0.0;
  }
  for (i = 0; i < 16; i=i+5) {
      currentTransform[i] = 1.0;
  }
}
```

translate

```
void
HierarchicalObject::translate(double dx, double dy, double dz) {
  GLint savedMode;
  glGetIntegerv(GL_MATRIX_MODE, &savedMode);
  glMatrixMode(GL_MODELVIEW_MODE);
  glPushMatrix();                 // save current state
  glLoadIdentity();
  glMultMatrixd(currentTransform);// apply the current transform
  glTranslated(dx, dy, dz);       // translate
  glGetDoublev(GL_MODELVIEW_MATRIX, currentTransform);
  glPopMatrix();                  // restore the state
  glMatrixMode(savedMode);
}
```

scale

```
void
HierarchicalObject::scale(double sx, double sy, double sz) {
  GLint savedMode;
  glGetIntegerv(GL_MATRIX_MODE, &savedMode);
  glMatrixMode(GL_MODELVIEW_MODE);
  glPushMatrix();                 // save current state
  glLoadIdentity();
  glMultMatrixd(currentTransform);// apply the current transform
  glScaled(sx, sy, sz);           // scale!
  glGetDoublev(GL_MODELVIEW_MATRIX, currentTransform);
  glPopMatrix();                  // restore the state
  glMatrixMode(savedMode);
}
```

rotate

```
void
HierarchicalObject::rotate(double angle, double vx, double vy,
  double vz) {
  GLint savedMode;
  glGetIntegerv(GL_MATRIX_MODE, &savedMode);
  glMatrixMode(GL_MODELVIEW_MODE);
  glPushMatrix();                 // save current state
  glLoadIdentity();
  glMultMatrixd(currentTransform);// apply the current transform
  glRotated(angle, vx, vy, vz);   // rotate around axis
  glGetDoublev(GL_MODELVIEW_MATRIX, currentTransform);
  glPopMatrix();                  // restore the state
  glMatrixMode(savedMode);
}
```

addSubObject

```
void HierarchicalObject::addSubObject(HierarchicalObject *obj) {
  subObjectList.push_back(obj); // adds the object to the
                                // end of the list.
}
```

Now, lets inherit the class

```
Class lampBase: public HierarchicalObject {
   public:
        void draw();
};

Void lampBase::draw()
{
   double angle;
   list<HierarchicalObject *>::iterator current;

   glPushMatrix();
   glPushAttrib(GL_ALL_ATTRIB_BITS);
   glMatrixd(currentTransformation);
   glColor3f(1.0, 1.0, 1.0);
   glBegin(GL_TRIANGLE_STRIP);
   for (angle = 0; angle < 2.0 * pi; angle += pi / 9) {
        glVertex3f(5.0*sin(angle), 0.0, 5.0 * cos(angle));
        glVertex3f(5.0*sin(angle), 0.5, 5.0 * cos(angle));
   }
   glEnd();
```

```
  glPushMatrix();
  glTranslatef(0.0, 0.5, 0.0);
  glutSolidSphere(0.5, 12, 12);
  glPopMatrix();
  glPopAttrib();
  if (subObjectList.size() > 0) {// Are there sub objects to draw…
      for (current = subObjectList.begin();
             current != subObjectList.end();
             current++) {
             (*current)->draw();
      }
  }
  glPopMatrix();
  return;
}
```

Why Trees of Objects?

- Often you have objects whose motion is linked together.
- Think of your arm – If you move your upper arm the hand moves too!

Why build off the Hierarchy?

- Prototyping speed – we don't have to make a lot of explicit connections.
- Generalization – we have gotten rid of special cases…

Why not use the Hierarchy

- Speed – I'm trying to be safe and sometimes that will affect speed.
- It may be tough to do many things – hard to plan for everything.
- Size – This isn't going to be small when talking about either code size or executable size.

What's posted

- What we are going to post is a version of this based off of last years.
- Corrections include a few more checks and balances.
- Has some attributes built in.
- Going to animate the infamous lamp.

Additions to Classes

- Among the additions to the existing classes we should add:
  - Materials (so that objects can be shaded appropriately)
  - Polygonal objects – objects made up of polygons for more complex objects.
  - Display lists to hold the items