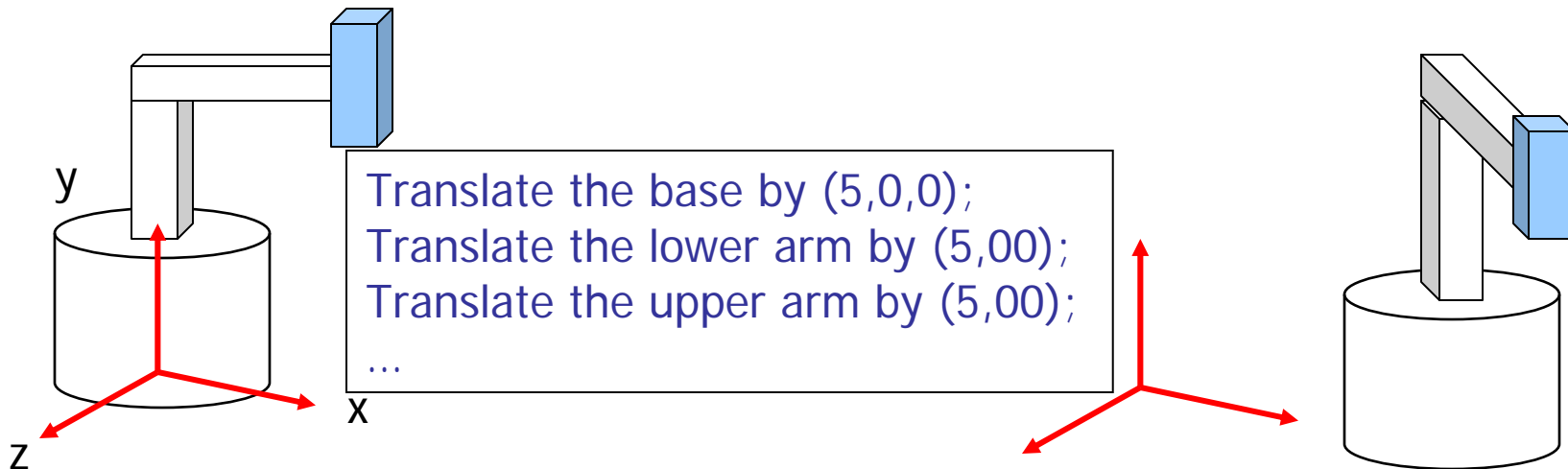


# Transformations

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

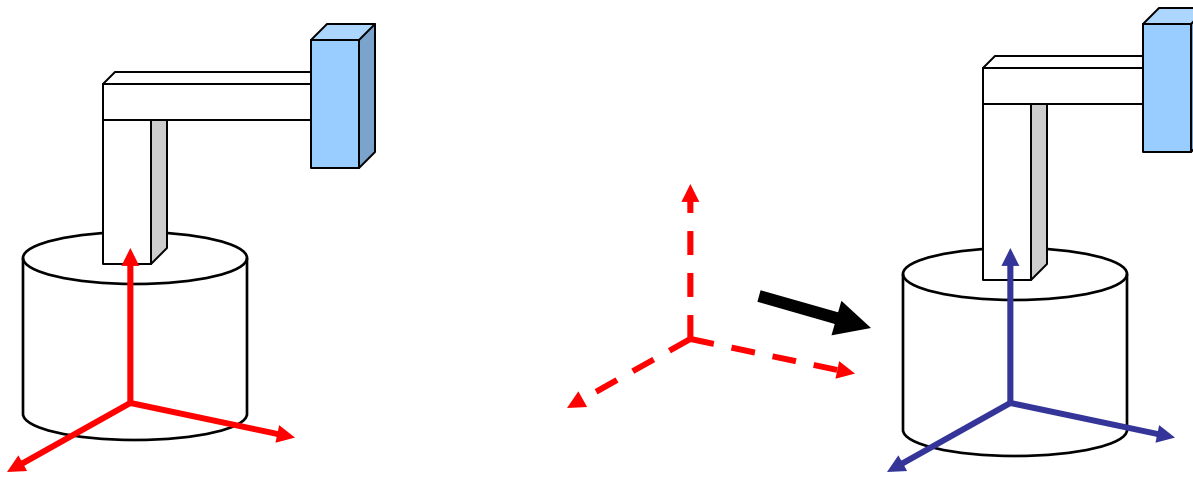
Not the OpenGL Way!



# Relative Transformation

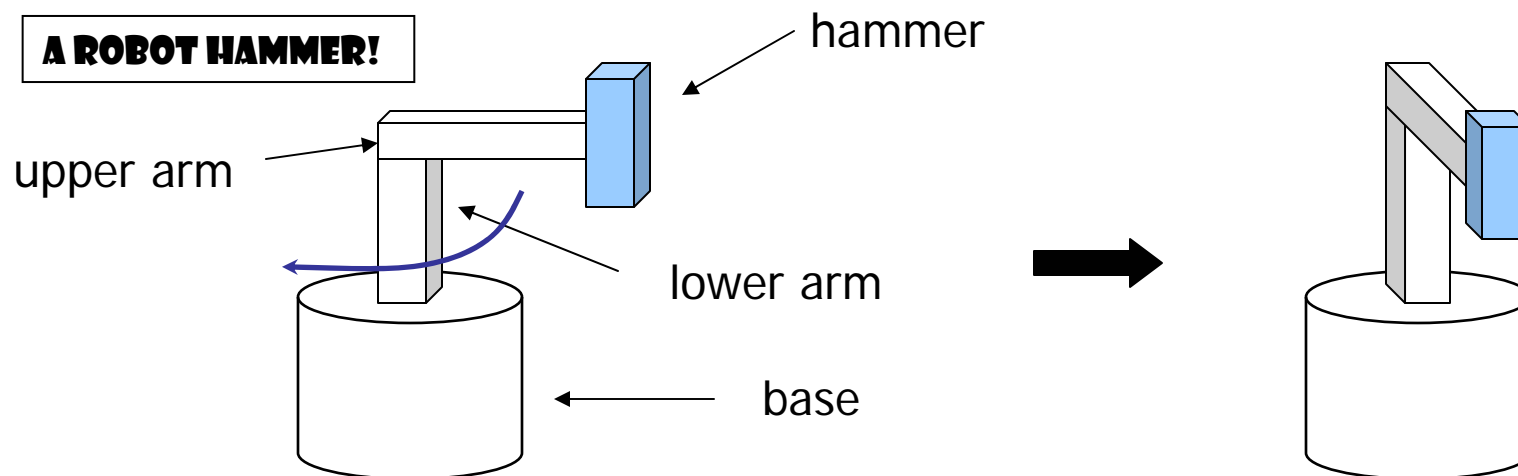
A better (and easier) way:

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



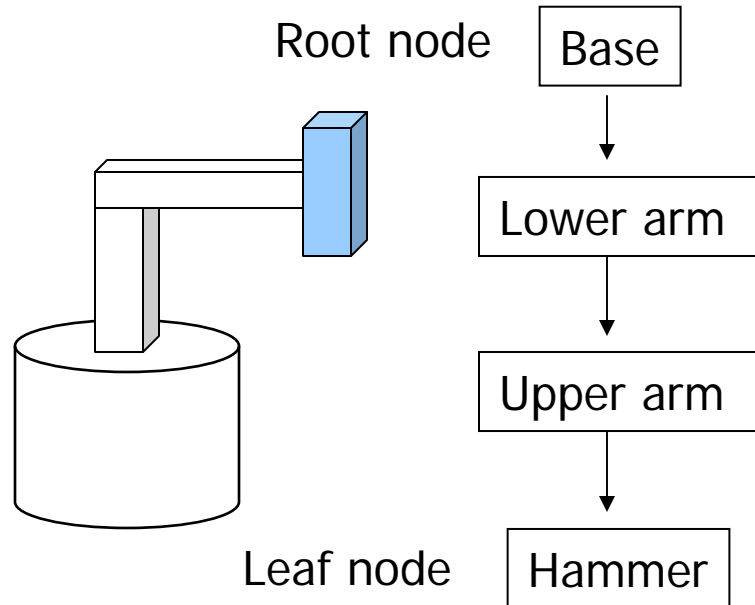
# Object Dependency

- 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

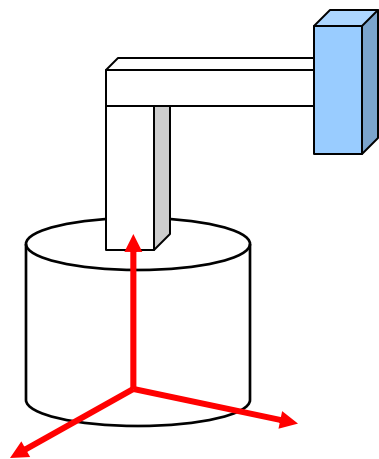


The position and orientation of an object can be affected by its parent, grand-parent, grand-grand-parent ... nodes

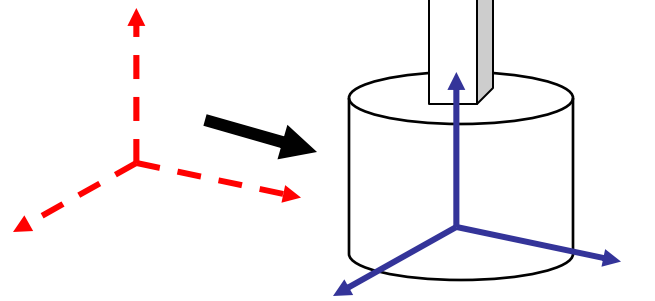
This hierarchical representation is referred to as **Scene Graph**

# Relative Transformation

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

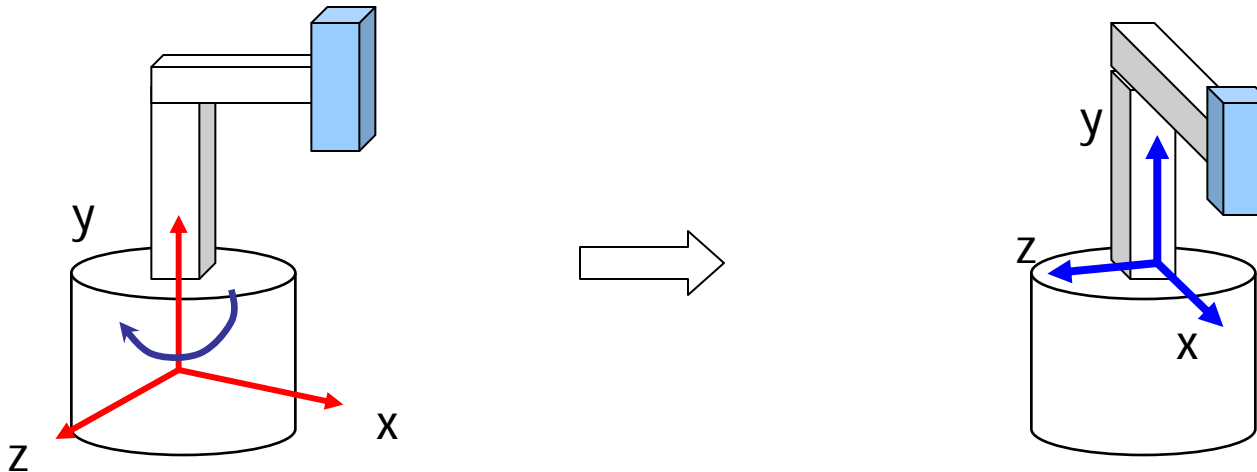


**Step 1: Translate base and its descendants by  $(5,0,0)$ ;**



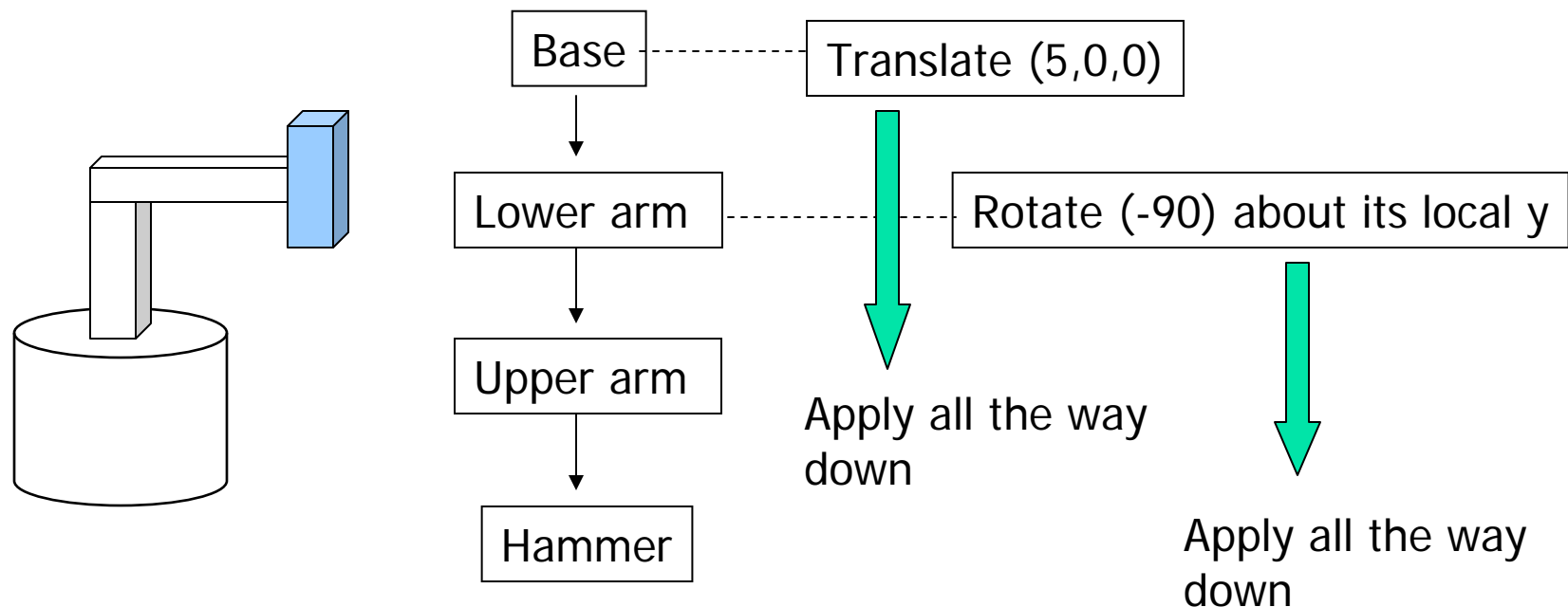
# Relative Transformation (2)

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



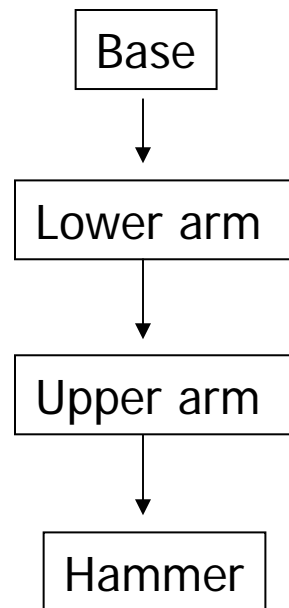
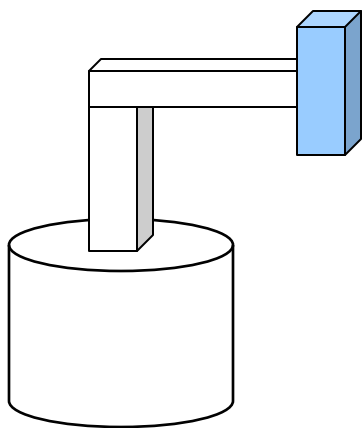
# Relative Transformation (3)

- Represent relative transformations using scene graph



# 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);
```

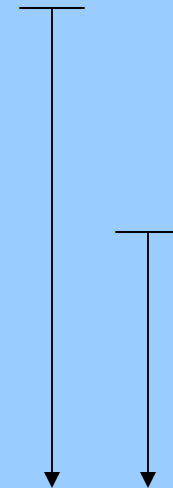
```
Draw_base();
```

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

```
Draw_lower_arm();
```

```
Draw_upper_arm();
```

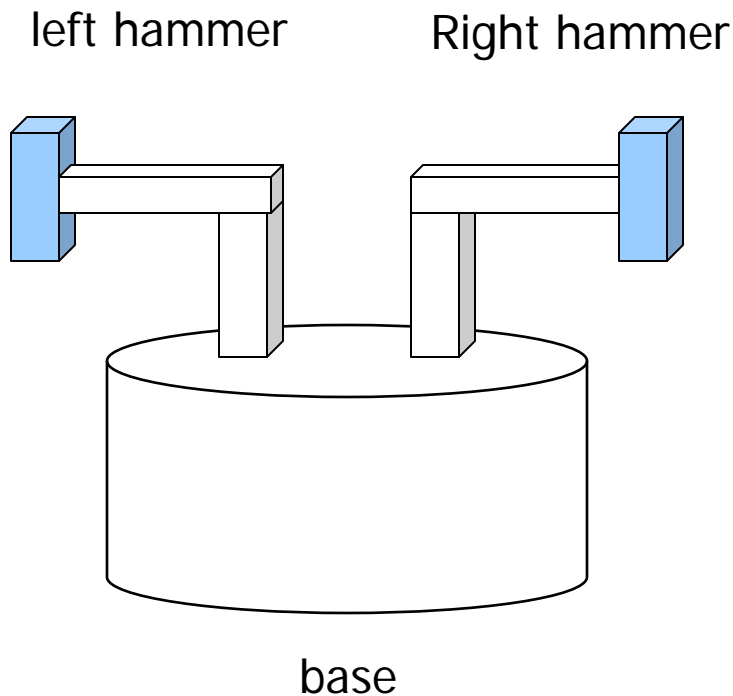
```
Draw_hammer();
```



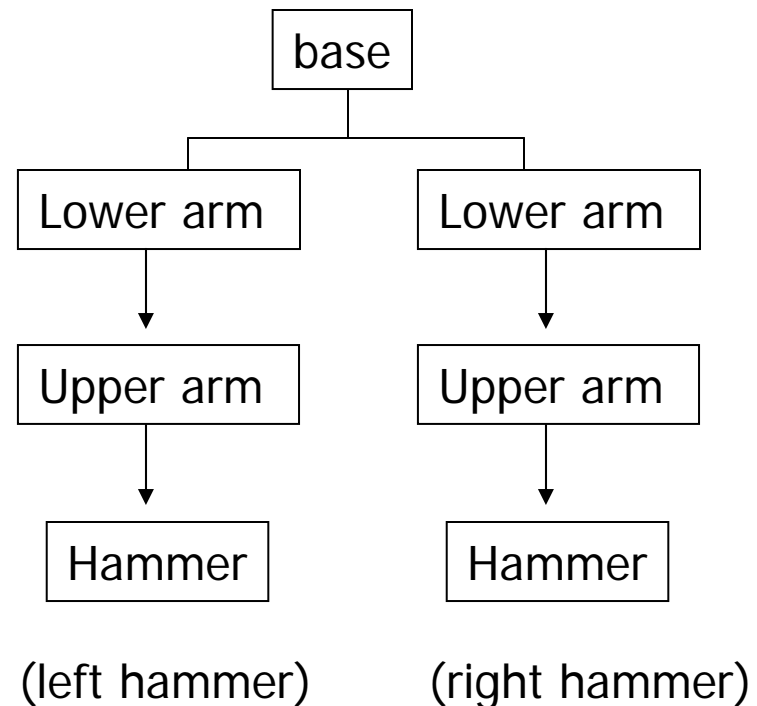


# A more complicated example

- How about this model?



Scene Graph?

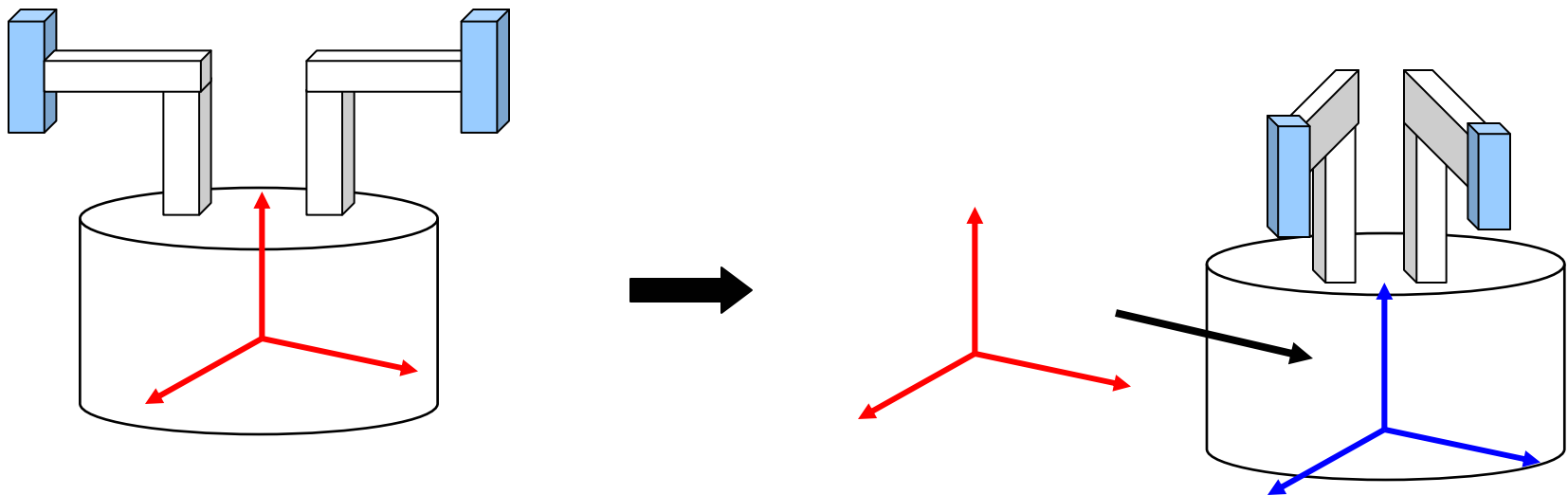




## Do this ...

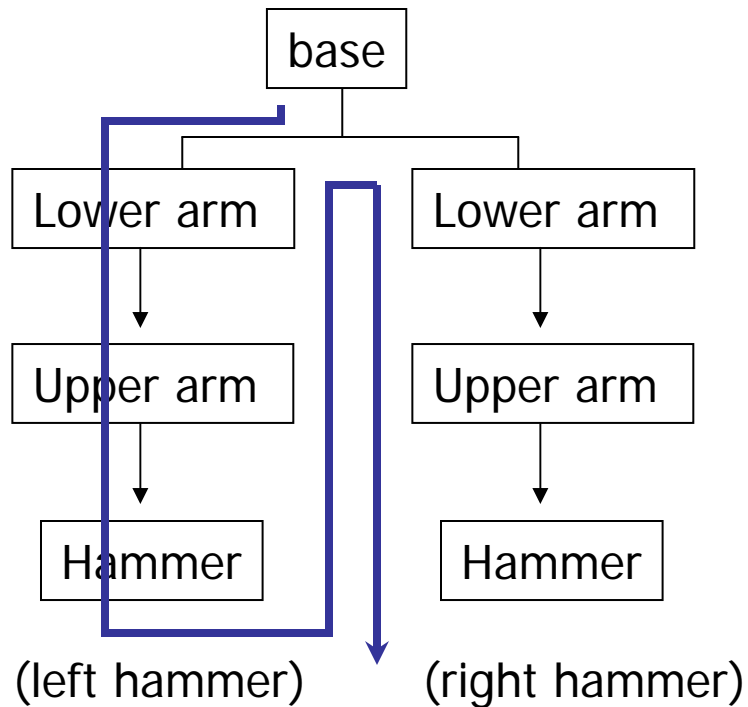
---

- 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)

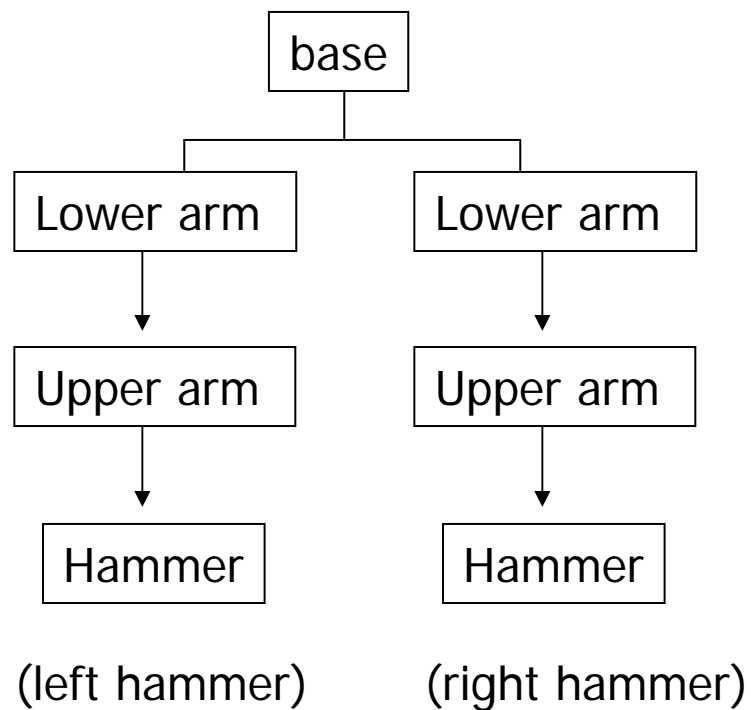
Draw left arm

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

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