CS 543: Computer Graphics
Lecture 4 (Part II): Introduction to 3D Modeling

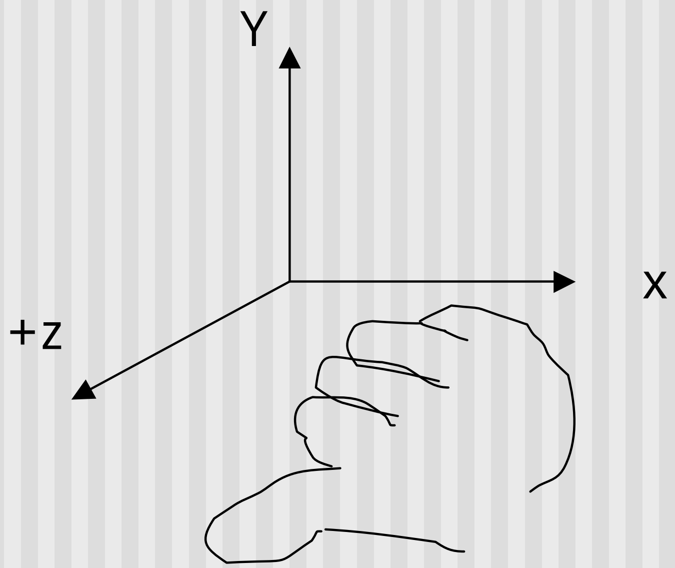
Emmanuel Agu

3D Modeling

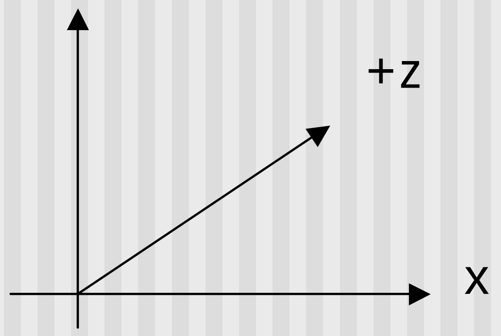
- Overview of OpenGL modeling (Hill 5.6)
- Modeling: create 3D model of scene/objects
- OpenGL commands
  - Coordinate systems (left hand, right hand, openGL-way)
  - Basic shapes (cone, cylinder, etc)
  - Transformations/Matrices
  - Lighting/Materials
  - Synthetic camera basics
  - View volume
  - Projection
- GLUT models (wireframe/solid)
- Scene Description Language (SDL): 3D file format

Coordinate Systems

- Tip: sweep fingers x-y: thumb is z



Right hand coordinate system



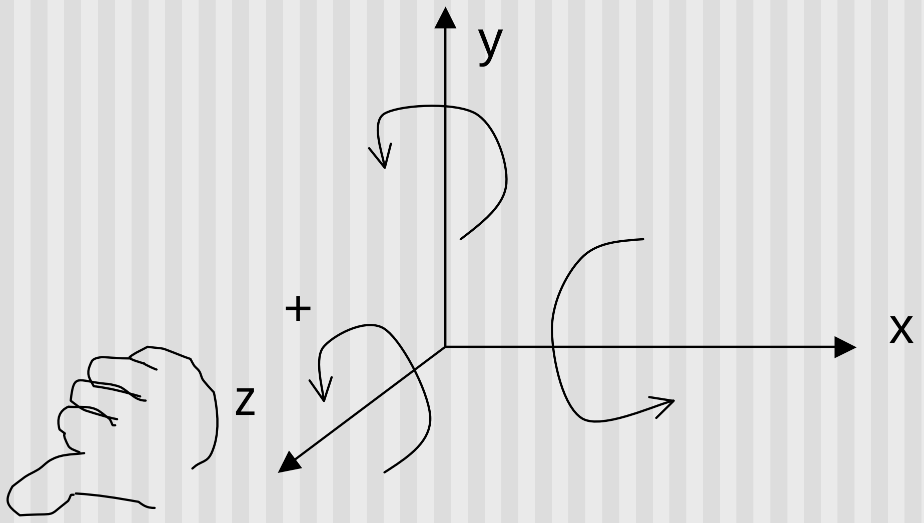
Left hand coordinate system

•Not used in this class and

•Not in OpenGL

Rotation Direction

- Which way is +ve rotation
  - Look in –ve direction (into +ve arrow)
  - CCW is +ve rotation

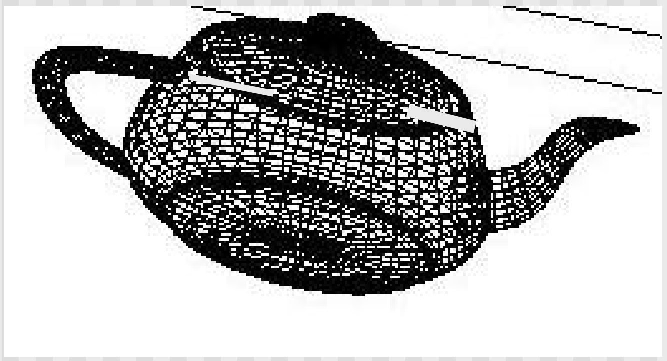


3D Modeling: GLUT Models

- Two main categories:
  - Wireframe Models
  - Solid Models
- Basic Shapes
  - Cylinder: glutWireCylinder( ), glutSolidCylinder( )
  - Cone: glutWireCone( ), glutSolidCone( )
  - Sphere: glutWireSphere( ), glutSolidSphere( )
  - Cube: glutWireCube( ), glutSolidCube( )
- More advanced shapes:
  - Newell Teapot: (symbolic)
  - Dodecahedron, Torus

GLUT Models: 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)
Also glutSolidTeapot( )

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

3D Modeling: GLUT Models

- Glut functions actually
  - generate sequence of points that define corresponding shape
  - centered at 0.0
- Without GLUT models:
  - Use generating functions
  - More work!!
- What does it look like?
  - Generates a list of points and polygons for simple shapes
  - Spheres/Cubes/Sphere

Cylinder Algorithm

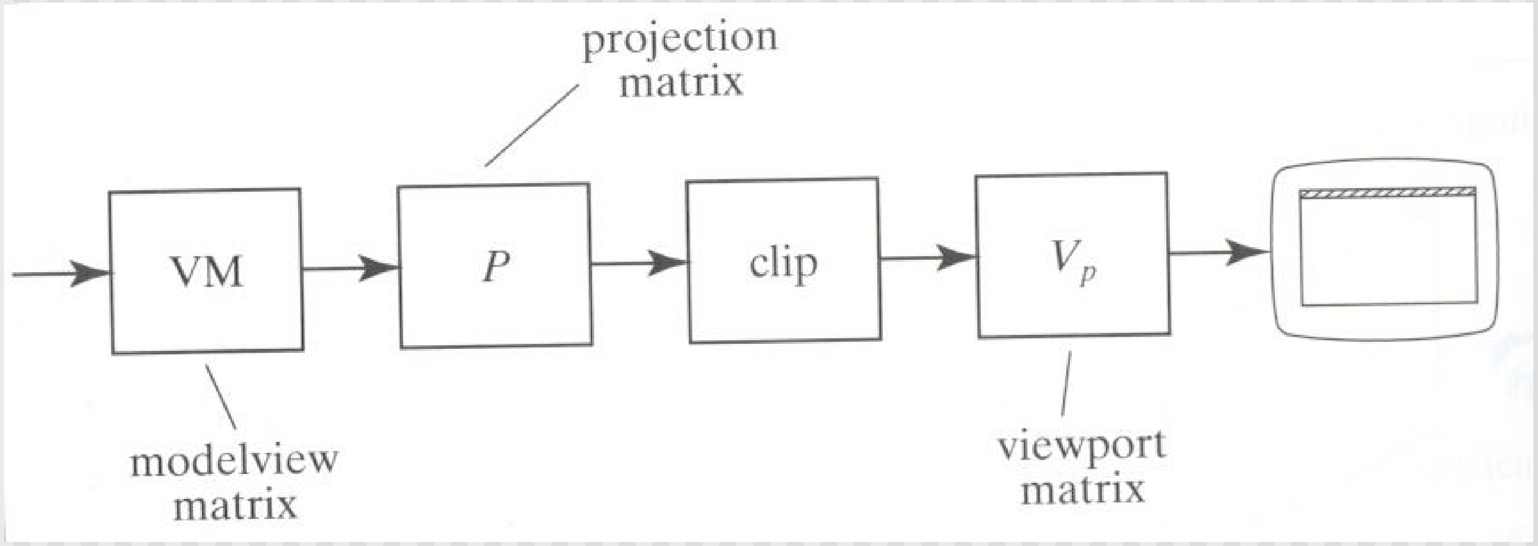
```
glBegin(GL_QUADS)
    For each A = Angles{
        glVertex3f(R*cos(A), R*sin(A), 0);
        glVertex3f(R*cos(A+DA), R*sin(A+DA), 0)
        glVertex3f(R*cos(A+DA), R*sin(A+DA), H)
        glVertex3f(R*cos(A), R*sin(a), H)
    }

// Make Polygon of Top/Bottom of cylinder
```

3D Transforms

- Scale:
  - glScaled(sx, sy, sz) - scale object by (sx, sy, sz)
- Translate:
  - glTranslated(dx, dy, dz) - translate object by (dx, dy, dz)
- Rotate:
  - glRotated(angle, ux, uy, uz) – rotate by angle about an axis passing through origin and (ux, uy, uz)
- OpenGL
  - Creates matrices for each transform (scale, translate, rotate)
  - Multiplies matrices together to form 1 combined matrix
  - Combined geometry transform matrix called ***modelview matrix***

OpenGL Matrices

Graphics pipeline: vertices goes through series of operations

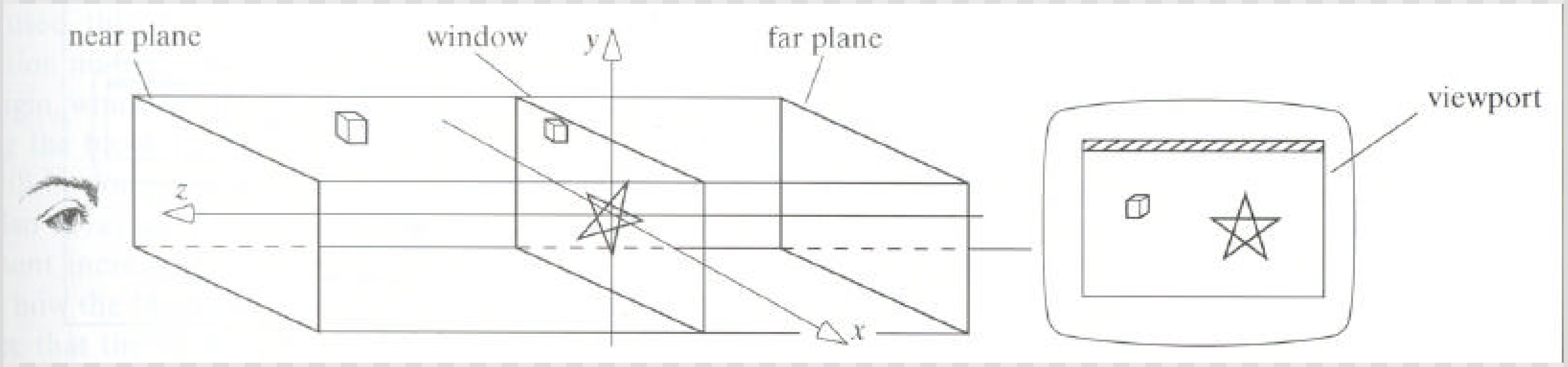


OpenGL Matrices/Pipeline

- OpenGL uses 3 matrices (simplified) for geometry:
  - Modelview matrix:
  - Projection matrix:
  - Viewport matrix:
- Modelview matrix:
  - combination of modeling matrix *M* and Camera transforms *V*
- Other OpenGL matrices include texture and color matrices
- glMatrixMode command selects matrix mode
- May initialize matrices with glLoadIdentity( )
- glMatrixMode parameters: GL_MODELVIEW, GL_PROJECTION, GL_TEXTURE, etc
- OpenGL matrix operations are 4x4 matrices
- Graphics card: fast 4x4 multiplier -> tremendous speedup

View Volume

- Side walls determined by window borders
- Other walls determined by programmer-defined
    - Near plane
    - Far plane
- Convert 3D models to 2D:
    - Project points/vertices inside view volume unto view window using parallel lines along z-axis



Projection

- Different types of projections?
  - Different view volume shapes
  - Different visual effects
- Example projections
  - Parallel
  - Perspective
- Parallel is simple
- Will use for this intro, expand later

OpenGL Matrices/Pipeline

- Projection matrix:
  - Scales and shifts each vertex in a particular way.
  - View volume lies inside cube of –1 to 1
  - Reverses sense of z: increasing z = increasing depth
  - Effectively squishes view volume down to cube centered at 1
- Clipping: (in 3D) then eliminates portions outside view volume
- Viewport matrix:
  - Maps surviving portion of block (cube) into a 3D viewport
  - Retains a measure of the depth of a point

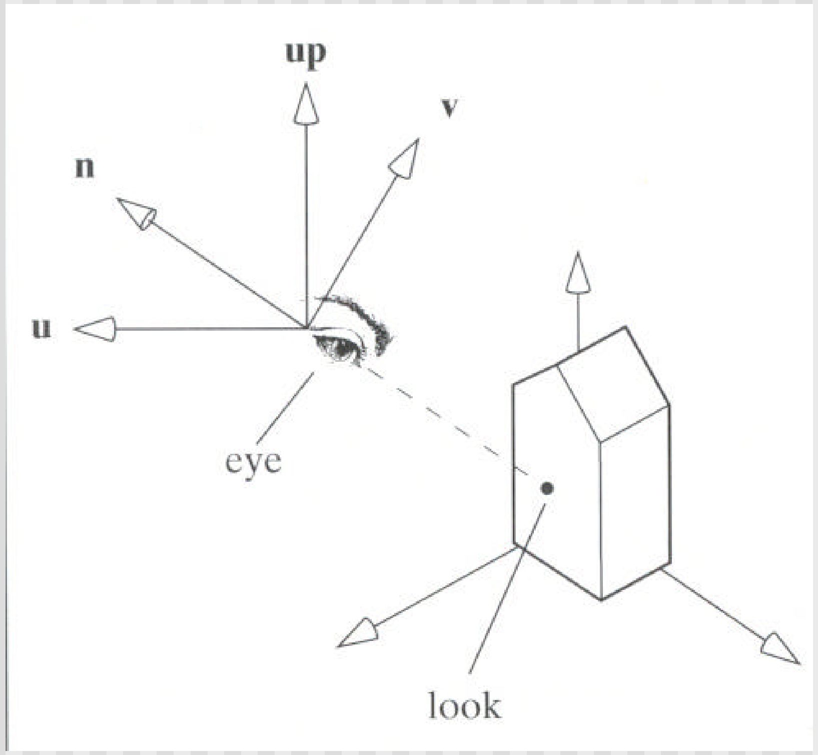
Lighting and Object Materials

- Light components:
  - Diffuse, ambient, specular
  - OpenGL: glLightfv( ), glLightf( )
- Materials:
  - OpenGL: glMaterialfv( ), glMaterialf( )

Synthetic Camera

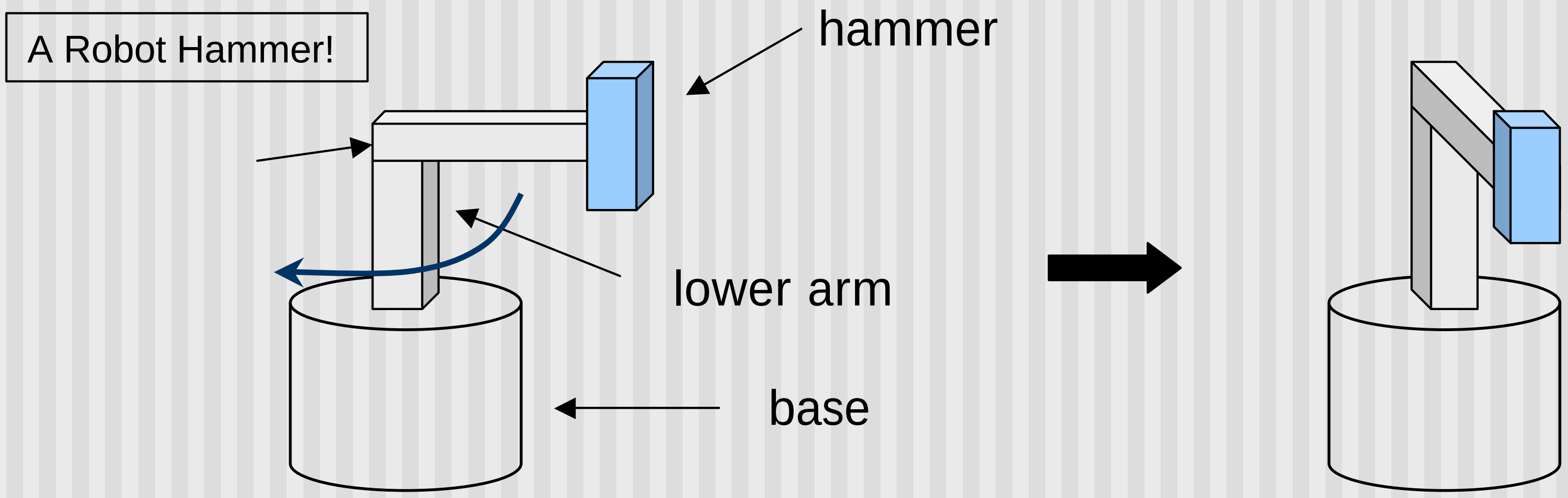
- Define:
  - Eye position
  - LookAt point
  - Up vector (if spinning: confusing)
- Programmer knows scene, chooses:
  - *eye*
  - *lookAt*
- *Up* direction usually set to (0,1,0)
- OpenGL:
  - gluLookAt*(eye.x, eye.y, eye.z, look.x, look.y, look.z, up.x, up.y, up.z)*

Synthetic Camera



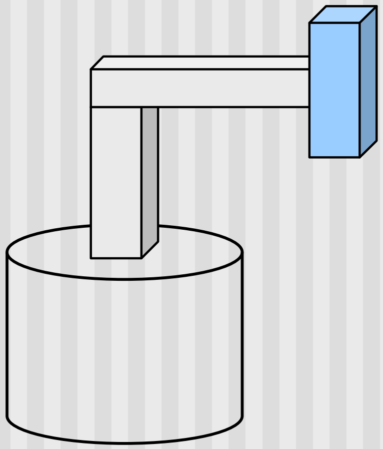
Hierarchical Transforms Using OpenGL

- Two ways to model
  - Immediate mode (OpenGL)
  - Retained mode (SDL)
- Graphical scenes have object dependency,
- Many small objects
- Attributes (position, orientation, etc) depend on each other



Hierarchical Transforms Using OpenGL

- Object dependency description using tree structure



Root node

$$\begin{tikzcd}
\boxed{\text{Base}} \arrow[d] \\
\boxed{\text{Lower arm}} \arrow[d] \\
\boxed{\text{Upper arm}} \arrow[d] \\
\boxed{\text{Hammer}}
\end{tikzcd}$$

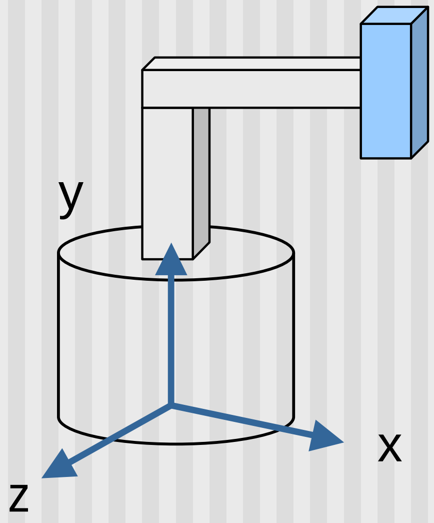
Leaf node

Object position and orientation can be affected by its parent, grand-parent, grand-grand-parent ... nodes

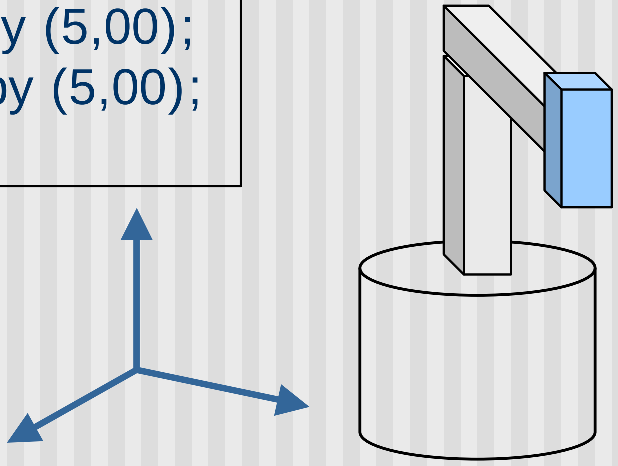
Hierarchical representation is known as Scene Graph

Transformations

- Two ways to specify transformations:
  - (1) Absolute transformation: each part of the object is transformed independently relative to the origin



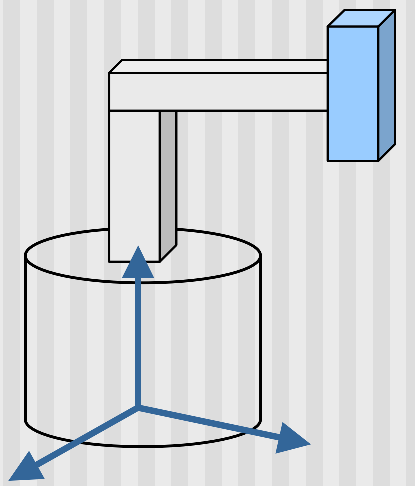
Translate the base by (5,0,0);
Translate the lower arm by (5,00);
Translate the upper arm by (5,00);
...



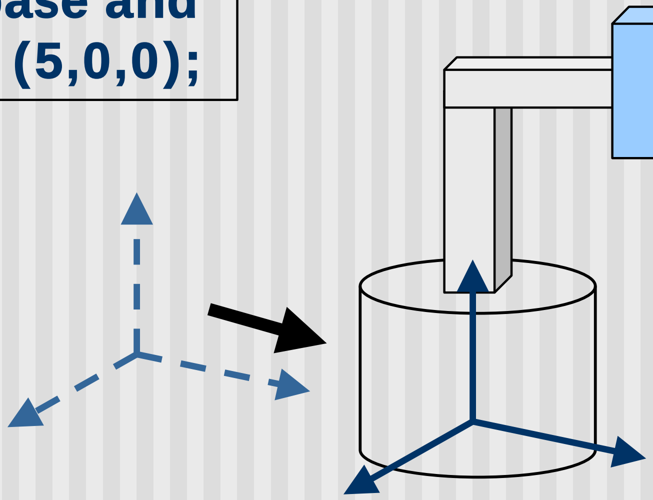
Relative Transformation

A better (and easier) way:

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

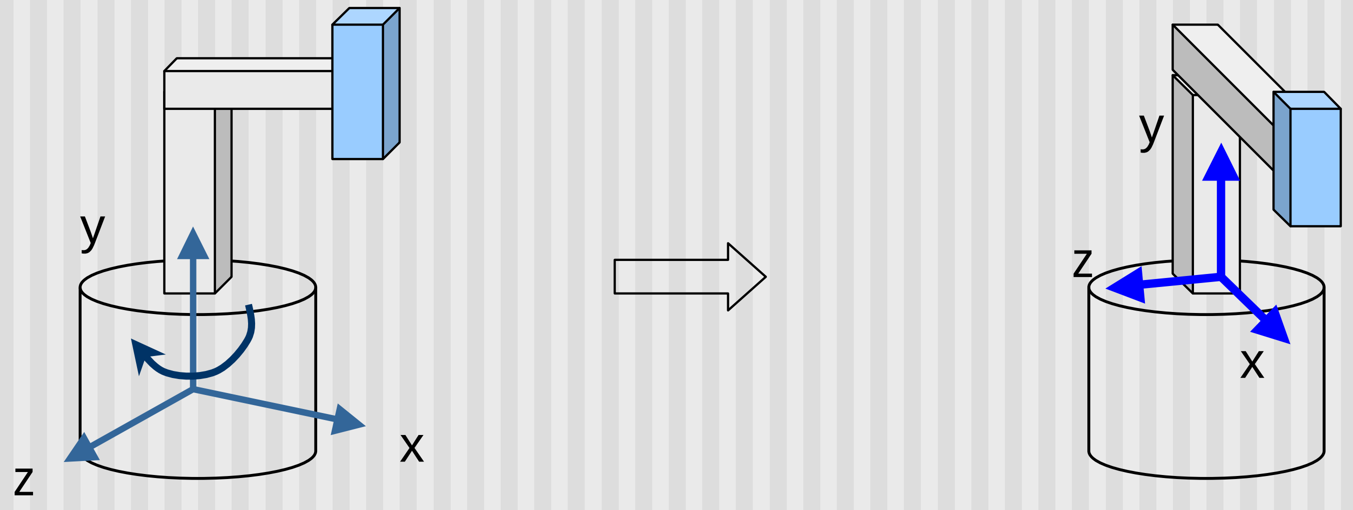


Step 1: Translate base and its descendants by (5,0,0);



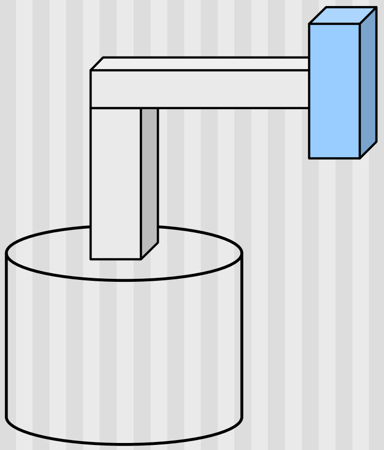
Relative Transformation

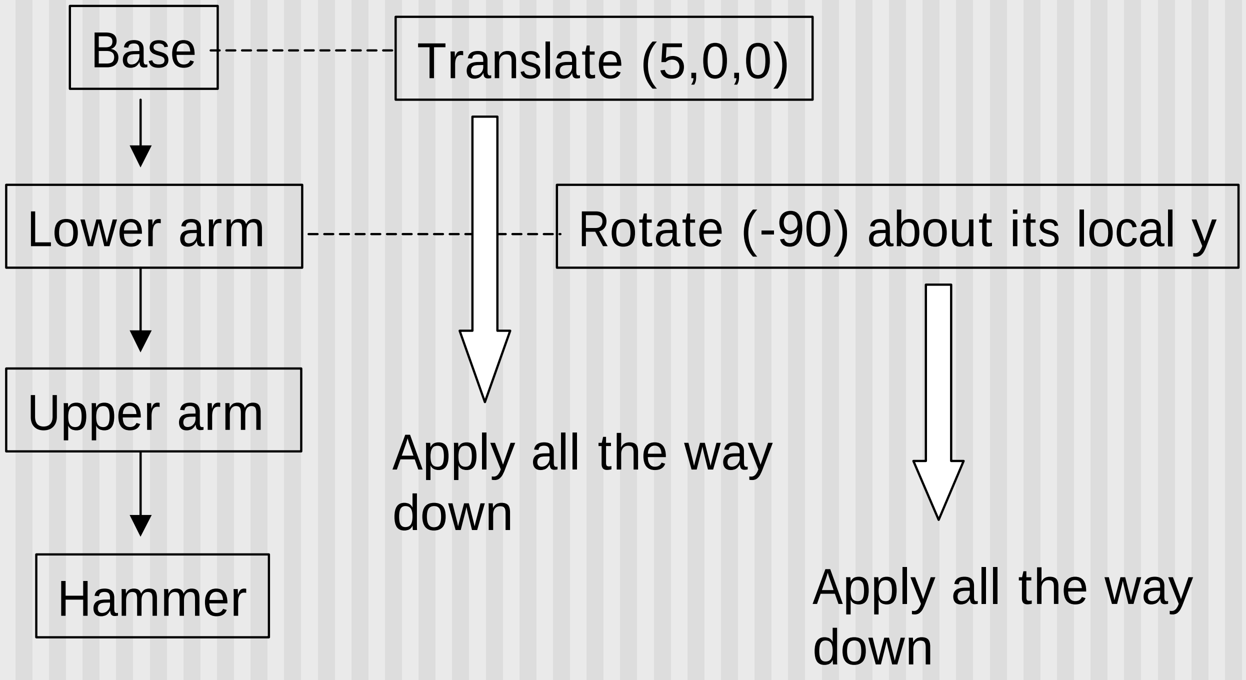
Step 2: Rotate the lower arm and all its descendants relative to the base's local y axis by -90 degree



Relative Transformation

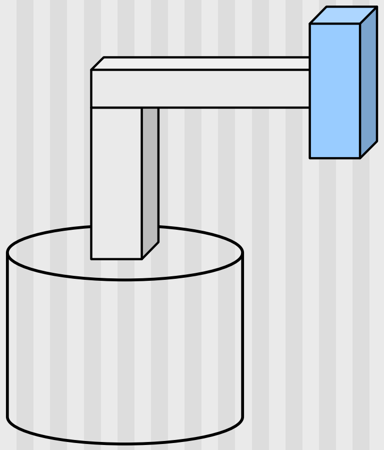
- Represent relative transformation using scene graph





Hierarchical Transforms Using 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



$$\begin{tikzcd}\boxed{\text{Base}} \arrow[d] \\ \boxed{\text{Lower arm}} \arrow[d] \\ \boxed{\text{Upper arm}} \arrow[d] \\ \boxed{\text{Hammer}}\end{tikzcd}$$

```
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();
```

Hierarchical Models

- Two important calls:
  - glPushMatrix( ): load transform matrix with following matrices
  - glPopMatrix( ): restore transform matrix to what it was before glPushMatrix( )
- If matrix stack has M1 at the top, after glPushMatrix( ), positions 1 and 2 on matrix stack have M1
- If M1 is at the top and M2 is second in position, glPopMatrix( ) destroys M1 and leaves M2 at the top
- To pop matrix without error, matrix must have depth of at least 2
- Possible depth of matrices vary.
  - Modelview matrix allows 32 matrices
  - Other matrices have depth of at least 2

Example: Table modeled with OpenGL

```
// define table leg
//------------------------------------------------------------------------------
void tableLeg(double thick, double len){
    glPushMatrix();
    glTranslated(0, len/2, 0);
    glScaled(thick, len, thick);
    glutSolidCube(1.0);
    glPopMatrix();
}

// note how table uses tableLeg-
void table(double topWid, double topThick, double legThick, double legLen){
    // draw the table - a top and four legs
    glPushMatrix();
    glTranslated(0, legLen, 0);
```

Example: Table modeled with OpenGL

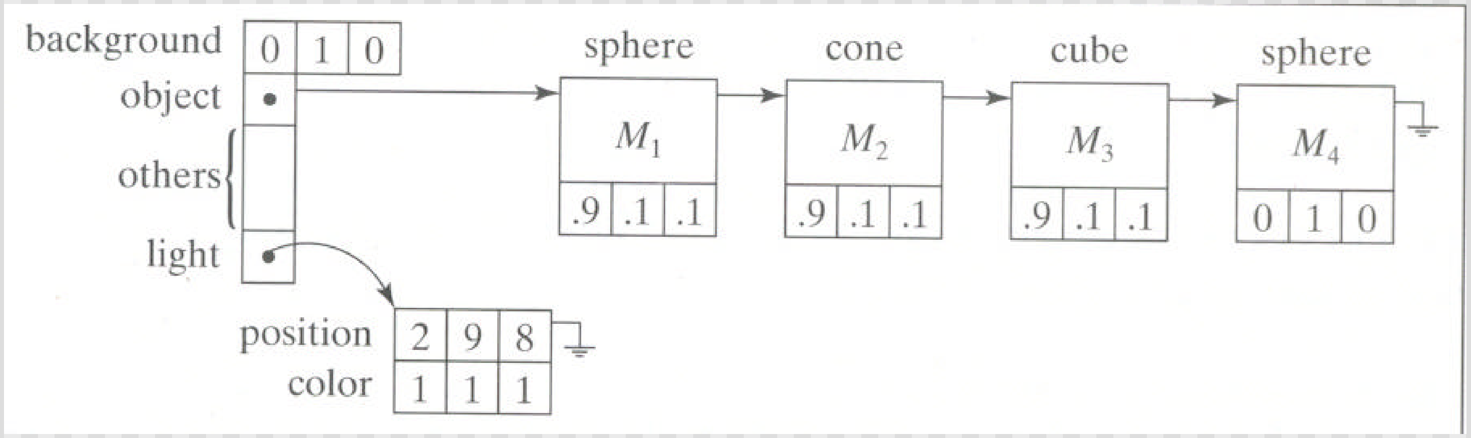
```
    scaled(topWid, topThick, topWid);
    glutSolidCube(1.0);
    glPopMatrix();

    double dist = 0.95 * topWid/2.0 - legThick / 2.0;
    glPushMatrix();
    glTranslated(dist, 0, dist);
    tableLeg(legThick, legLen);
    glTranslated(0, 0, -2*dist);
    tableLeg(legThick, legLen);
    glTranslated(-2*dist, 0, 2*dist);
    tableLeg(legThick, legLen);
    glTranslated(0, 0, -2*dist);
    tableLeg(legThick, legLen);
    glPopMatrix();
}
```

Example: Table modeled with OpenGL

```
// translate and then call

    glTranslated(0.4, 0, 0.4);
    table(0.6, 0.02, 0.02, 0.3); // draw the table
```

SDL

- Immediate mode graphics with openGL: a little tougher
- SDL: Example language for **retained mode** graphics
- SDL makes hierarchical modeling easy
- SDL data structure format



SDL

- Easy interface to use
- 3 steps:
- Step One
  - #include “sdl.h”
  - Add sdl.cpp to your make file/workspace
- Step Two:
  - Instantiate a Scene Object
  - Example: Scene scn;
- Step Three:
  - scn.read(“your scene file.dat”); // reads your scene
  - scn. makeLightsOpenGL(); // builds lighting data structure
  - scn. drawSceneOpenGL(); // draws scene using OpenGL

Example: Table with SDL

```
def leg{push translate 0 .15 0 scale .01 .15 .01 cube pop}

def table{
push translate 0 .3 0 scale .3 .01 .3 cube pop
push
translate .275 0 .275 use leg
translate 0 0 -.55 use leg
translate -.55 0 .55 use leg
translate 0 0 -.55 use leg pop
}

push translate 0.4 0 0.4 use table pop
```

Examples

- Hill contains useful examples on:
  - Drawing fireframe models (example 5.6.2)
  - Drawing solid models and shading (example 5.6.3)
  - Using SDL in a program (example 5.6.4)
- Homework 2:
  - involves studying these examples
  - Work with SDL files in OpenGL
  - Start to build your own 3D model (robot)

References

- Hill, 5.6, appendix 3
- Angel, Interactive Computer Graphics using OpenGL (4th edition)
- Hearn and Baker, Computer Graphics with OpenGL (3rd edition)