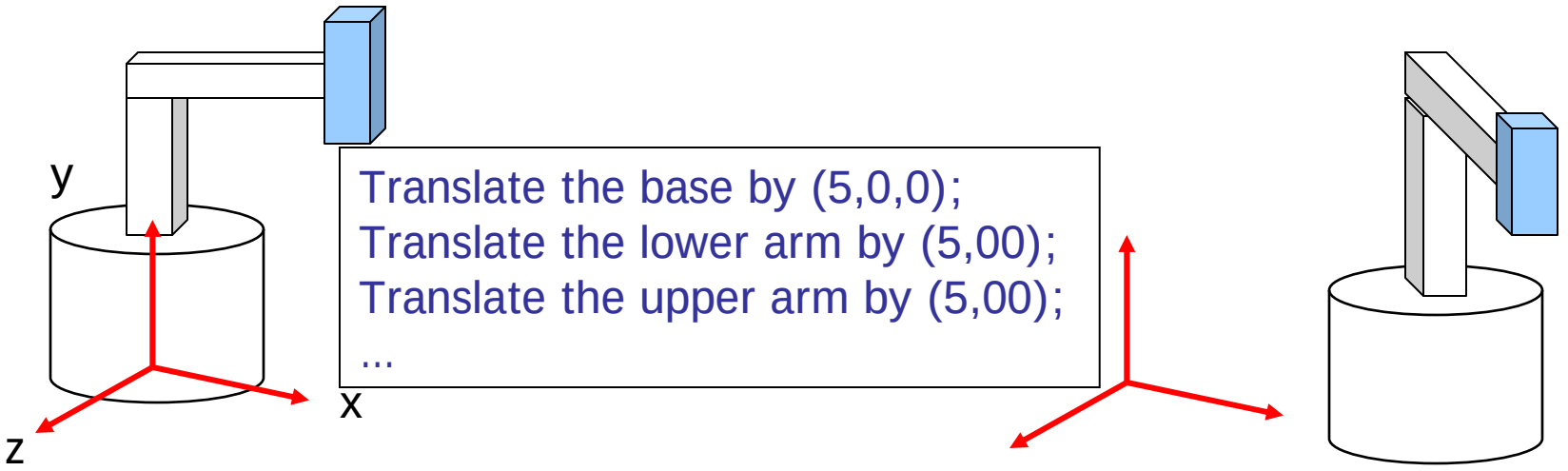
Transformations

- Two ways to specify transformations
  - (1) Each part of the object is transformed independently relative to the origin

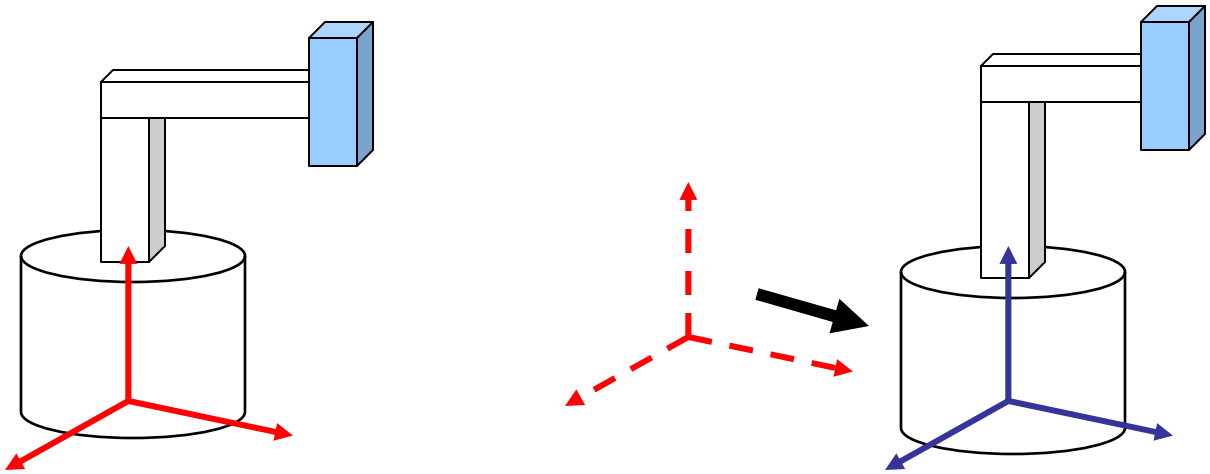
Not the OpenGL Way!



Relative Transformation

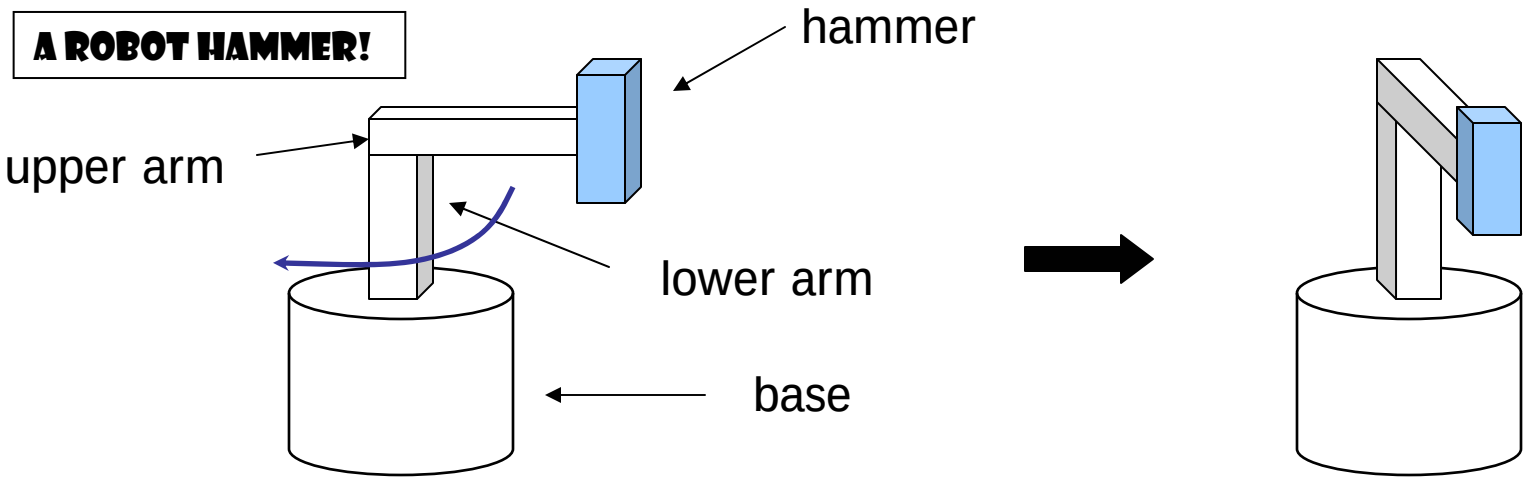
A better (and easier) way:

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



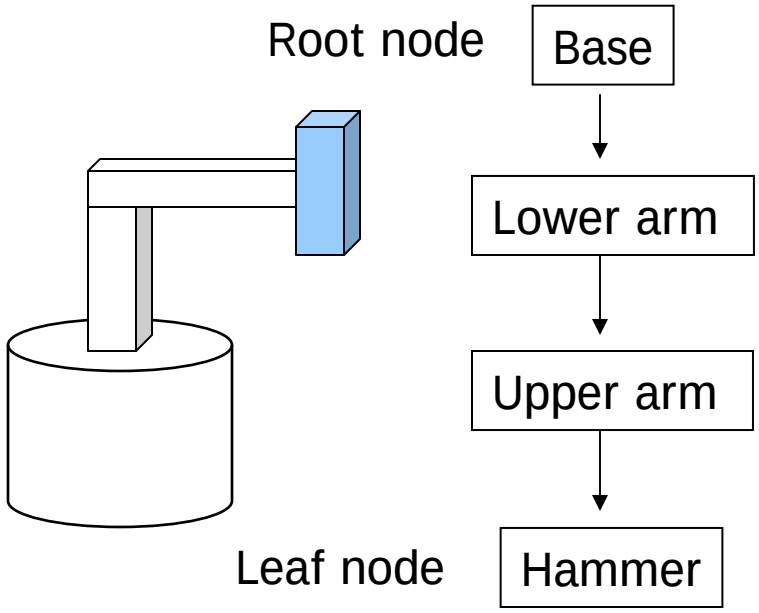
Object Dependency

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



Hierarchical Representation - Scene Graph

- We can describe the object dependency using a tree structure

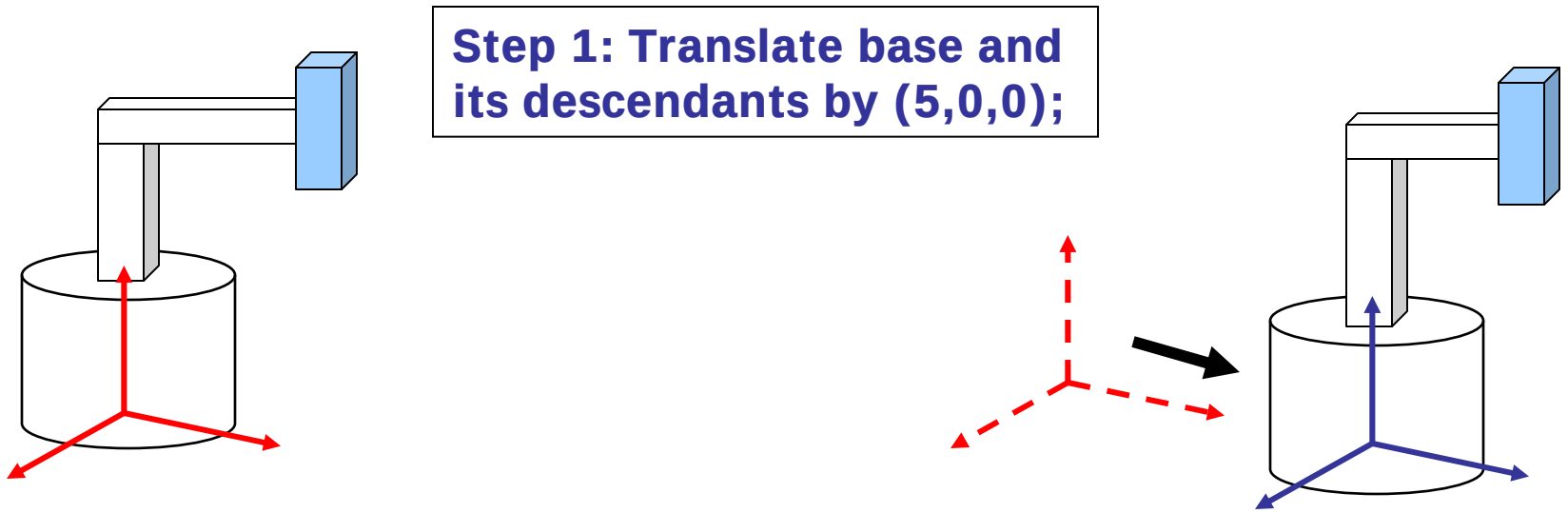


The position and orientation of an object can be affected by its parent, grand-parent, grand-grand-parent ... nodes

This hierarchical representation is referred to as Scene Graph

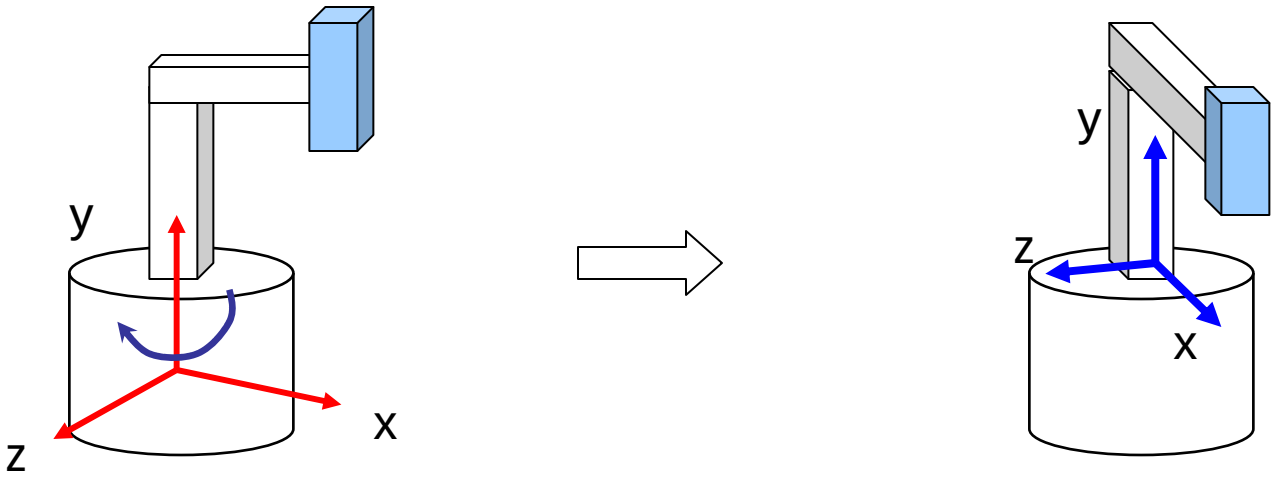
Relative Transformation

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



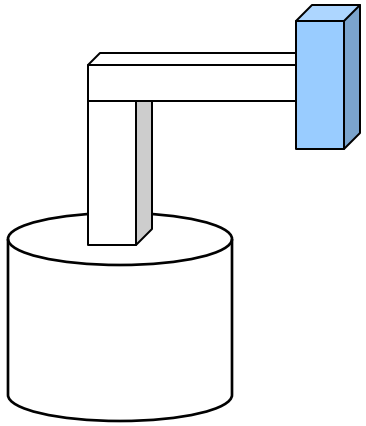
Relative Transformation (2)

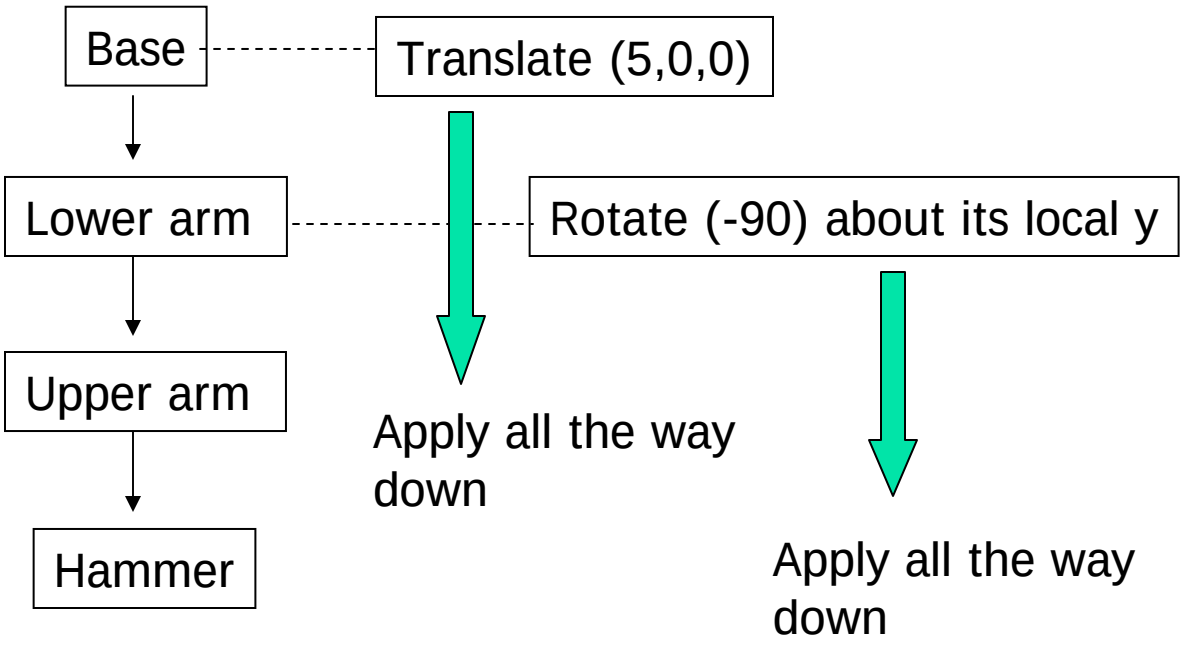
Step 2: Rotate the lower arm and all its descendants relative to its local y axis by -90 degree



Relative Transformation (3)

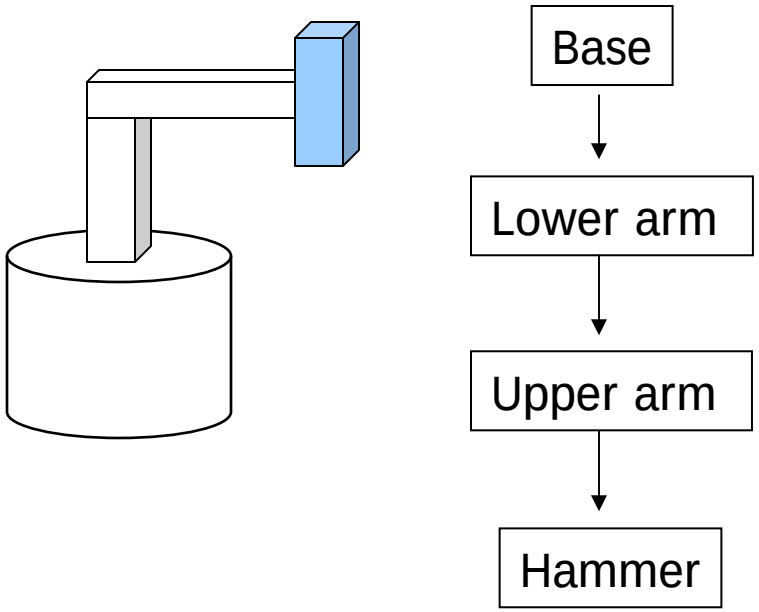
- Represent relative transformations using scene graph





Do it in OpenGL

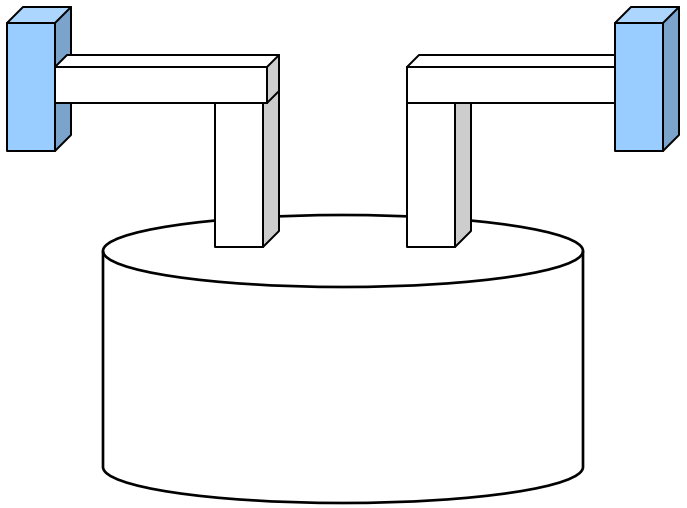
- Translate base and all its descendants by (5,0,0)
- Rotate the lower arm and its descendants by -90 degree about the local y



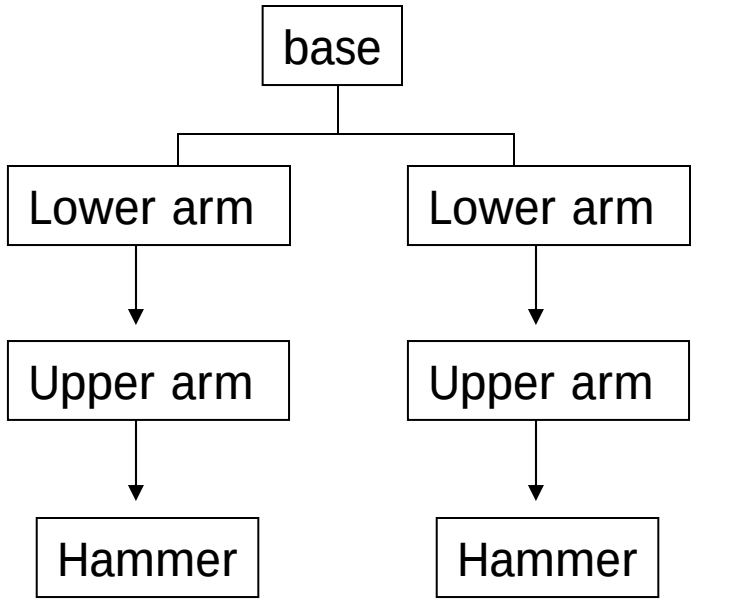

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

... // 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?

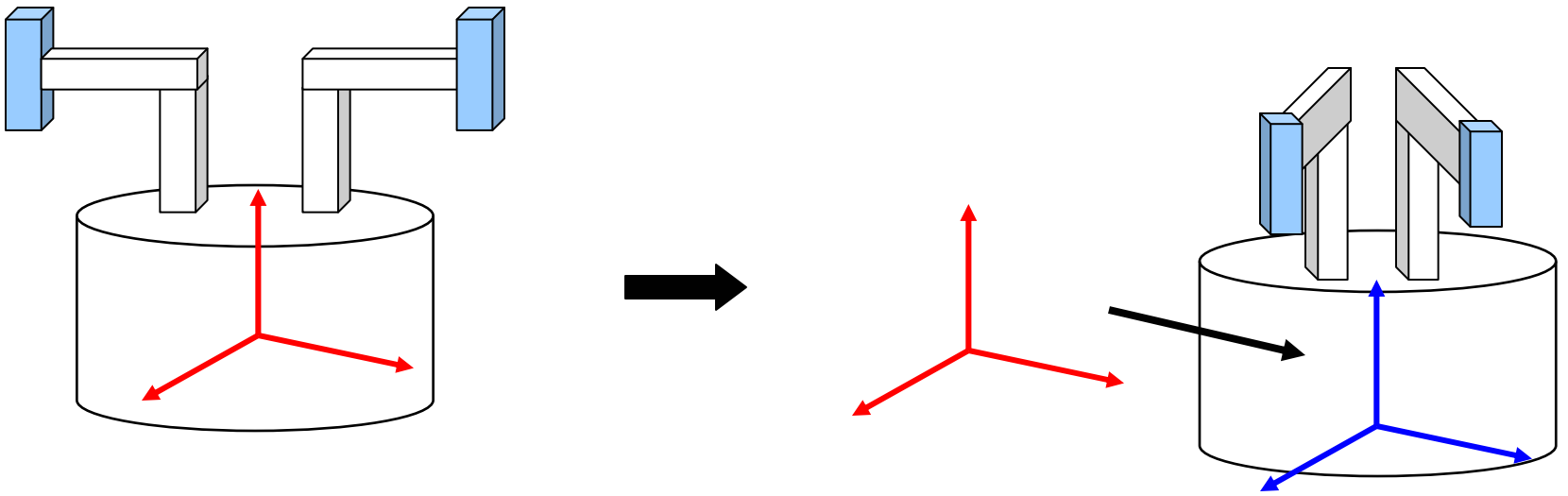


Scene Graph?



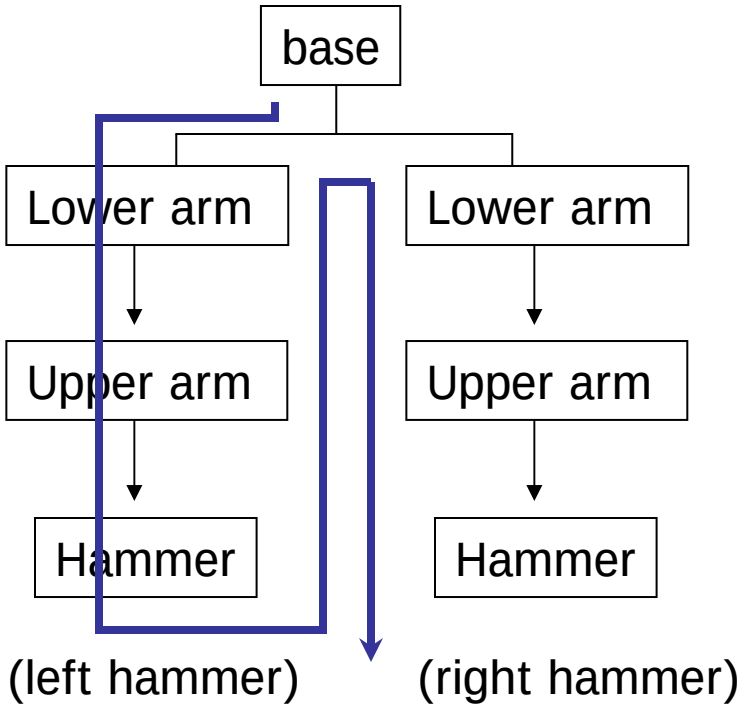
Do this ...

- 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)

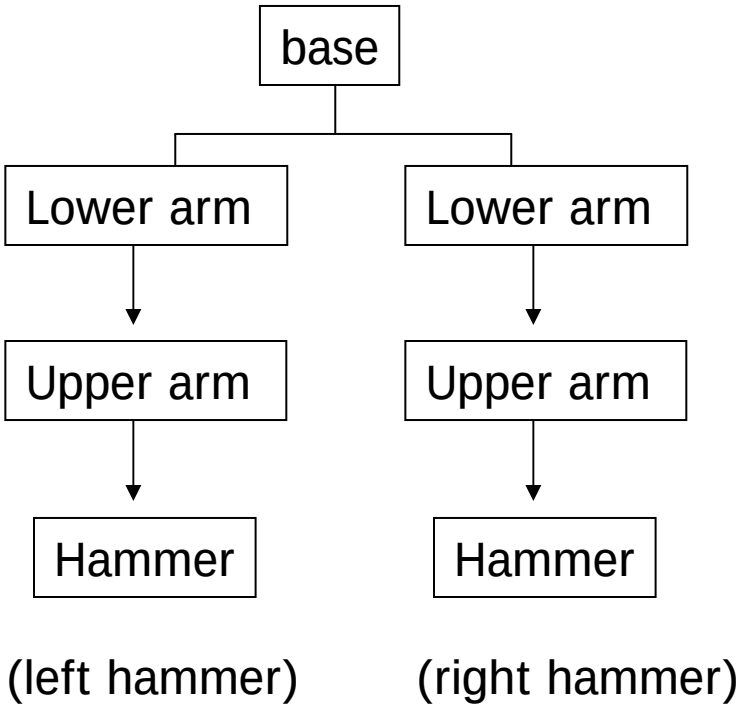
Draw left arm

Do _____ transformation(s)

Draw right arm

What are they?

How about this?



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

What's wrong?!

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

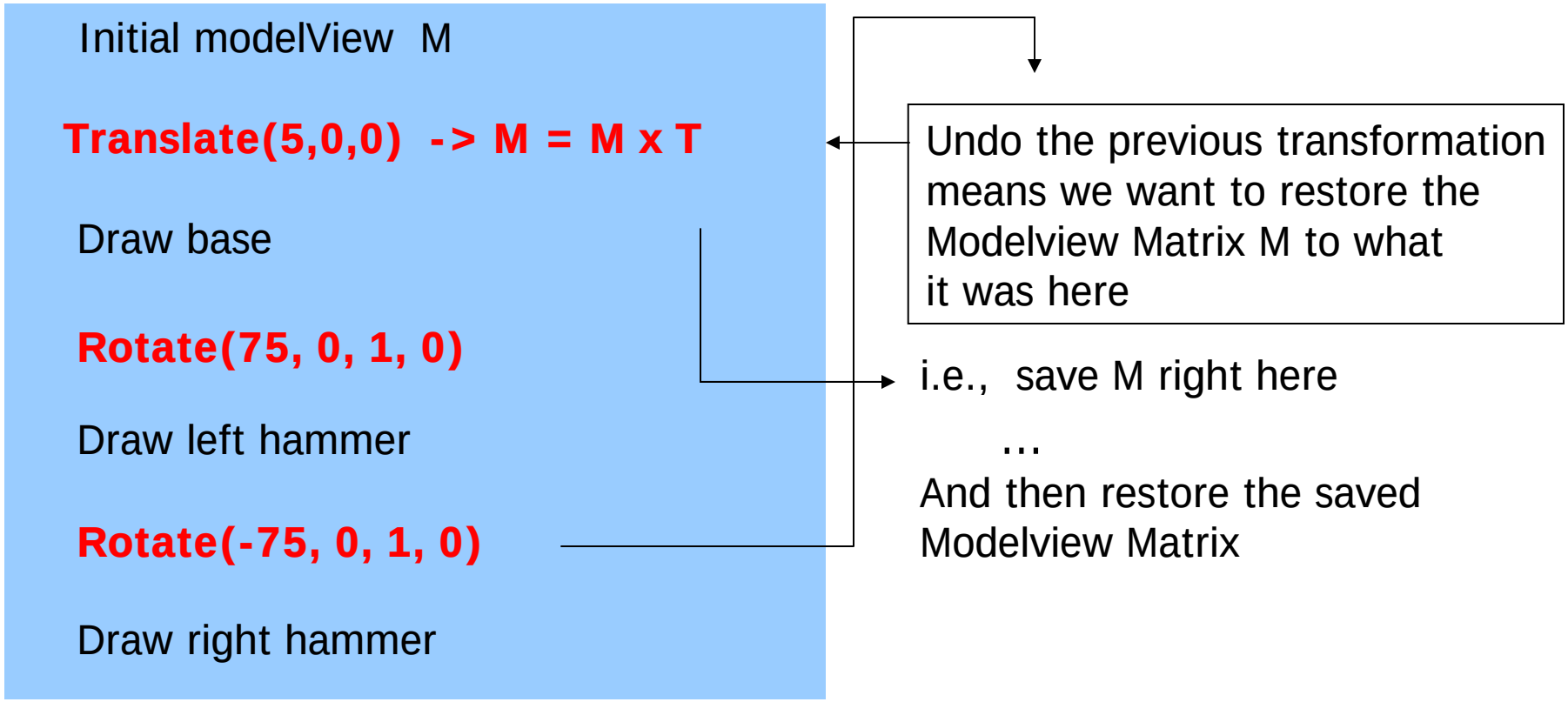
Draw right hammer

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



OpenGL Matrix Stack

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

Initial modelView M

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

Draw base

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

Draw left hammer

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

Draw right hammer

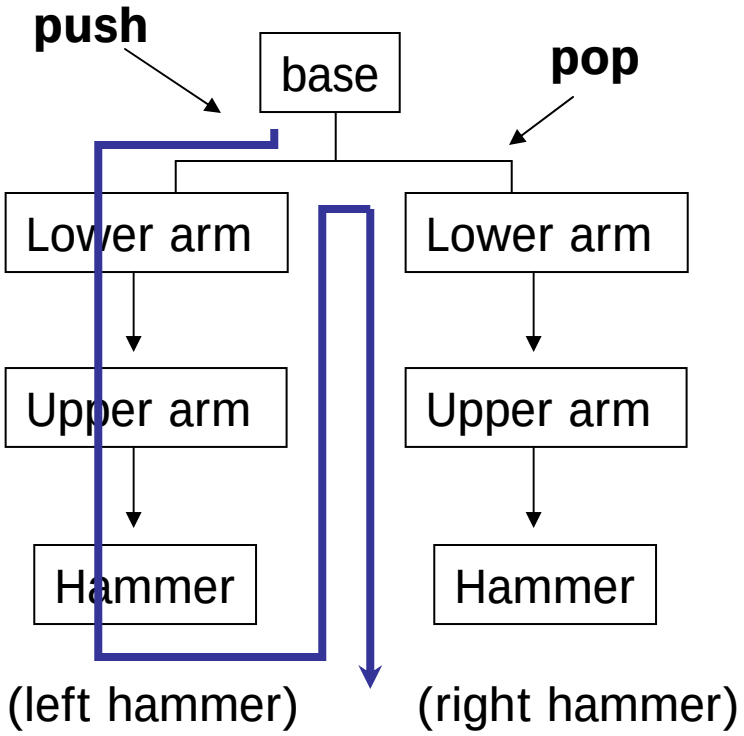
* Store the current modelview matrix
- Make a copy of the current matrix and **push** into OpenGL Matrix Stack: call glPushMatrix()

- 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:



Depth First Traversal

```
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