Using GLU/GLUT Objects

- GLU/GLUT provides very simple object primitives:
  - `glutWireCube`
  - `glutWireCone`
  - `gluCylinder`
  - `glutWireTeapot`
GLU/GLUT Objects

- 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)`
Three steps to create a cylinder

1. Create a GLU quadric object
   \[ \text{GLUquadricObj} \ *p = \text{gluNewQuadric}(); \]
2. Set to wire frame mode
   \[ \text{gluQuadricDrawStyle}(	ext{GLU__LINE}); \]
3. Derive a cylinder object from \( p \)
   \[ \text{gluCylinder}(p, \text{base}, \text{top}, \text{height}, \text{slice}, \text{stacks}) \]

The default position is also with base at \( z = 0 \) plane
Use `glutWireCone` and `gluCylinder` to make a lamp

```cpp
glutWireCone(base, height, slices, stacks)
```

- 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);
...

x
y
z
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!

upper arm

lower arm

base

hammer
We can describe the object dependency using a tree structure.

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);
Relative Transformation (2)

Step 2: Rotate the lower arm and all its descendants relative to its local y axis by -90 degree
Relative Transformation (3)

- Represent relative transformations using scene graph

```
Base
  └── Lower arm
    └── Upper arm
        └── Hammer
```

- Translate (5,0,0)
- Rotate (-90) about its local y
- Apply all the way down
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);
glLoadIdentity();
... // setup your camera

glTranslatef(5,0,0);
Draw_base();

glRotatef(-90, 0, 1, 0);
Draw_lower_arm();

Draw_upper_arm();

Draw_hammer();
```
A more complicated example

- How about this model?

Scene Graph?

- base
  - Lower arm
    - Upper arm
      - Hammer
      - (left hammer)
  - Lower arm
    - Upper arm
      - 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
Depth-first traversal

- Program this transformation by depth-first traversal

Depth First Traversal

- Do ____ transformation(s)
  - Draw base
  - Do ____ transformation(s)
  - Draw left arm
  - Do ____ transformation(s)
  - Draw right arm

What are they?
How about this?

- base
  - Lower arm
    - Upper arm
      - Hammer (left hammer)
  - Lower arm
    - Upper arm
      - Hammer (right hammer)

Translate(5,0,0)
Draw base
Rotate(75, 0, 1, 0)
Draw left hammer
What’s wrong?!
Rotate(-75, 0, 1, 0)
Draw right hammer
Something is wrong ...

What’s wrong? – We want to transform the right hammer relative to the base, not to the left hammer.

How about this?

Do **Translate(5,0,0)**

Draw base

Do **Rotate(75, 0, 1, 0)**

Draw left hammer

Do **Rotate(-75, 0, 1, 0)**

Draw right hammer

What’s wrong?!

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

Need to undo this first.
Undo the previous transformation(s)

- Need to save the modelview matrix right after we draw base

```
Initial modelView M

Translate(5,0,0) -> M = M x T
```

- Draw base

```
Rotate(75, 0, 1, 0)
```

- Draw left hammer

```
Rotate(-75, 0, 1, 0)
```

- Draw right hammer

Undo the previous transformation means we want to restore the Modelview Matrix M to what it was here

i.e., save M right here

... And then restore the saved Modelview Matrix
OpenGL Matrix Stack

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

Initial modelView $M$

Do $\text{Translate}(5, 0, 0) \rightarrow M = M \times T$

Draw base

Do $\text{Rotate}(75, 0, 1, 0)$

Draw left hammer

Do $\text{Rotate}(-75, 0, 1, 0)$

Draw right hammer

* Store the current modelview matrix
  - Make a copy of the current matrix and **push** into OpenGL Matrix Stack
  - Call `glPushMatrix()`

- continue to modify the current matrix

* Restore the saved Matrix
  - **Pop** the top of the Matrix and copy it back to the current Modelview Matrix:
  - Call `glPopMatrix()`
Push and Pop Matrix Stack

- A simple OpenGL routine:

```c
glTranslate(5, 0, 0)
Draw_base();
glPushMatrix();
glRotate(75, 0, 1, 0);
Draw_left_hammer();
glPopMatrix();
glRotate(-75, 0, 1, 0); Draw_right_hammer();
glPopMatrix();
```

Depth First Traversal
Push and Pop Matrix Stack

- Nested push and pop operations

```c
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
... // Transform using M1;
... // Transform using M2;
glPushMatrix();
... // Transform using M3
glPushMatrix();
... // Transform using M4
glPopMatrix();
... // Transform using M5
...
glPopMatrix();
```

<table>
<thead>
<tr>
<th>Modelview matrix (M)</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>M = I</td>
<td></td>
</tr>
<tr>
<td>M = M1</td>
<td></td>
</tr>
<tr>
<td>M = M1 x M2</td>
<td>M1xM2</td>
</tr>
<tr>
<td>M = M1 x M2 x M3</td>
<td>M1xM2xM3</td>
</tr>
<tr>
<td>M = M1 x M2 x M3 x M4</td>
<td></td>
</tr>
<tr>
<td>M = M1 x M2 x M3 x M5</td>
<td></td>
</tr>
<tr>
<td>M = M1 x M2</td>
<td></td>
</tr>
<tr>
<td>M = M1 x M2</td>
<td></td>
</tr>
</tbody>
</table>