Transformations

Two possible ways to specify transformations

- Each part of the object is transformed independently relative to the origin
  
  - Not convenient!

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

y

x

z
Hierarchical Transformation

A better (and easier) way:

Hierarchical (relative) transformation: Specify the transformation for each object relative to its parent (or the current local reference frame)
Object Dependency

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

A Robotic Hammer!
Hierarchical Representation - Scene Graph

- 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**
Hierarchical Transformation

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

Step 1: Translate the 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
Hierarchical Transformation(3)

- Represent hierarchical transformations using scene graph

- Base
  - Translate (5,0,0)
  - Upper arm
    - Rotate (-90) about its local y
    - Apply all the way down
      - Hammer

- Apply all the way down
Incremental update of the modeling transformation matrix

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

Initialize your matrix to identity I

\[ M_1 \leftarrow \text{Translate}(5, 0, 0) \]
\[ M = M \times M_1 \]

\[ \text{Draw_base();} \quad // \text{apply } M \text{ to the vertices} \]

\[ M_2 = \text{Rotate}(-90, 0, 1, 0) \]
\[ M = M \times M_2 \quad // \text{accumulate } M_2 \text{ to } M \]

\[ \text{Draw_lower_arm();} \]
\[ \text{Draw_upper_arm();} \]
\[ \text{Draw_hammer();} \]
OpenGL Fixed Function Pipeline

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

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

... Base
    ↓
    Lower arm
        ↓
        Upper arm
            ↓
            Hammer
```

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

glTranslatef(5,0,0);

draw_base();

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

draw_upper_arm();

draw_lower_arm();

draw_hammer();
```
A more complicated example

- How about this model?

Scene Graph?

```
base
  /\  
Lower arm
   /\  
Upper arm
    /\  
Hammer
```

(base)

(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
Depth-first traversal

- Program this transformation by depth-first traversal

Do _____ transformation(s)
Draw base
Do _____ transformation(s)
Draw left arm
Do _____ transformation(s)
Draw right arm

What are they?

Depth First Traversal
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
- **Rotate**(-75, 0, 1, 0)
- Draw right hammer

What’s wrong?!
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

Need to undo this first

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.
Undo the previous transformation(s)

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

  Initial modelView $M$

  **Translate**$(5,0,0)$ $\Rightarrow M = M \times T$

  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
Matrix Stack

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

Initial modelView $M$

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

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 a Matrix Stack
  - continue to modify the current matrix

* Restore the saved Matrix
  - Pop the top of the matrix stack and copy it back to the current Modelview Matrix:
OpenGL fixed function pipeline has a 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

$\text{glPushMatrix}()$

$\text{glPopMatrix}()$

$\text{glPopMatrix}()$
Push and Pop Matrix Stack

- Nested push and pop operations

```
... // Transform using M1;
... // Transform using M2;
Push Matrix();
... // Transform using M3
Push Matrix();
... // Transform using M4
Pop Matrix();
...
... // Transform using M5
...
Pop Matrix();
```

```
M = M1
M = M1 x M2
M = M1 x M2 x M3
M = M1 x M2 x M3 x M4
M = M1 x M2 x M3 x M5
M = M1 x M2
```
Push and Pop Matrix Stack

- Nested push and pop operations

```c
// Push operations
glMatrixMode(GL_MODELVIEW);
MatrixXd identity();
... // Transform using M1;
... // Transform using M2;
MatrixXd pushMatrix();
... // Transform using M3
MatrixXd pushMatrix();
.. // Transform using M4
MatrixXd popMatrix();
... // Transform using M5
...
MatrixXd popMatrix();
```

<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>M1xM2</td>
</tr>
<tr>
<td>M = M1 x M2</td>
<td>M1xM2xM3</td>
</tr>
<tr>
<td>M = M1 x M2 x M3</td>
<td>M1 x M2</td>
</tr>
<tr>
<td>M = M1 x M2 x M3 x M4</td>
<td></td>
</tr>
<tr>
<td>M = M1 x M2 x M3</td>
<td></td>
</tr>
<tr>
<td>M = M1 x M2 x M3 x M5</td>
<td>M1 x M2</td>
</tr>
<tr>
<td>M = M1 x M2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Push and Pop Matrix Stack

- A simple OpenGL routine:

```c
glTranslate(5,0,0)
Draw_base();
glPushMatrix();
push

Lower arm
Upper arm
Hammer
(left hammer)
Depth First Traversal

pop

Lower arm
Upper arm
Hammer
(right hammer)
glPopMatrix();
glRotate(-75, 0,1,0);
Draw_right_hammer();
```

Depth First Traversal
Creating your own objects

- Create simple geometry primitives

- Cone
- Cylinder
- Teapot
- Cube
Objects in local coordinates

- It is more convenient to create all objects with a default size, position, and orientation, i.e., defined in their local coordinate systems.
- Then you can perform modeling transformation to make it right for you.

![Cube(1.0)](image)

Put a 1x1x1 cube with its center at world (0,0,0)

To create a 2 x 0.1 x 2 table top - you can scale (2, 0.1, 2) before you draw the Cube(1.0)
Cylinder()

- Three steps to create a cylinder

Cylinder(base, top, height, slice, stacks)

- Base
  - yz base radius
- Top
  - yz top radius
- Height
- Num. of vertical lines
- Num. of horizontal lines

The default position is with base at z = 0 plane
Cone(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
- stace: the number of horizontal lines used to make up the cone
Teapot, Bunny etc

- There are many famous geometric models on the web that you can use. For example, Utah Teapot, Stanford Bunny etc
- Read the PLY help in our class web site for more information