CS3241: Computer Graphics Topics Today

3D Object Representations

3D Transformation

Hierarchical Transformation

3D Object Representations

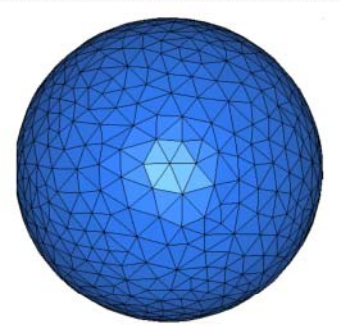
And Polygonal Meshes

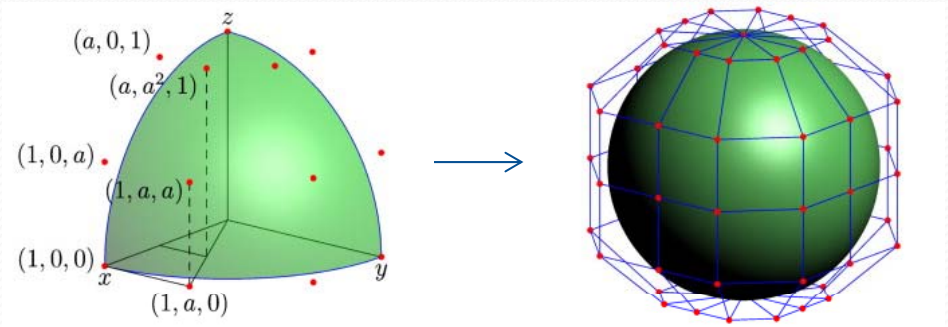
What is a “Representation”?

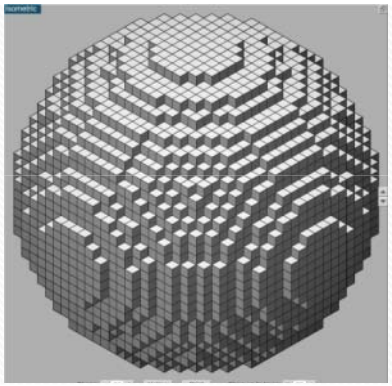
- For the same data/concept, we can have different representations inside the computer
  - E.g. the number `12` can be represented as an `integer`, or a float or a `double` in the memory
  - Or, a set of names, can be stored as a link list, or an array, or even a tree

Various 3D Object Representations

- So for the <u>same</u> 3D object, we can have <u>different</u> representations
  - E.g. a **<u>sphere</u>**, and its FIVE different representations



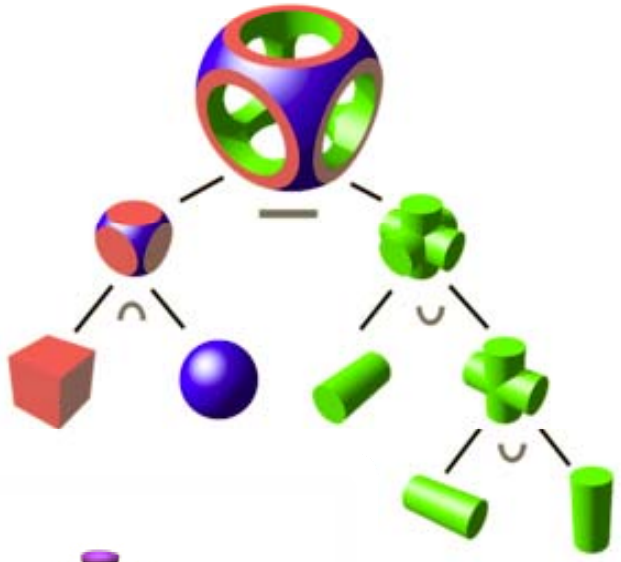


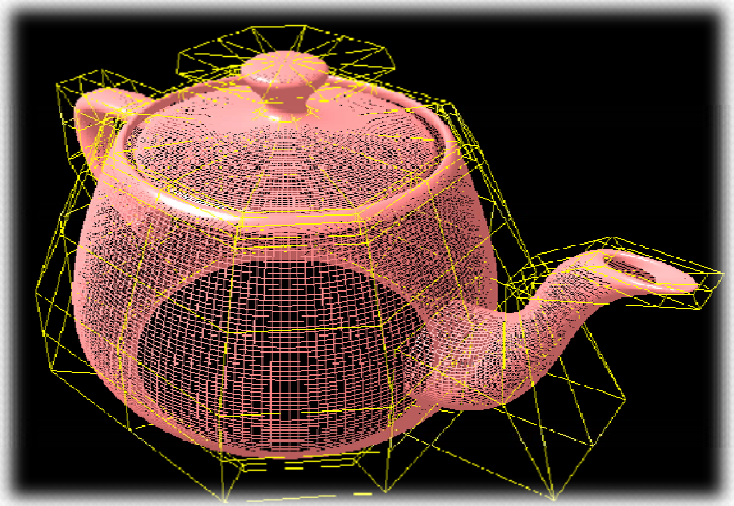


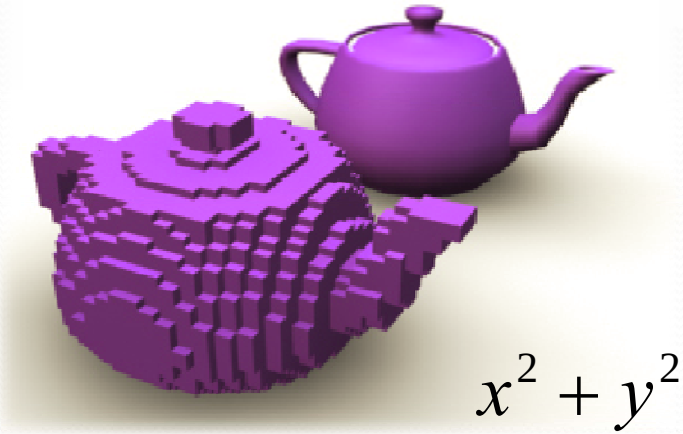
$$x^2 + y^2 + z^2 = 1$$

Various Object **Representations**

- Polygonal meshes
- Parametric patches (Lecture 9)
- Constructive solid geometry (CSG)
- Spatial subdivision techniques
- Implicit representation (Lecture 11)







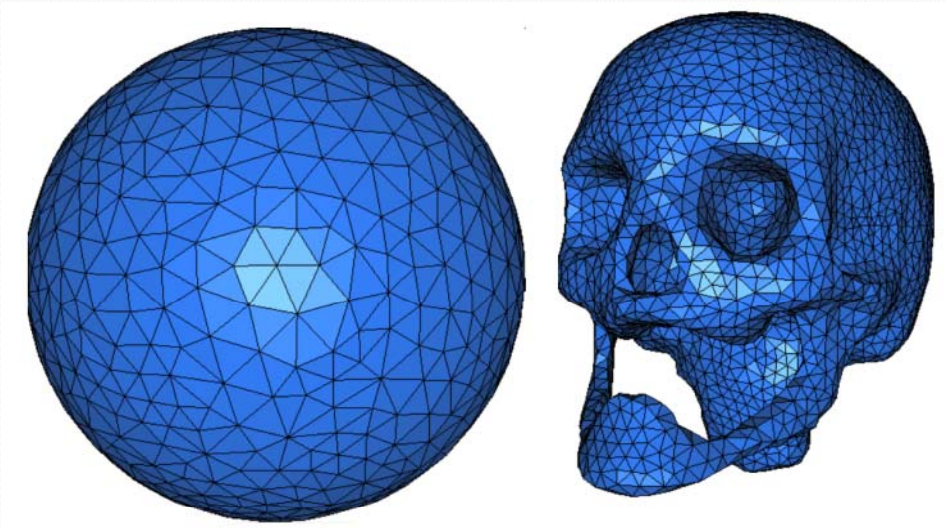
$$x^2 + y^2 + z^2 = 1$$

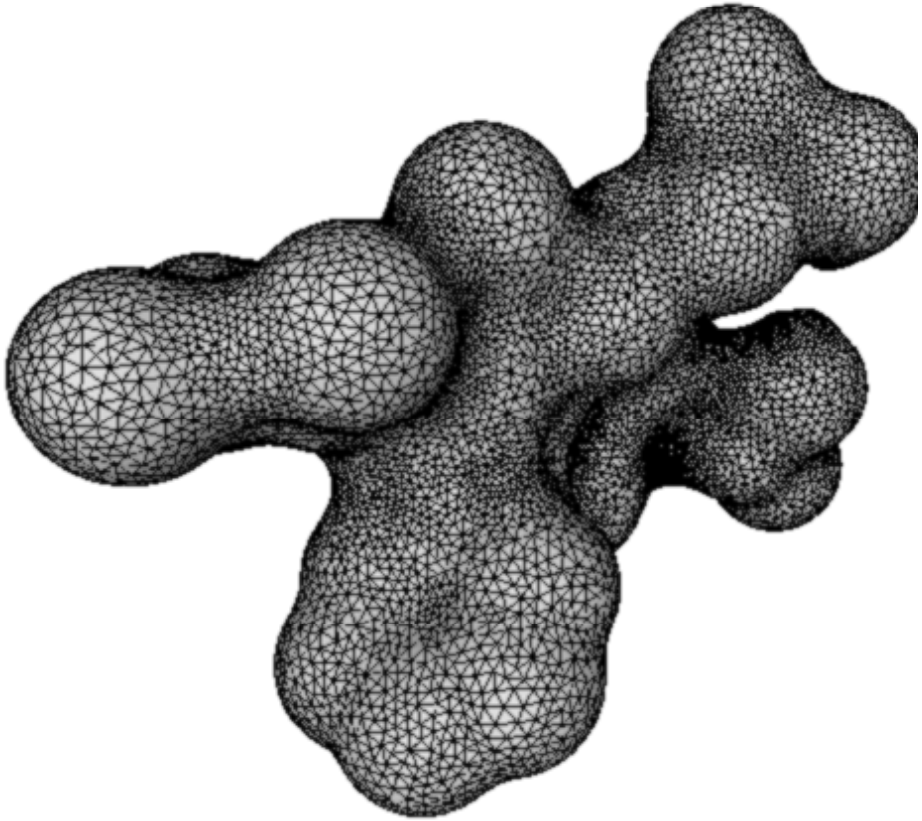
Polygonal Mesh

(Aka surface triangulation)

Polygonal Meshes

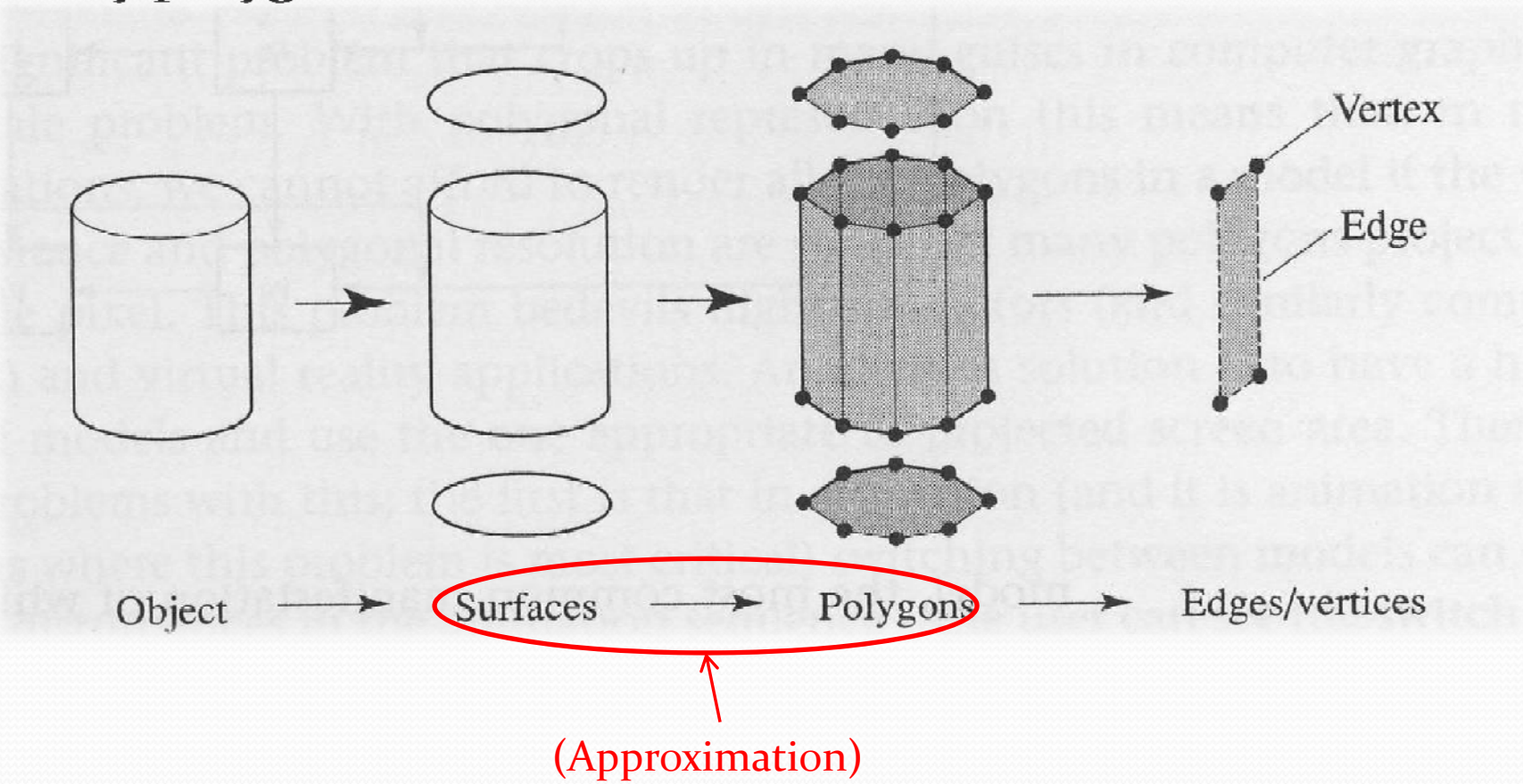
- We can use "soups" of polygons (preferably triangles) to model (actually "approximate") objects
  - It's basically very similar to a "graph"

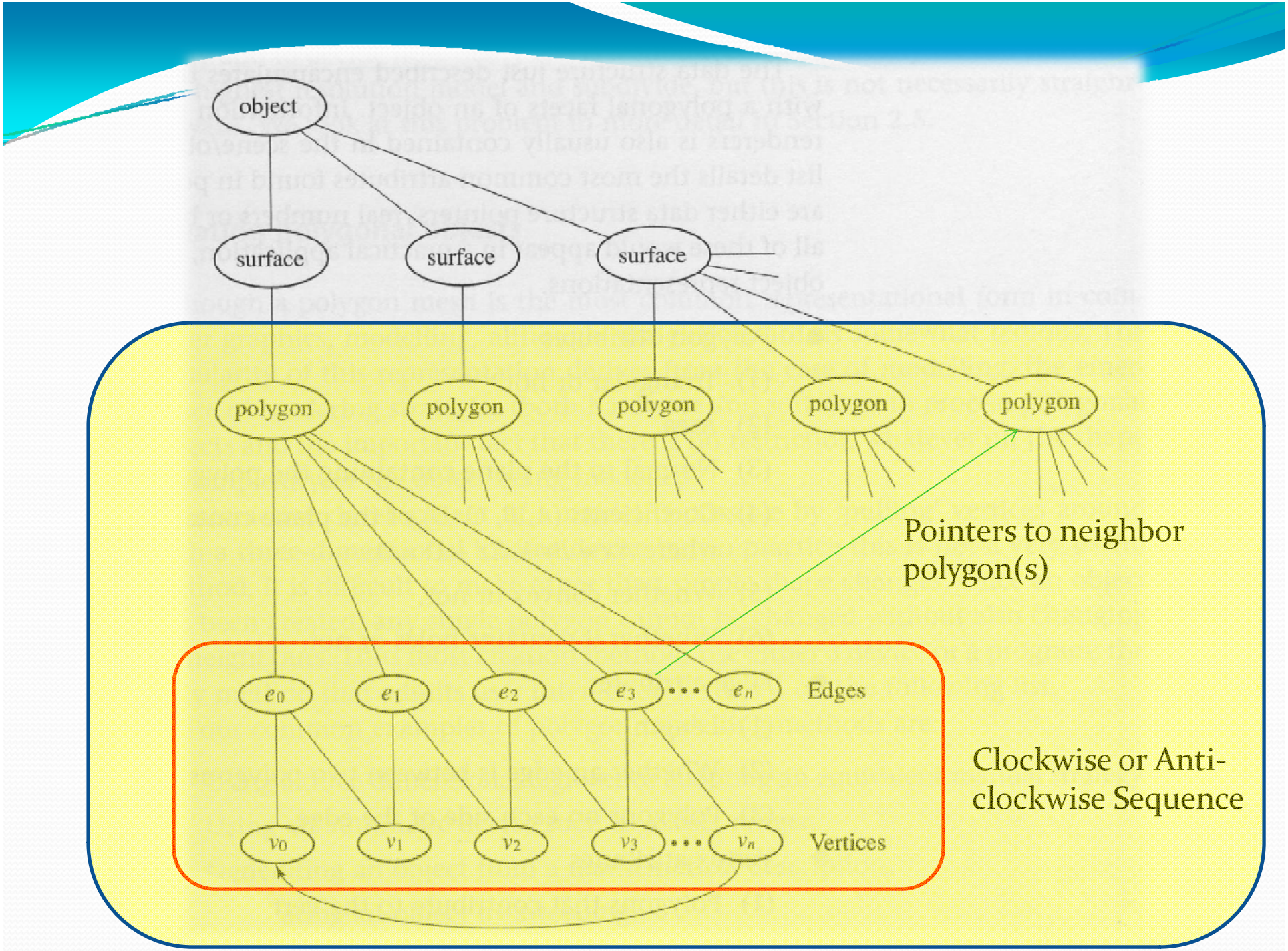




Basic Representation of 3D Objects

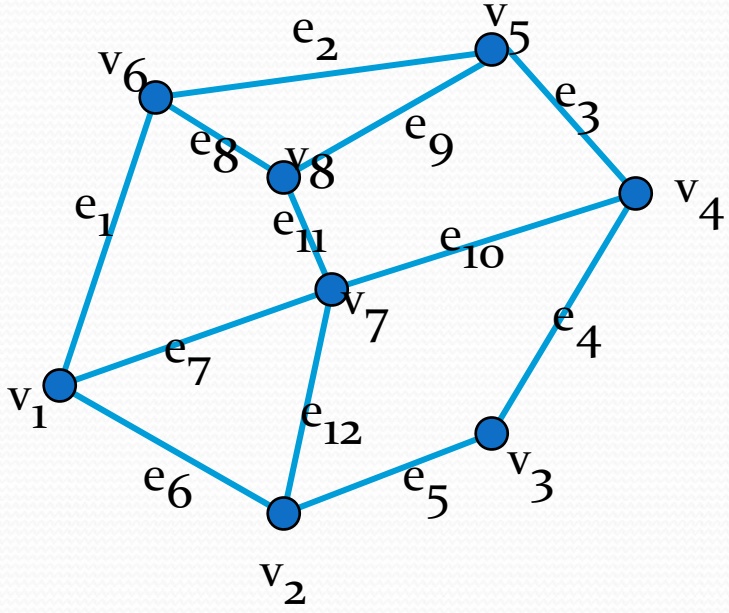
- By polygons




object
surface
surface
surface
polygon
polygon
polygon
polygon
polygon
Pointers to neighbor polygon(s)
e_0
e_1
e_2
e_3
e_n
Edges
v_0
v_1
v_2
v_3
v_n
Vertices
Clockwise or Anti-clockwise Sequence

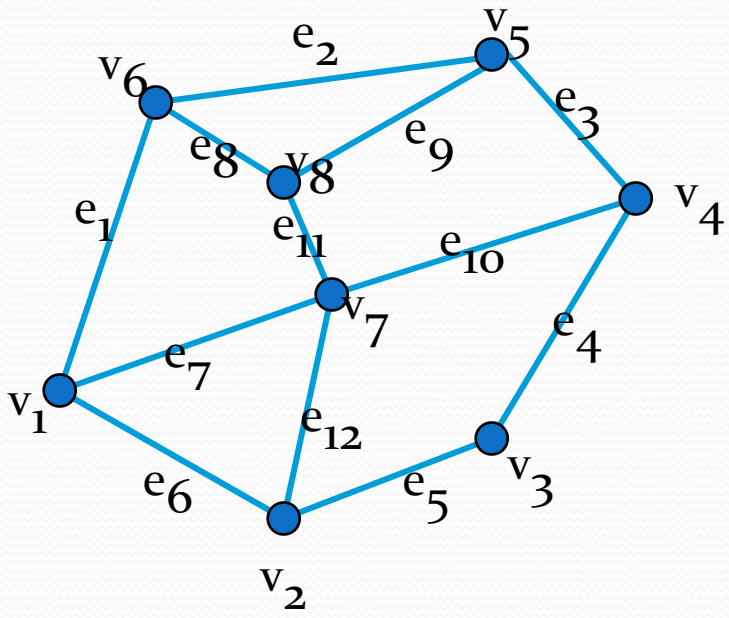
Mesh Representation

- An example of a simple mesh on the right
- There are 8 nodes and 12 edges
  - 5 interior polygons
  - 6 interior (shared) edges
- Each vertex has a location
  - $v_i = (x_i\ y_i\ z_i)$
- We can think of it as a *graph*, vertices are connected by edges

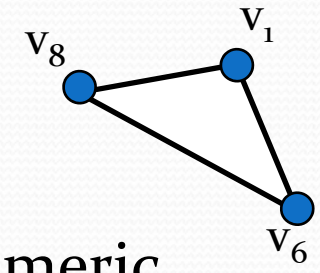


Mesh Topology

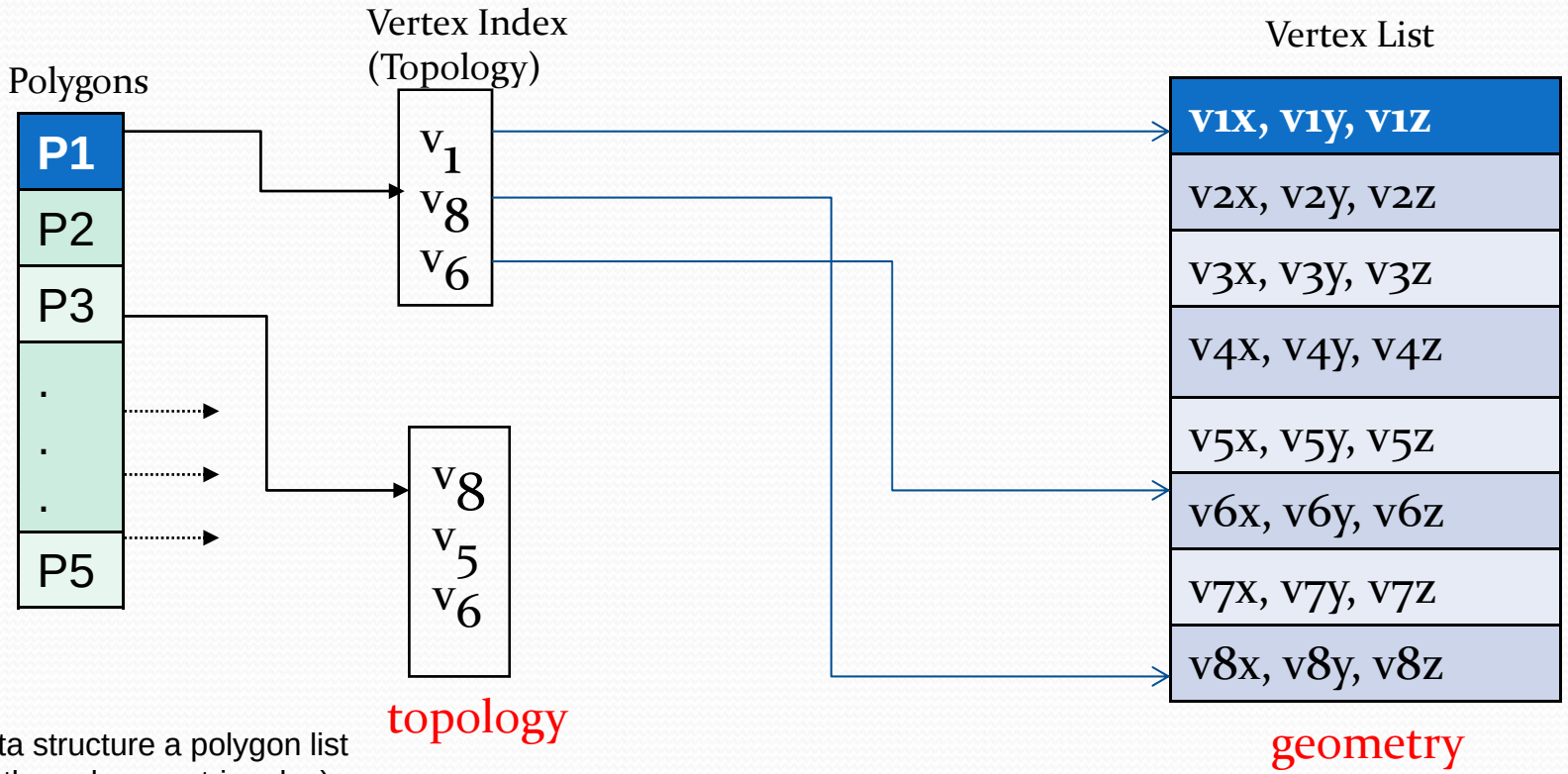
- The edges form the "Topology" of the mesh
- Note that these connections are not changed **even if we modify the positions of the vertices**
- We often consider the topology (connections) to be independent of the geometry (actual 3D point positions)



Vertex List and Polygon List



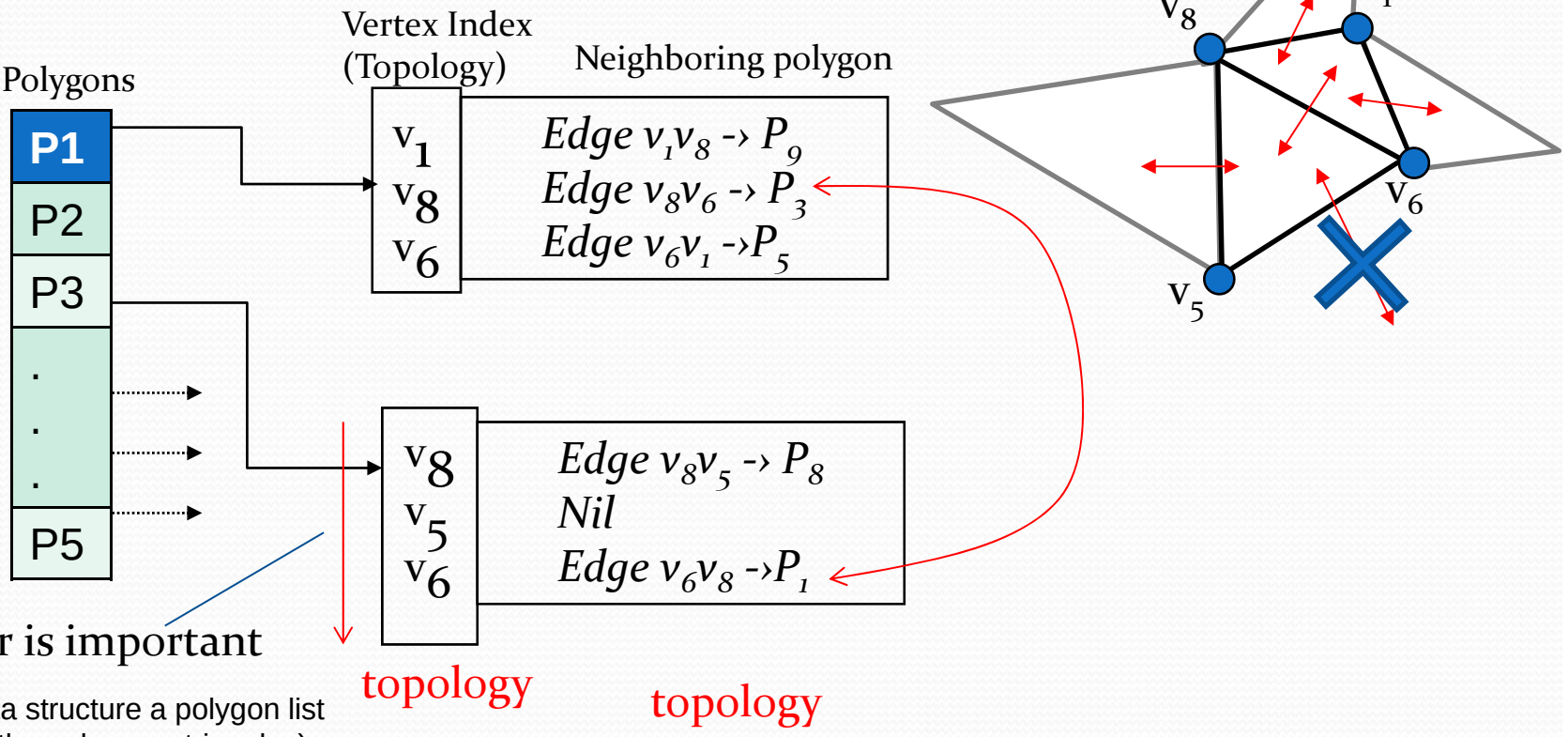
- Store the vertex coordinates in a separate list
  - **Advantages**: save memory, avoid confusion/numeric errors



* We call this data structure a polygon list or triangle list (if the polygons=triangles)
Sometimes the polygons are called “faces”

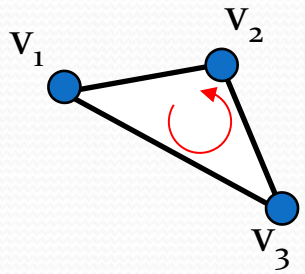
Vertex List and Polygon List

- Store the vertex coordinates in a separate list
  - Advantages: save memory, avoid confusion/numeric errors

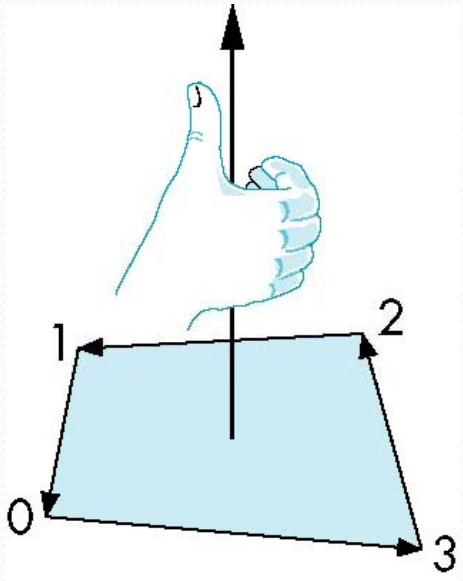


* We call this data structure a polygon list or triangle list (if the polygons=triangles) Sometimes the polygons are called “faces”

Polygon Orientations

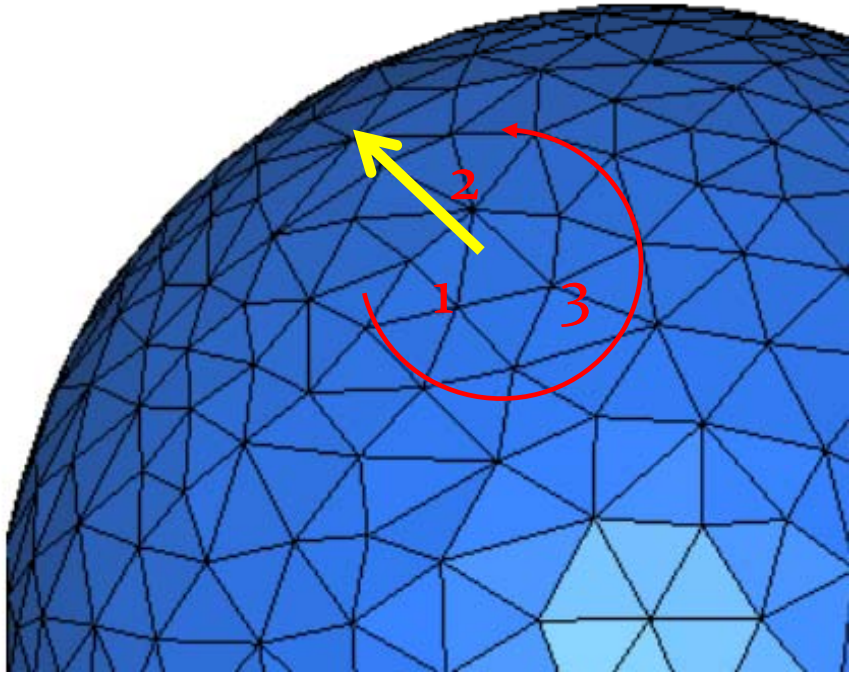


- For a triangle with three vertices v_1, v_2 and v_3
  - The order (v_1, v_3, v_2) and (v_3, v_2, v_1) are considered to be equivalent during drawing, e.g. `glVertex`
- But the order (v_1, v_2, v_3) is considered to be different (or "opposite")
- (v_1, v_3, v_2) and (v_3, v_2, v_1) describe a **facing direction** of a polygon/triangle
  - Right-hand rule
  - The order (v_1, v_2, v_3) creates an opposite facing direction
- So, geometry is the same, but normal vector for that polygon will be different
  - Usually we follow the "natural" order in the polygon list to define the "outward" direction of an object



Vertex List and Polygon List

- For example, if it is a “solid” ball, the triangle below must be stored in the order of
  - (v1, v3, v2) or (v3,v2,v1) or (v2,v1,v3)
  - But NOT (v1,v2,v3) or some other orders



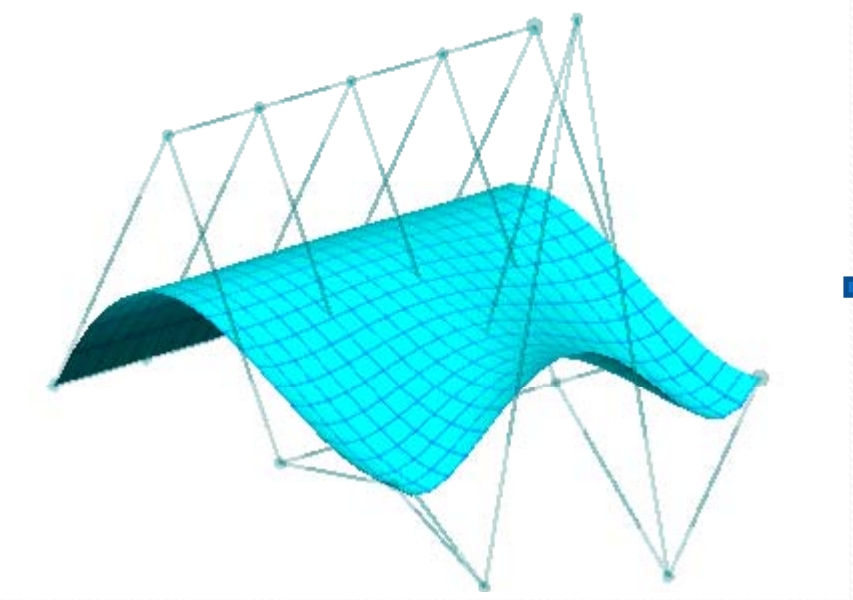
Switching the "Winding" in OpenGL

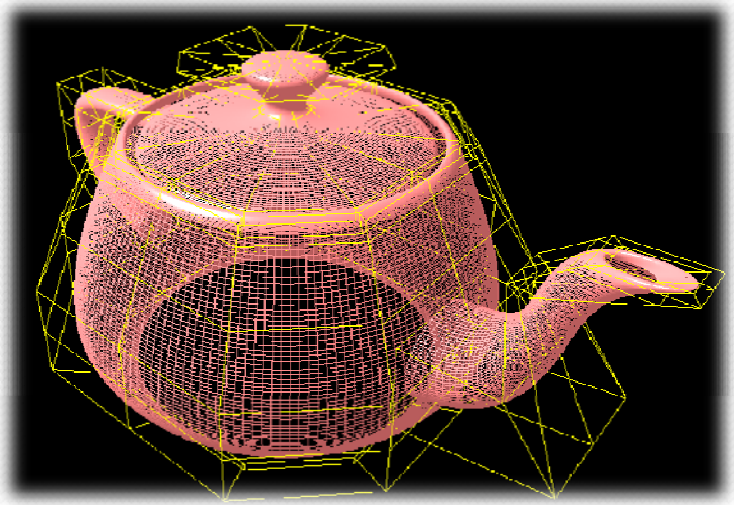
- OpenGL default is that outward facing polygons are specified in counter-clockwise fashion
  - Because, there is a speedup feature such that OpenGL will STOP drawing those polygons when the "outward" direction is not facing your eyes
- But you change this by:
  - `glFrontFace(GL_CW);` `// Clockwise Winding = outward face`
  - `glFrontFace(GL_CCW);` `// Counter-Clockwise Winding = outward face`

Other Object Representations

Various Object **Representations**

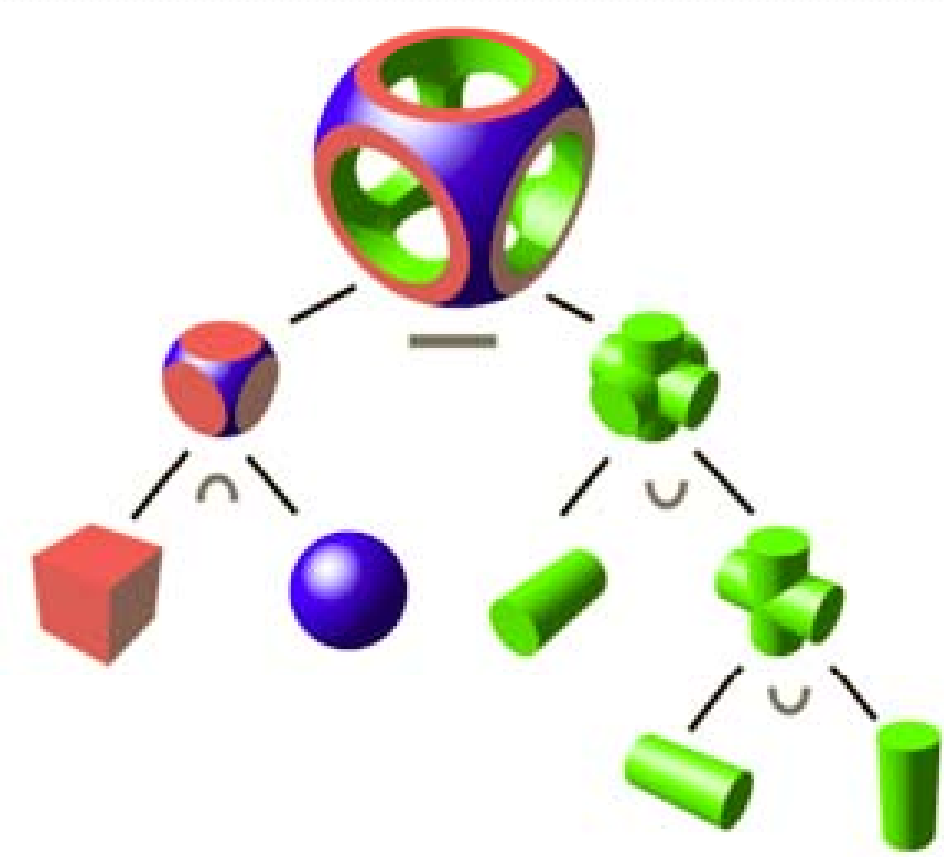
- Parametric patches (Lecture 9)





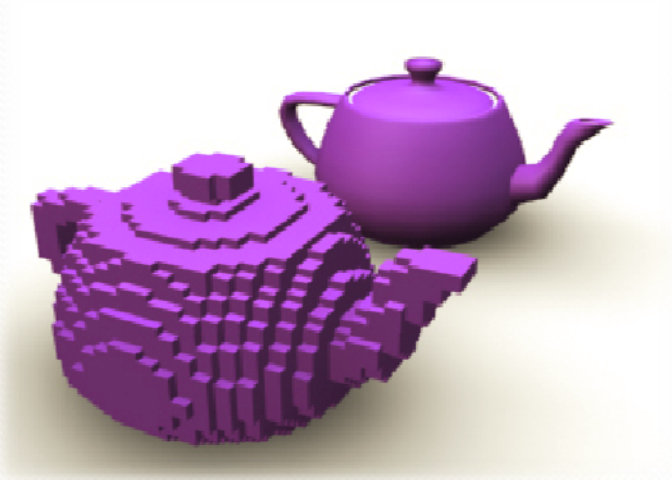
Various Object **Representations**

- Constructive solid geometry (CSG)
- Constructed from a set of primitive shapes
  - E.g. spheres, cubes, cylinders
- Then combined them with different operations:
  - E.g. intersection, union, subtractions, etc



Various Object **Representations**

- Spatial subdivision techniques
  - 3D array of voxels
  - If the object occupies a voxel, turns the voxel on



- Implicit representation

$$x^2 + y^2 + z^2 = 1$$

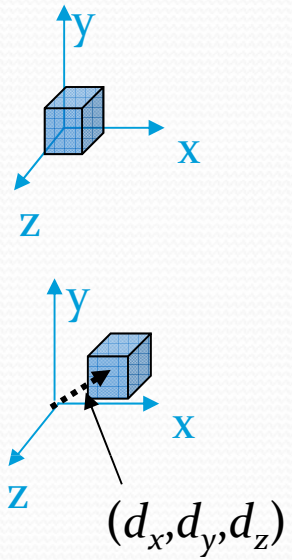
3D Transformation

3D Translation Matrix

- Now a point in 3D is
  - $\boldsymbol{p} = [\, x \;\; y \;\; z \;\; 1 \,]^T$
- We can also express translation using a 4 x 4 matrix **T** in homogeneous coordinates

$$\boldsymbol{p}' = \boldsymbol{T}\boldsymbol{p}$$

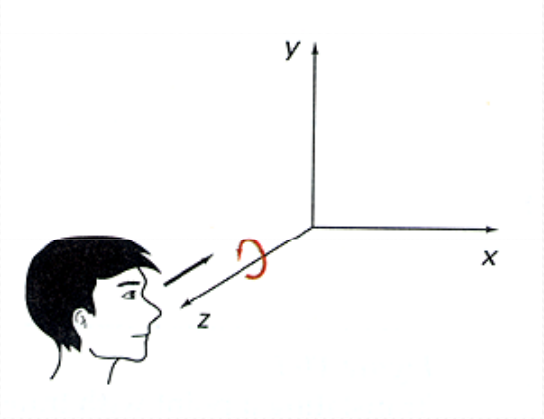
where $\boldsymbol{T} = \boldsymbol{T}(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$



3D Rotation Matrix

- Same argument as for rotation about *z*-axis

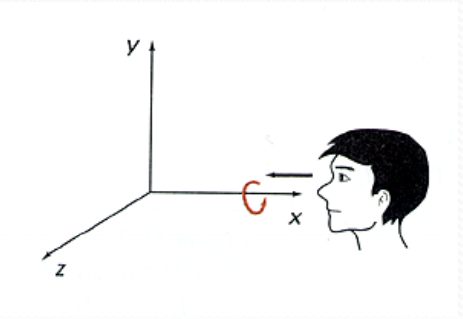
$$\boldsymbol{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



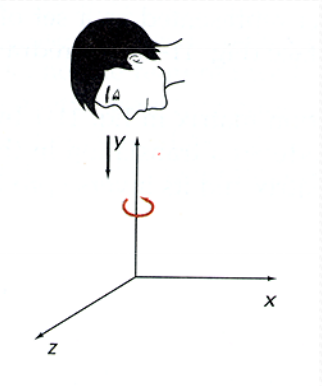
Rotation About x- and y-Axes

- x-axis and y-axis

$$\boldsymbol{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

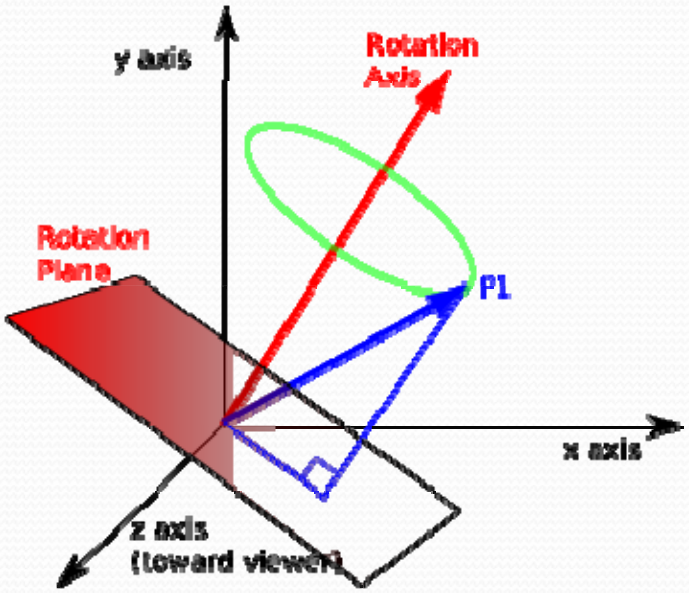


$$\boldsymbol{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



3D Rotation in OpenGL

- **`glRotatef(alpha, x, y, z)`**
  - A point P1 will be rotated around the vector (*x*, *y*, *z*) (that passes through the origin) by an angle `alpha`



Scaling

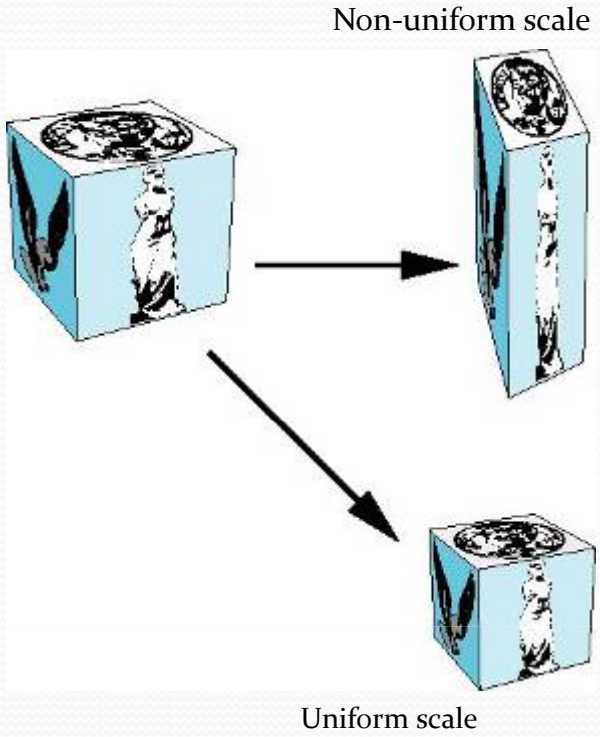
- Expanding or contracting along any axis

$$x' = s_x\, x$$
$$y' = s_y\, y$$
$$z' = s_z\, z$$

  - Or in homogeneous coordinates

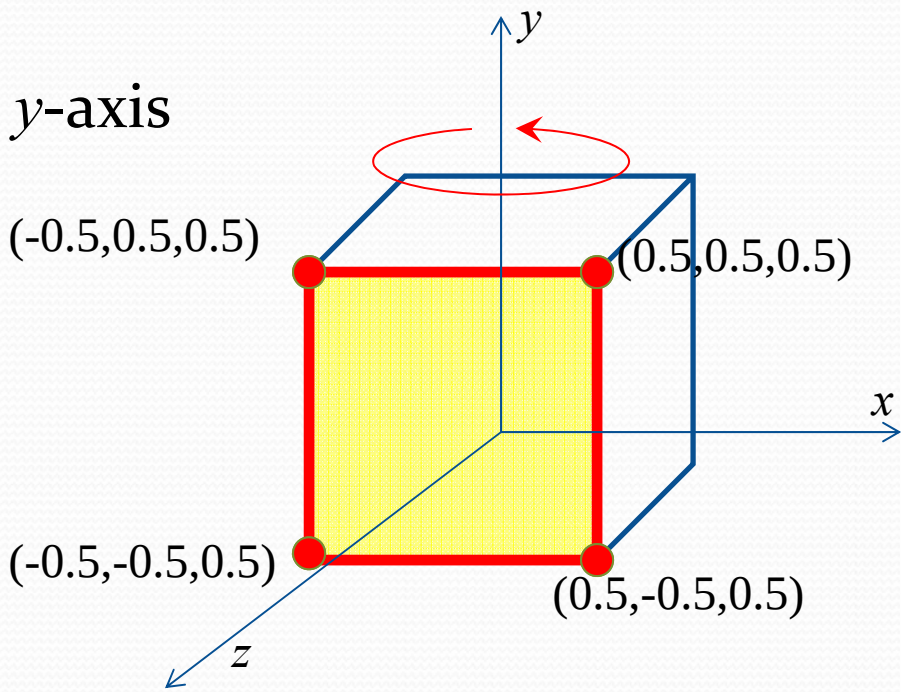
$$\boldsymbol{p}' = \boldsymbol{S}(s_x, s_y, s_z)\, \boldsymbol{p}$$

$$\boldsymbol{S}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



An Example: Drawing a Wireframe Cube

- All the vertices of the cubes are $(\pm 0.5, \pm 0.5, \pm 0.5)$
- Plan:
  - Draw 1 wireframe square
  - Rotate it 4 times around the *y*-axis

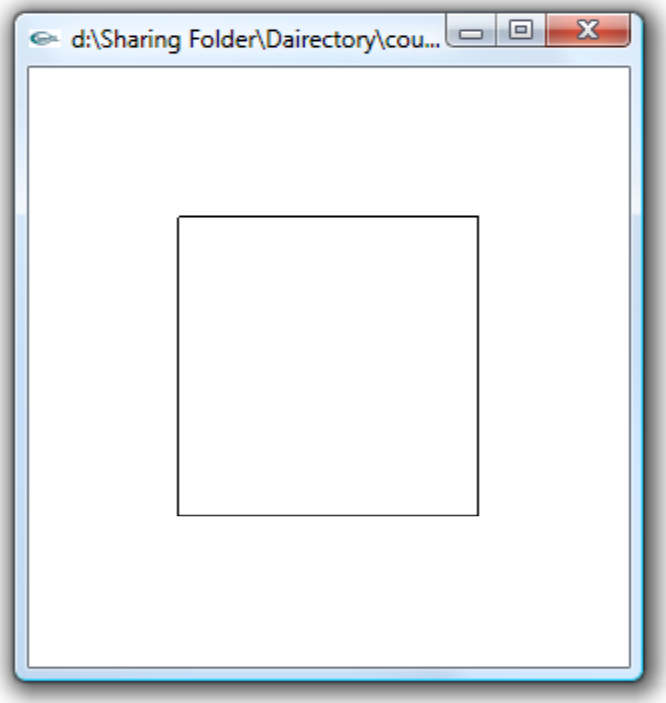


An Example: Drawing a Wireframe Cube

```
void draw3DCubeWireFrame()
{
  glColor3f(0,0,0);
  glPushMatrix();
       for(int i=0;i<4;i++)
       {
              glBegin(GL_LINE_LOOP); // draw one face
                     glVertex3f(0.5,0.5,0.5);
                     glVertex3f(-0.5,0.5,0.5);
                     glVertex3f(-0.5,-0.5,0.5);
                     glVertex3f(0.5,-0.5,0.5);
              glEnd();
              glRotatef(90,0,1,0); // rotate 90 degree (x4)
       }
  glPopMatrix();
}
```

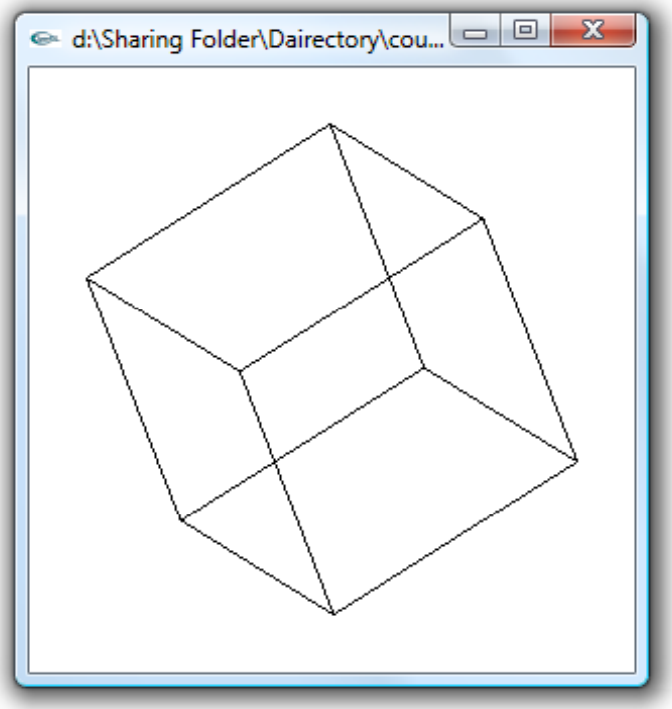
An Example: Drawing a Wireframe Cube

- Just call the routine



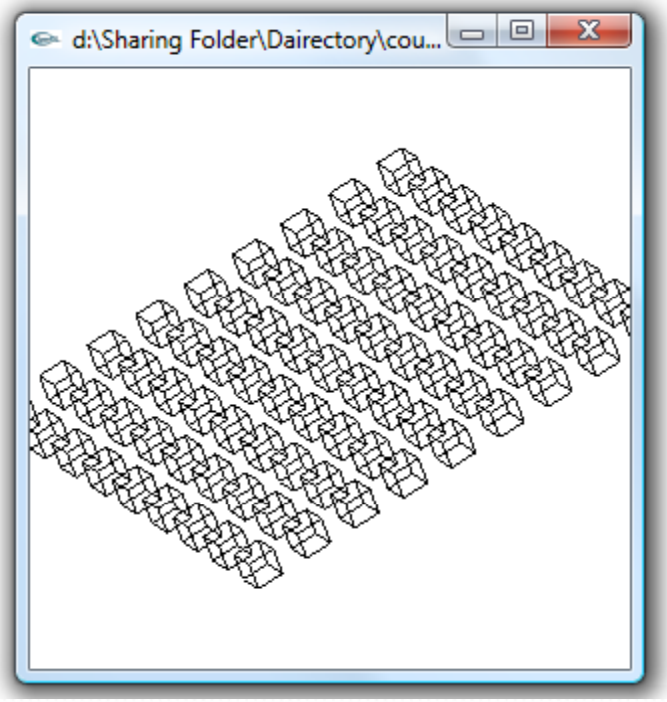
- With

```
glRotatef(45,1,1,1);
draw3DCubeWireFrame();
```



An Example: Drawing a Wireframe Cube

```
glRotatef(45,1,1,1);
glScalef(0.1,0.1,0.1);
for(int i=-8;i<=8;i+=2)
  for(int j=-8;j<=8;j+=2)
  {
    glPushMatrix();
      glTranslatef(i,0,j);
      draw3DCubeWireFrame();
    glPopMatrix();
  }
```



Model View Matrix in OpenGL

The Way That OpenGL Does it

- Whenever you draw a point p by `glVertex`, with a point position p, OpenGL will draw it at the position

$$p'' = Mp$$

  - In which M is the *Model View Matrix*
- For example
  - After one translation, and one rotation

$$M = T(0,0.5)\ R(45)$$

  - So when you execute `glVertex2f(1.0,1.0)`
    - Actually the "real" point will be drawn on the position

$$p'' = Mp = T(0,0.5)\ R(45)\ (1.0,1.0)$$

And OpenGL will only save "M" instead of $T(0,0.5)R(45)$....etc.

The Way That OpenGL Does it

- Whenever you draw a point p by `glVertex`, OpenGL will draw it at the position

$$p'' = Mp$$

$$p'' = T(0,0.5)R(-45^\circ)p$$

M

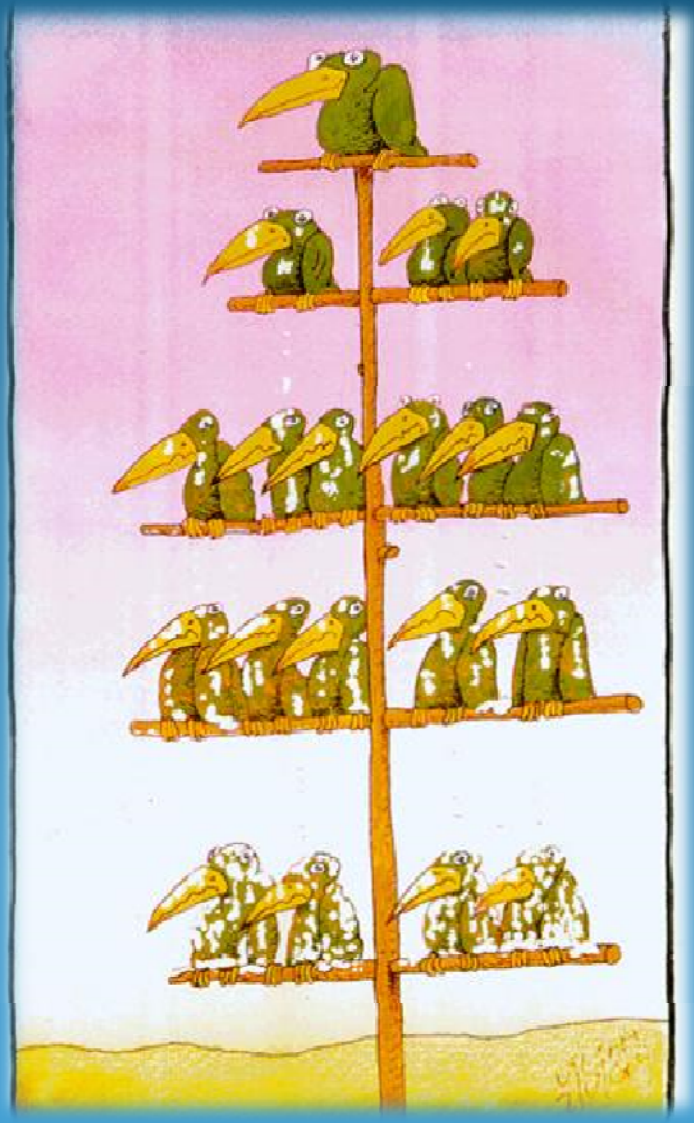
  - In which M is the *Model View Matrix*
- When you want to change this transformation matrix M, you can:
  - **`glMatrixMode(GL_MODELVIEW);`**
    - Switch to this transformation matrix and stack
  - **`glLoadIdentity();`**
    - Set M to be the identity matrix
  - **`glTranslatef()`**
    - Set $M := MT$
  - **`glRotatef()`**
    - Set $M := MR$

Choices:
GL_MODELVIEW
GL_PROJECTION
GL_TEXTURE
GL_COLOR

The Way OpenGL Does it

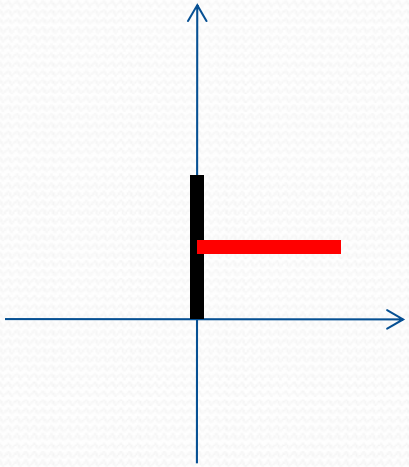
- **`glScalef()`**
  - Set $M := MS$
- **`glPushMatrix();`**
  - Push the current M onto the stack
  - Meaning "Saving the current reference frame"
- **`glPopMatrix();`**
  - Pop a matrix M' from the stack, and replace the current M with M'
  - Meaning "Resuming the last saved reference frame"

Hierarchical Transformation



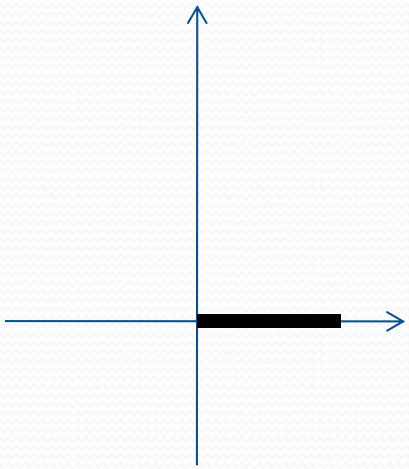
Hierarchical Transformation

- Example
  - Draw a stick figure
- A function called **`drawAUnitLine()`**
  - Draw a line from (0,0) to (0,1)
- Draw the "east" arm
  - Translate (0,0.5)
  - Rotate (-90)
  - `drawAUnitLine()`



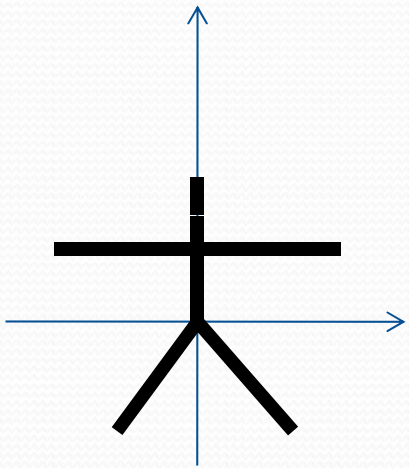
Hierarchical Transformation

```
drawLeftArm()
{
   glPushMatrix();
      glRotatef(-90,0,0,1);
      drawAUnitLine();
   glPopMatrix();
}
```



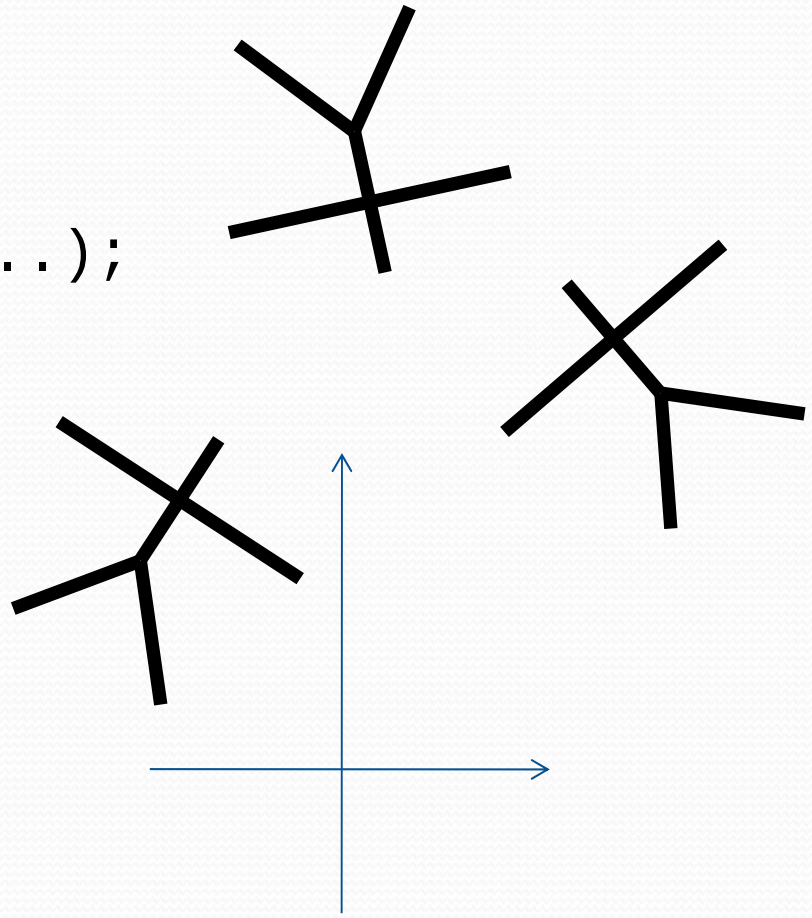
Hierarchical Transformation

```
drawAStickFigure()
{
  glPushMatrix();
    glTranslatef(0,0.5,0);
    drawLeftArm();
    drawRightArm();
  glPopMatrix();
  glPushMatrix();
    drawLeftLeg();
    drawRightLeg();
  glPopMatrix();
}
```



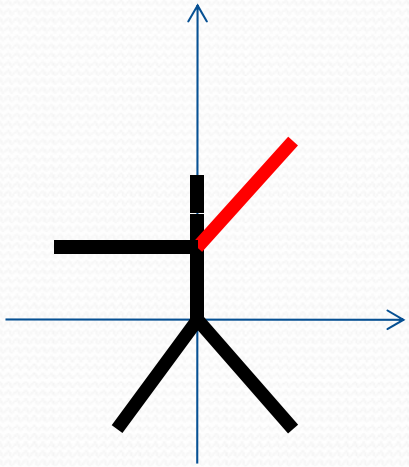
How do I Draw this?

```
myDisplay()
{
  glPushMatrix();
    glTranslatef(some where..);
    glRotatef(some angle…);
    drawAStickFigure();
  glPopMatrix();
  .
  .
  .
  // draw many many
}
```



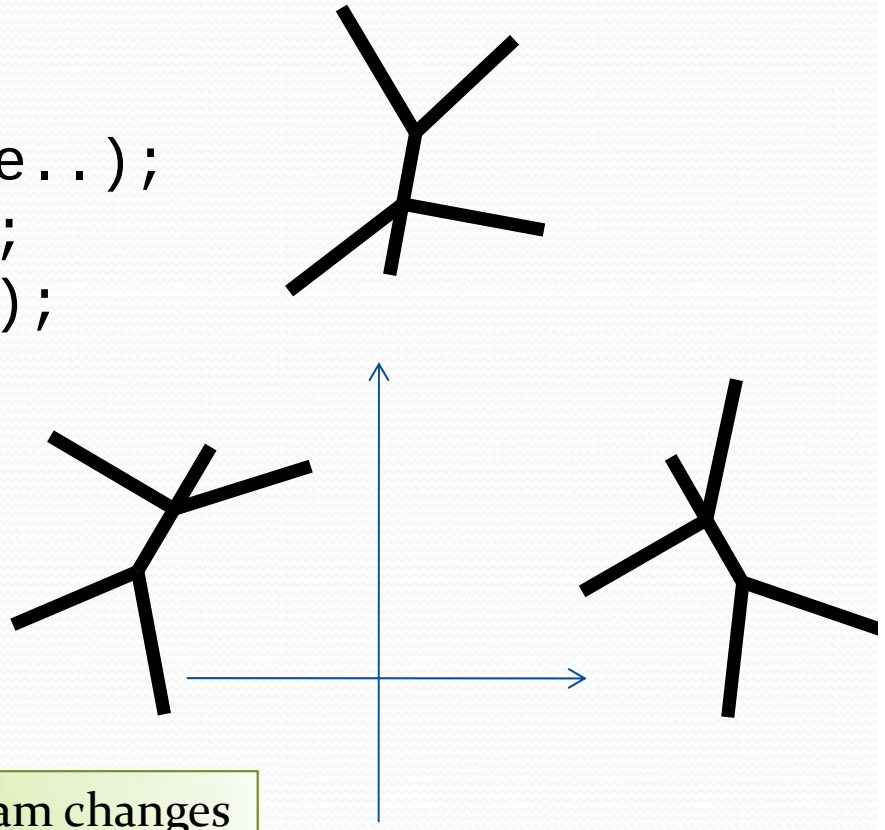
Make the Left Arm Wave

```
drawAStickFigure(double angle)
{
  glPushMatrix();
    glTranslatef(0,0.5,0);
    glPushMatrix();
      glRotatef(angle,0,0,1);
      drawLeftArm();
    glPopMatrix();
    drawRightArm();
  glPopMatrix();
  glPushMatrix();
    drawLeftLeg();
    drawRightLeg();
  glPopMatrix();
}
```



How do I Draw this?

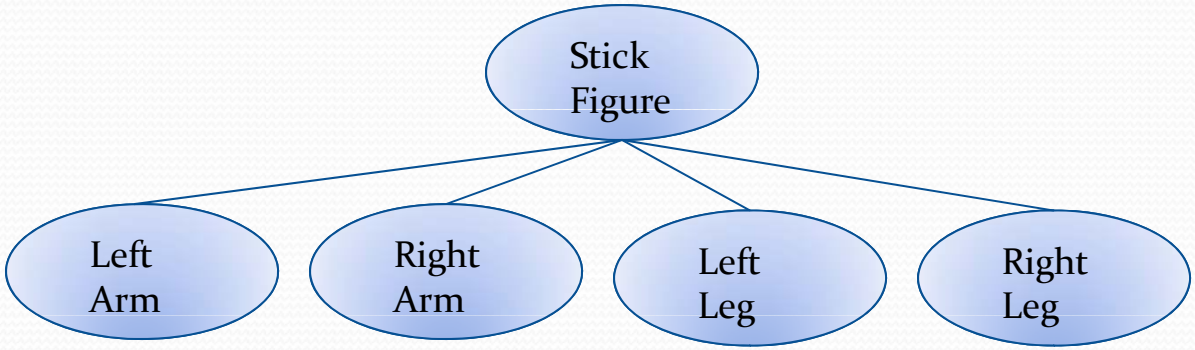
```
myDisplay()
{
   glPushMatrix();
      glTranslatef(some where..);
      glRotatef(some angle…);
      drawAStickFigure(angle);
   glPopMatrix();
   .
   .
   .
   // draw many many
}
```



Imagine that if your program changes the variable “angle” according to time

Hierarchical Transformation

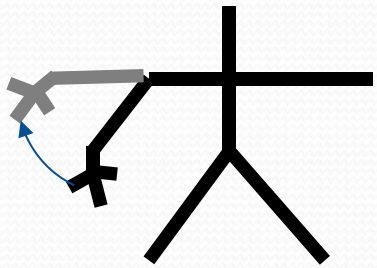
- The figure can be understood as a hierarchy as

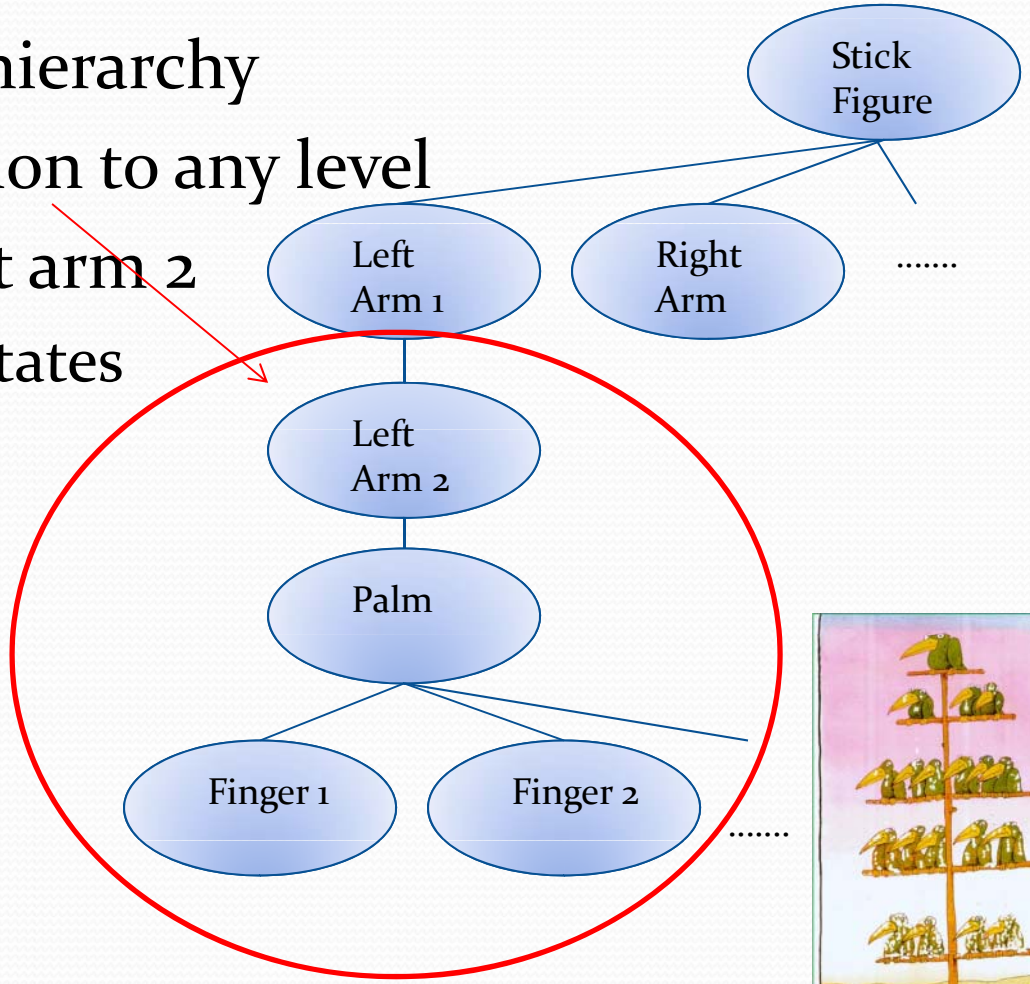


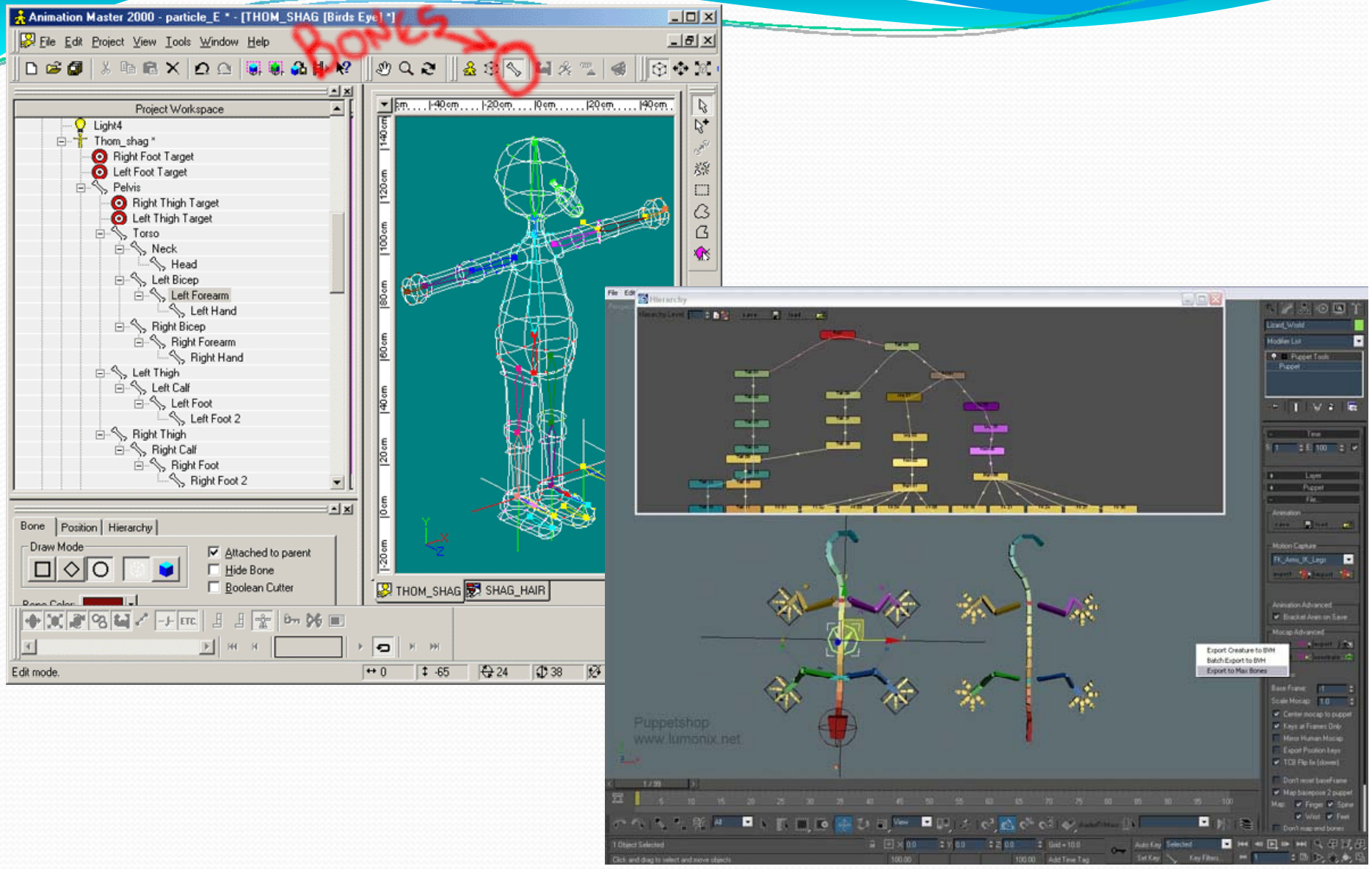
- In which each node is a new reference frame
- When we apply a transformation to a node, all its children in its sub-tree are affected
- E.g. when we translate the stick figure, all the limbs are translated

Hierarchical Transformation

- More deeper levels of hierarchy
- Apply the transformation to any level
- E.g. Rotation at the left arm 2
  - The lower sub-tree rotates





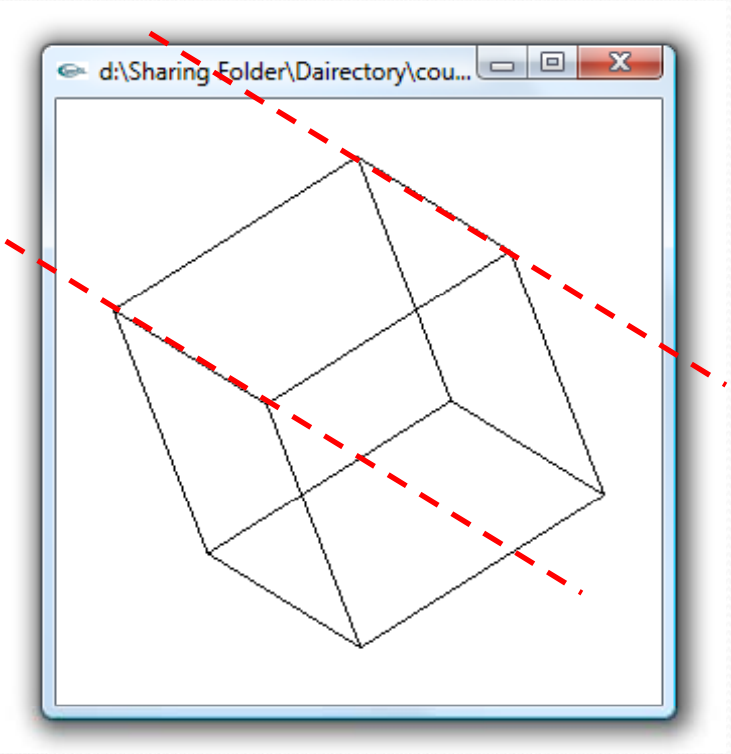

Animation Master 2000 - particle_E * - [THOM_SHAG [Birds Eye] *]
File Edit Project View Tools Window Help
BONES
Project Workspace
Light4
Thom_shag *
Right Foot Target
Left Foot Target
Pelvis
Right Thigh Target
Left Thigh Target
Torso
Neck
Head
Left Bicep
Left Forearm
Left Hand
Right Bicep
Right Forearm
Right Hand
Left Thigh
Left Calf
Left Foot
Left Foot 2
Right Thigh
Right Calf
Right Foot
Right Foot 2
Bone
Position
Hierarchy
Draw Mode
Attached to parent
Hide Bone
Boolean Cutter
THOM_SHAG
SHAG_HAIR
Edit mode.
Export Creature to BVH
Batch Export to BVH
Export to Max Bones
Puppetshop
www.lumonix.net

What you should be able to do now

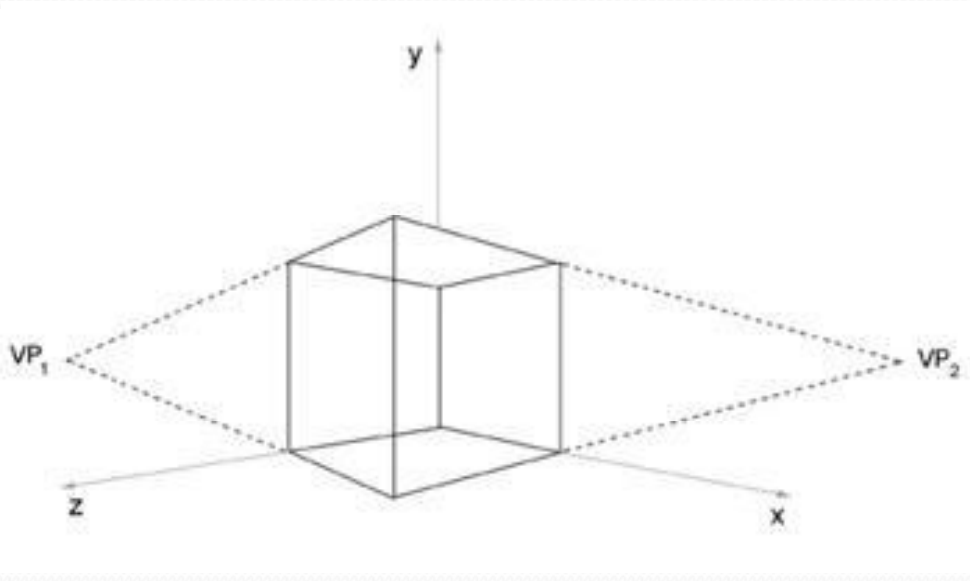
- Romanesco broccoli

Something is not very right?!

- Parallel Projection



- Perspective Projection
  - Next Lecture



Announcement

- No tutorial for this week
- Assignment #1 Deadline 8th Feb 23:59
  - Please do not start working until Friday afternoon!
  - And do NOT submit the zip file until the last minute
  - And do not spend too much time in the assignment
    - You should be able to finish the assignment within 3 hours
  - You can use either MSVS 8.0 or 10.0 or 12.0
    - Please state your version in your readme.txt
    - Or also put it in your file name like

      `U12345678.MSVC8.0.zip`
    - In order to make your tutors happier….

Assignment 1

- Post your picture onto NUSCG facebook page
  - The first 5 have the most “like” will be voted again in the class
  - The first 3 get bonus points
- Late submission
  - 1 day late, 33% off
  - 2 day late, 66% off
  - 3 day late, you submit it for fun
  - Unless you have some serious excuses