Using GLU/GLUT Objects

GLU/GLUT provides very simple object primitives:

- `glutWireCube`
- `glutWireCone`
- `gluCylinder`
- `glutWireTeapot`

Each glu/glut object has its default size, position, and orientation.
You need to perform modeling transformation to make it right for you.

- `glutWireCube(1.0)` - 'wire' means wire frame
  - Put a 1x1x1 cube with its center at world (0,0,0)

To create a 2 x 0.1 x 2 table top, need to call `glScalef(2, 0.1, 2)` before you call `glutWireCube(1.0)`

---

`gluCylinder()`

Three steps to create a cylinder:

1. Create a GLU quadric object
   ```
   GLUquadricObj *p = gluNewQuadric();
   ```
2. Set to wire frame mode
   ```
   gluQuadricDrawStyle(p, GLU_LINE);
   ```
3. Derive a cylinder object from p
   ```
   gluCylinder(p, base, top, height, slice, stacks)
   ```

The default position is also with base at z = 0 plane

---

`glutWireCone()`

Use `glutWireCone` and `gluCylinder` to make a lamp.

- A polygon approximation of a cone.

**Default position**: its base at Z = 0 plane
- base: the width of its base
- height: the height of the cone
- slices: the number of vertical lines used to make up the cone
- stacks: the number of horizontal lines used to make up the cone
glutWireTeapot()

- The famous Utah Teapot has become an unofficial computer graphics mascot

```
  glutWireTeapot(0.5) - -
  Create a teapot with size 0.5, and position its center at (0,0,0)
```

Again, you need to apply transformations to position it at the right spot

Transformations

- Two ways to specify transformations
  1. Each part of the object is transformed independently relative to the origin
     Not the OpenGL Way!

```
  Translate the base by (5,0,0);
  Translate the lower arm by (5,0,0);
  Translate the upper arm by (5,0,0);
```

Relative Transformation

A better (and easier) way:
(2) Relative transformation: Specify the transformation for each object relative to its parent

Object Dependency

- A graphical scene often consists of many small objects
- The attributes of an object (positions, orientations) can depend on others

A Robot Hammer!
Hierarchical Representation - Scene Graph

- We can describe the object dependency using a tree structure

Hierarchical Structure:

- Root node
- Lower arm
- Upper arm
- Hammer

The position and orientation of an object can be affected by its parent, grand-parent, grand-grand-parent, ... nodes.

This hierarchical representation is referred to as Scene Graph.

Relative Transformation

Relative transformation: Specify the transformation for each object relative to its parent

Step 1: Translate base and its descendants by (5,0,0);

Step 2: Rotate the lower arm and all its descendants relative to its local y axis by -90 degree

Represent relative transformations using scene graph
Do it in OpenGL

- Translate base and all its descendants by (5,0,0)
- Rotate the lower arm and its descendants by -90 degree about the local y

```c
glMatrixMode(GL_MODELVIEW);
gLoadIdentity();
... // setup your camera
glTranslatef(5,0,0);
Draw_base();
gRotatef(-90, 0, 1, 0);
Draw_lower_arm();
Draw_upper_arm();
Draw_hammer();
```

A more complicated example

- How about this model?

```
base
  ↘
  Lower arm
  ↘
  Upper arm
  ↘
  Hammer

(left hammer) (right hammer)
```

Depth-first traversal

- Program this transformation by depth-first traversal

```
base
  ↘
  Lower arm
  ↘
  Upper arm
  ↘
  Hammer
  ↘
  Lower arm
  ↘
  Upper arm
  ↘
  Hammer

(left hammer) (right hammer)
```

Do this ...

- Base and everything - translate (5,0,0)
- Left hammer - rotate 75 degree about the local y
- Right hammer - rotate -75 degree about the local y
How about this?

Something is wrong ...

What's wrong? - We want to transform the right hammer relative to the base, not to the left hammer

We should undo the left hammer transformation before we transform the right hammer

Undo the previous transformation(s)

Need to save the modelview matrix right after we draw base

OpenGL Matrix Stack

We can use OpenGL Matrix Stack to perform matrix save and restore
Push and Pop Matrix Stack

- A simple OpenGL routine:
  ```
  glTranslate(5, 0, 0)
  Draw_base();
  glPushMatrix();
  glRotate(75, 0, 1, 0);
  Draw_left_hammer();
  glPopMatrix();
  glRotate(-75, 0, 1, 0);
  Draw_right_hammer();
  ```

Push and Pop Matrix Stack

- Nested push and pop operations

<table>
<thead>
<tr>
<th>Modelview matrix (M)</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>M = I</code></td>
<td></td>
</tr>
<tr>
<td><code>M = M1</code></td>
<td><code>M2</code></td>
</tr>
<tr>
<td><code>M = M1 x M2</code></td>
<td><code>M3</code></td>
</tr>
<tr>
<td><code>M = M1 x M2 x M3</code></td>
<td><code>M4</code></td>
</tr>
<tr>
<td><code>M = M1 x M2 x M3 x M4</code></td>
<td><code>M5</code></td>
</tr>
<tr>
<td><code>M = M1 x M2 x M3 x M4 x M5</code></td>
<td><code>M1 x M2</code></td>
</tr>
<tr>
<td><code>M = M1 x M2 x M3 x M4 x M5</code></td>
<td><code>M1 x M2</code></td>
</tr>
<tr>
<td><code>M = M1 x M2 x M3 x M4</code></td>
<td><code>M5</code></td>
</tr>
<tr>
<td><code>M = M1 x M2 x M3</code></td>
<td><code>M4</code></td>
</tr>
<tr>
<td><code>M = M1 x M2</code></td>
<td><code>M3</code></td>
</tr>
<tr>
<td><code>M = M1</code></td>
<td><code>M2</code></td>
</tr>
<tr>
<td><code>M = I</code></td>
<td></td>
</tr>
</tbody>
</table>

Depth First Traversal