COMP 3069
Computer Graphics

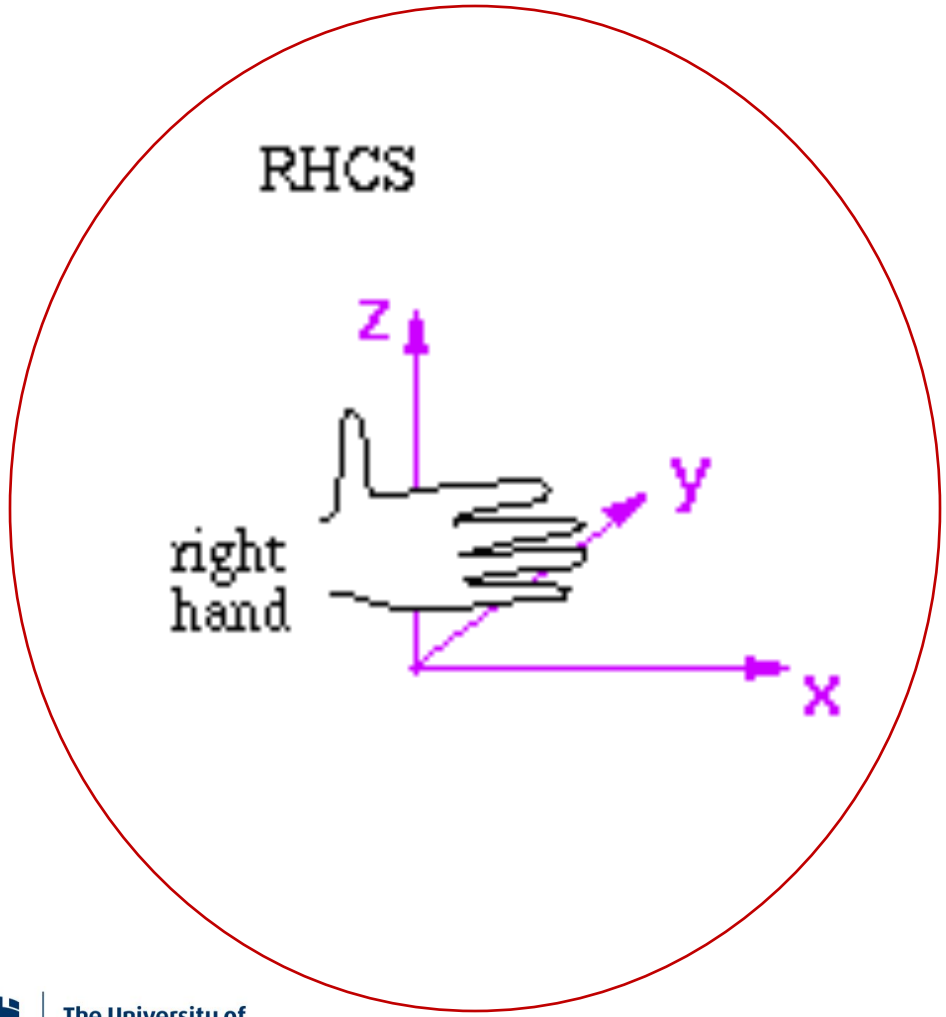
Lecture 3:
Transformation

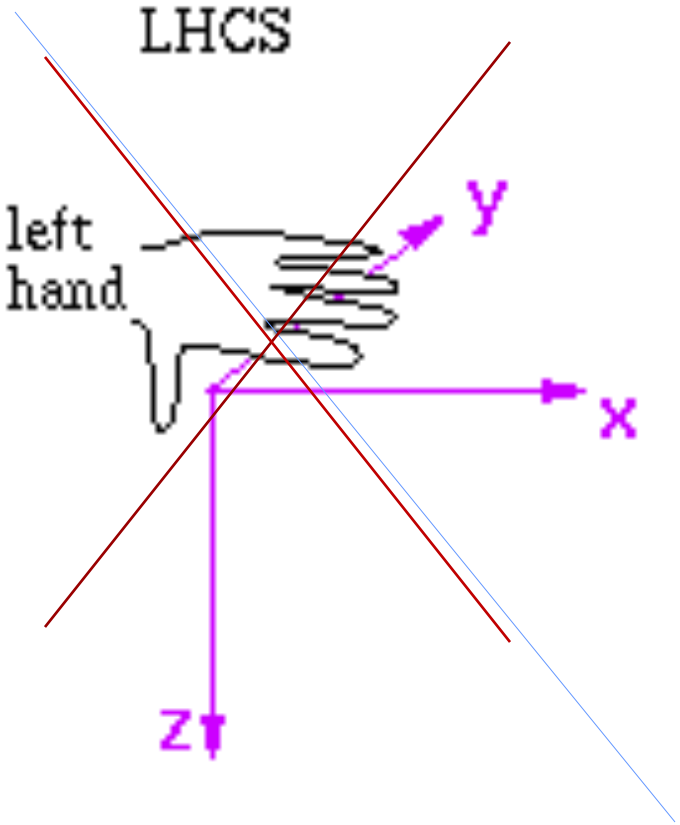
Autumn 2018

The University of Nottingham

UNITED KINGDOM · CHINA · MALAYSIA

Coordinate System (we use)





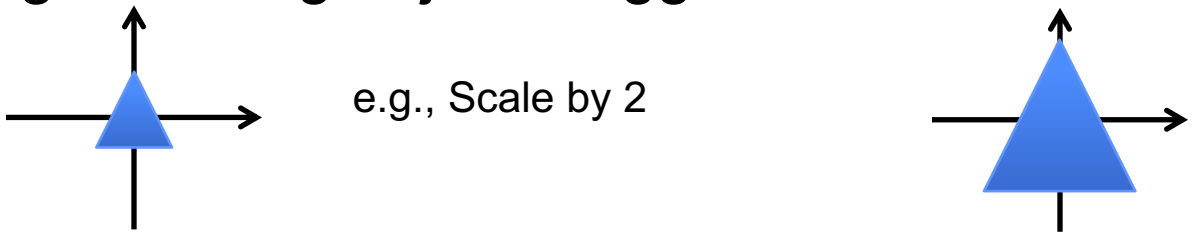
3D Transformation

- In the modeling lecture we saw how to define an object in terms of its polygons/edges/vertices in object space
- We will now see how to use transformations to move objects around
- In computer graphics a transformation is represented by a matrix - to transform an object is to multiply the vertices (represented as vectors) of the object with the matrix
- The vertices of objects are usually represented using **homogeneous** coordinates, which adds an extra component to the vertex vector

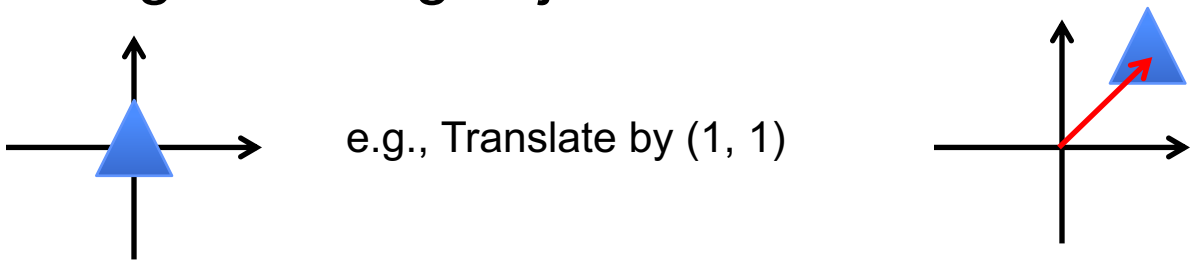
3D Transformations

3 commonly used transformations are:

- Scaling – making objects bigger/smaller



- Translating – moving object around



- Rotating – rotating objects in space

3D Scaling Matrix

- By multiplying a point (x,y,z,1) by a scaling matrix we turn it to ($S_x x$, $S_y y$, $S_z z$,1), all coordinates are scaled

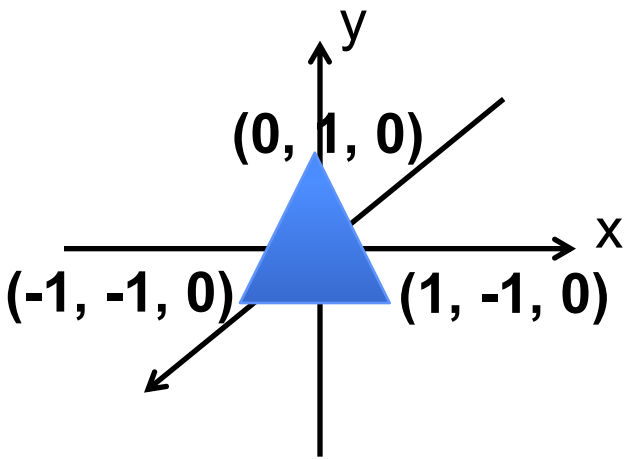
$$\begin{pmatrix} S_x x \\ S_y y \\ S_z z \\ 1 \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

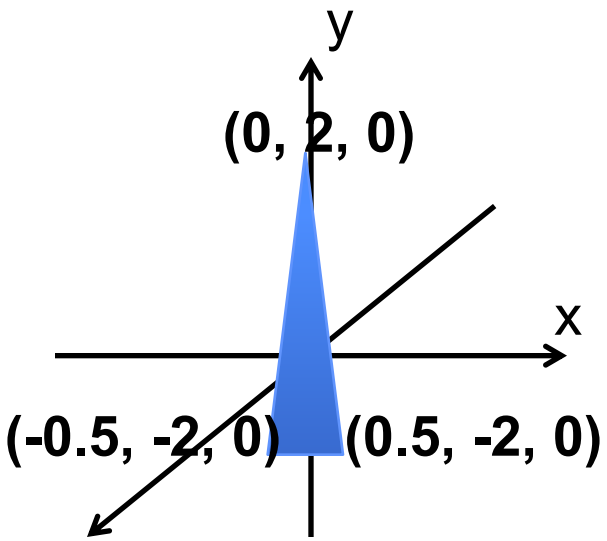
Scaling coefficients in x, y, z directions

- We use homogeneous coordinates. This allows us to represent all transformations (e.g., scale, rotate, translate) using matrices

Scaling Example

- Make the triangle twice as tall but half as wide





$$\begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.5 \\ -2 \\ 0 \\ 1 \end{pmatrix}$$

3D Translation Matrix

- By multiplying a point (x,y,z,1) by the translation matrix we turn it to ($x+T_x$, $y+T_y$, $z+T_z$, 1):

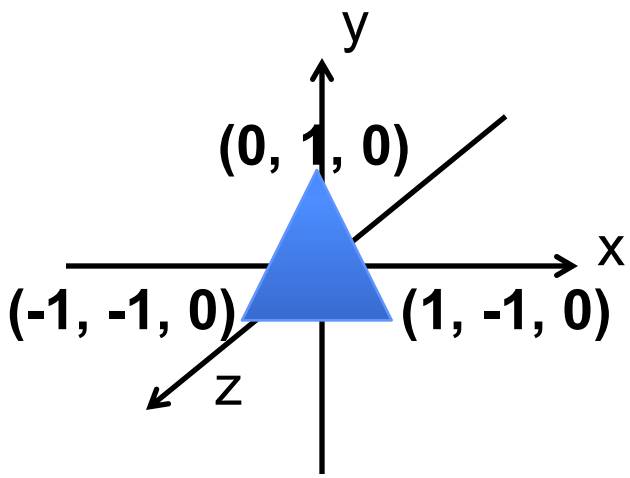
$$\begin{pmatrix} x+T_x \\ y+T_y \\ z+T_z \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

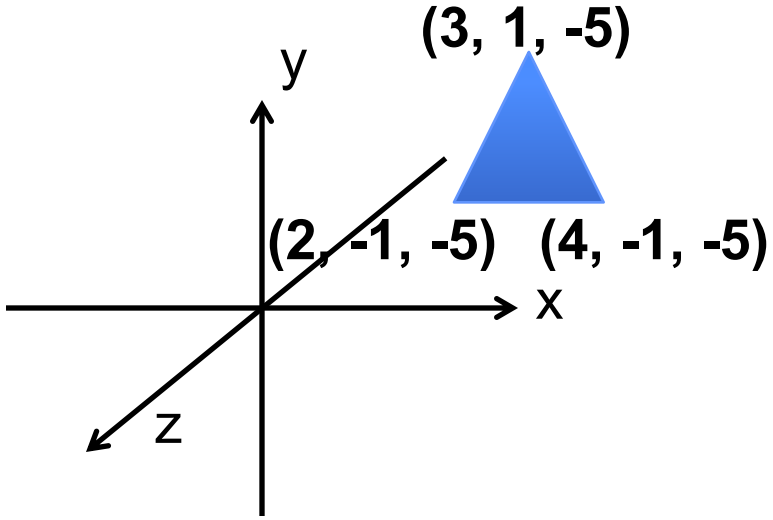
Translation in x, y, z directions

- Note: It won't be possible to use a 3x3 matrix to represent the translation of a vertex; that's why we use a 4x4 homogenous matrix instead!

Translation Example

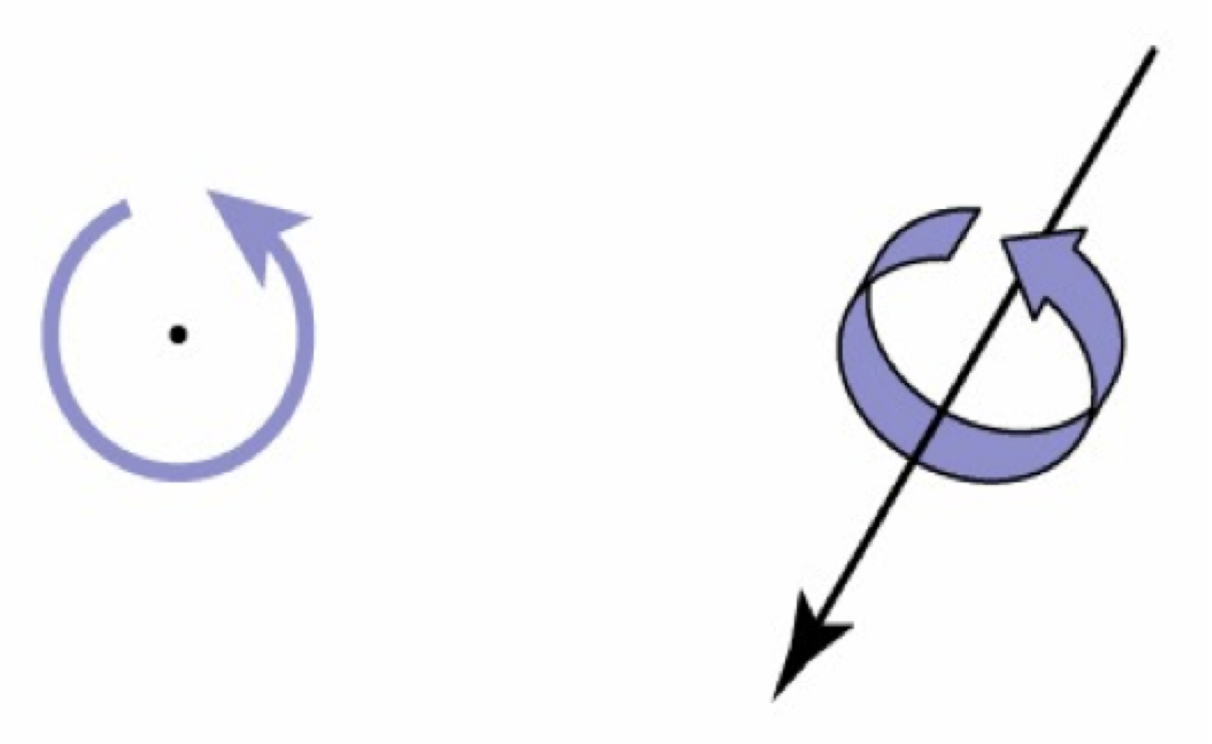
- Move the triangle back by 5 and right by 3





$$\begin{pmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -5 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 4 \\ -1 \\ -5 \\ 1 \end{pmatrix}$$

Rotation in Space



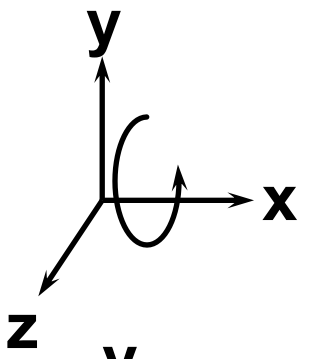
2D

3D

3D Rotation About Coordinate Axes

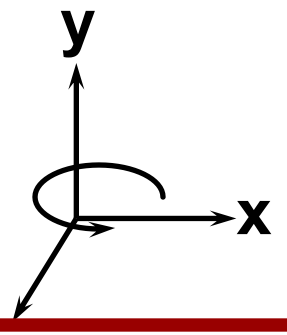
- about X axis

$$\mathbf{R}_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



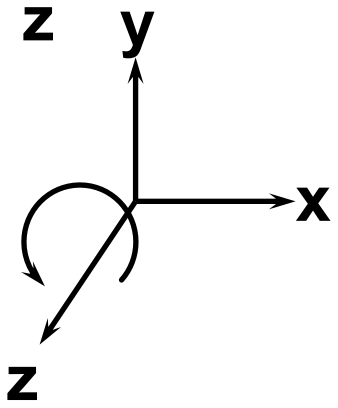
- about Y axis

$$\mathbf{R}_y = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



- about Z axis

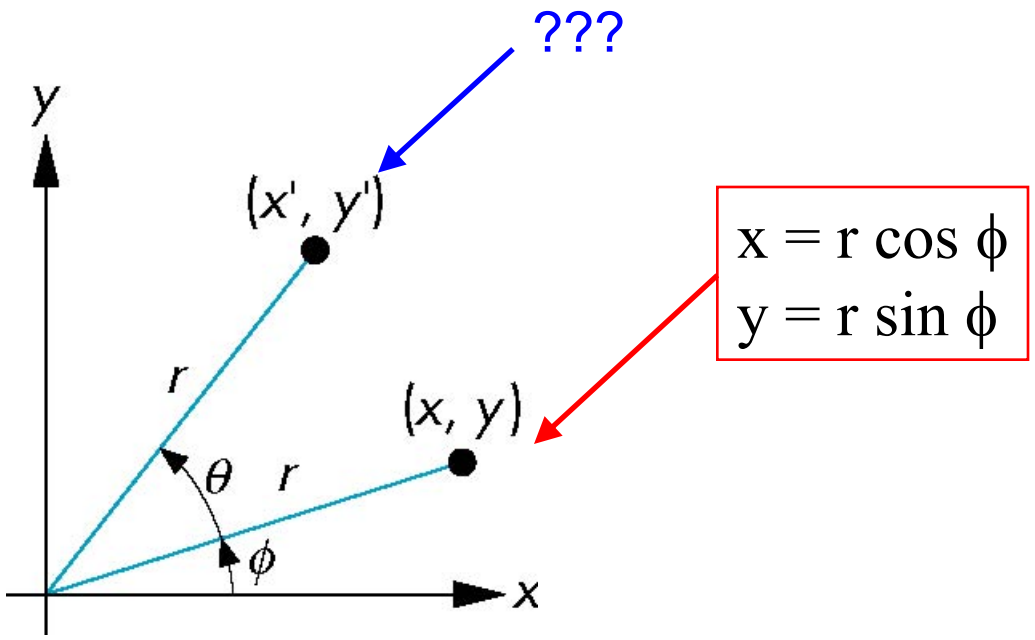
$$\mathbf{R}_z = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Rotation (2D) - Ignoring Z-axis

Consider rotation about the origin by θ degrees

- radius stays the same, angle increases by θ

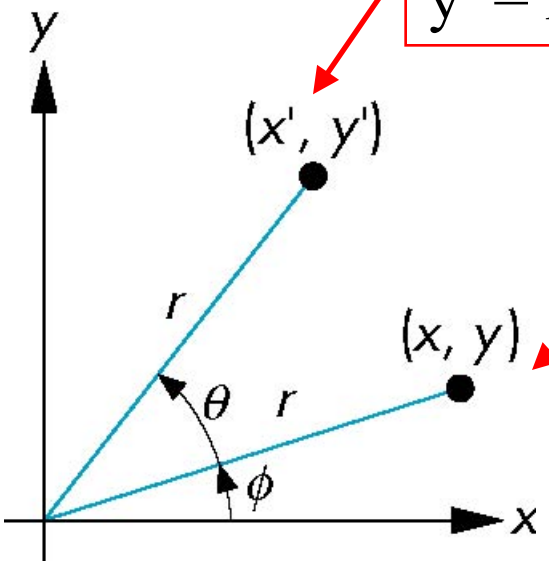


Rotation (2D) - Ignoring Z-axis

Consider rotation about the origin by θ degrees

- radius stays the same, angle increases by θ

$$x' = r\cos(\phi + \theta) = r(\cos(\phi)\cos(\theta) - \sin(\phi)\sin(\theta))$$
$$y' = r\sin(\phi + \theta) = r(\cos(\phi)\sin(\theta) + \sin(\phi)\cos(\theta))$$



$$x = r\cos\phi$$
$$y = r\sin\phi$$

Hence,

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

Rotation about the Z-axis

- Rotation about z axis in three dimensions leaves all points with the same z
  - Equivalent to rotation in two dimensions in planes of constant z

$$x' = x \cos\theta - y \sin\theta$$
$$y' = x \sin\theta + y \cos\theta$$
$$z' = z$$

  - or in homogeneous coordinates

$$\mathbf{p}' = \mathbf{R}_z(\theta)\,\mathbf{p} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{p}$$

Matrix for Rotation about Z-axis

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

Rotation about X and Y axes

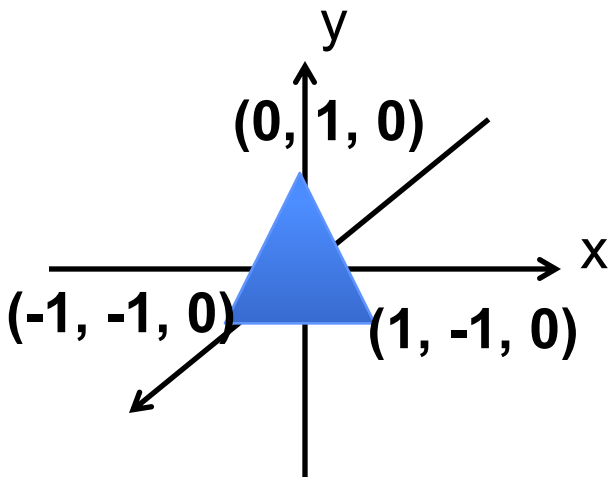
- Same argument as for rotation about *z* axis
  - For rotation about x axis, x is unchanged
  - For rotation about y axis, y is unchanged

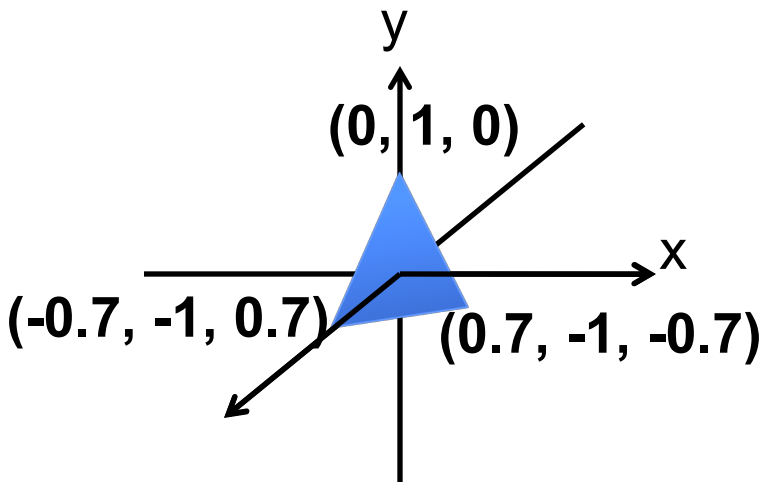
$$\mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

An Example Rotation About Y-Axis

- Rotate by 45°around y axis

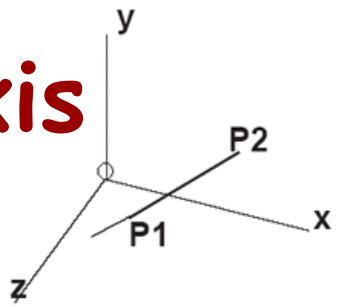




$$\mathbf{R}_y = \begin{pmatrix} \cos 45 & 0 & \sin 45 & 0 \\ 0 & 1 & 0 & 0 \\ -\sin 45 & 0 & \cos 45 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 0.7.. & 0 & 0.7.. & 0 \\ 0 & 1 & 0 & 0 \\ -0.7.. & 0 & 0.7.. & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.7.. \\ -1 \\ -0.7 \\ 1 \end{pmatrix}$$

Rotation about Arbitrary Axis



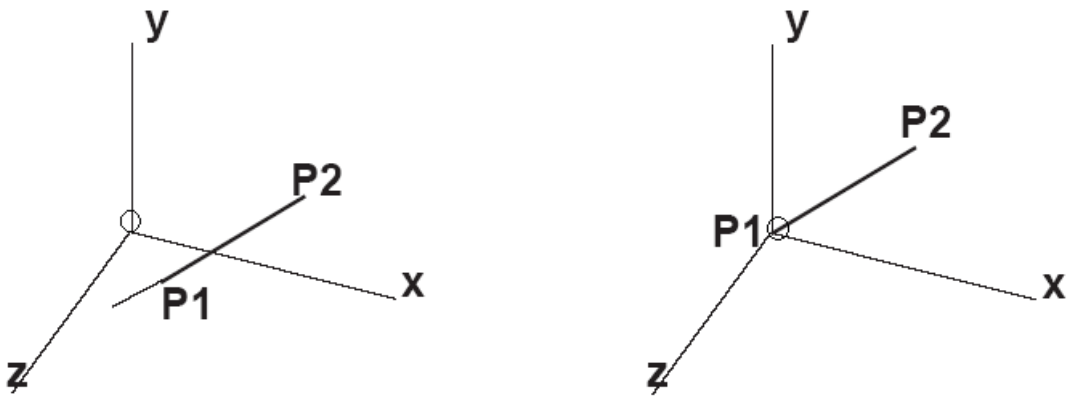
- Often we need rotation about a vector U that is not a coordinate axis (not X, nor Y, nor Z) and may not pass through the coordinate origin

- In this situation we will align U with either X, or Y, or Z, then rotate about this coordinate axis **- any transformation to align U will be applied to all objects in the scene**

- The translation or the rotations done to align U with the coordinate axis need to be reversed to move U back to its initial position **- any transformation to move U back will be applied to all objects in the scene**

Rotation θ about an Arbitrary Axis

- First, translate the vector to be incident the origin:



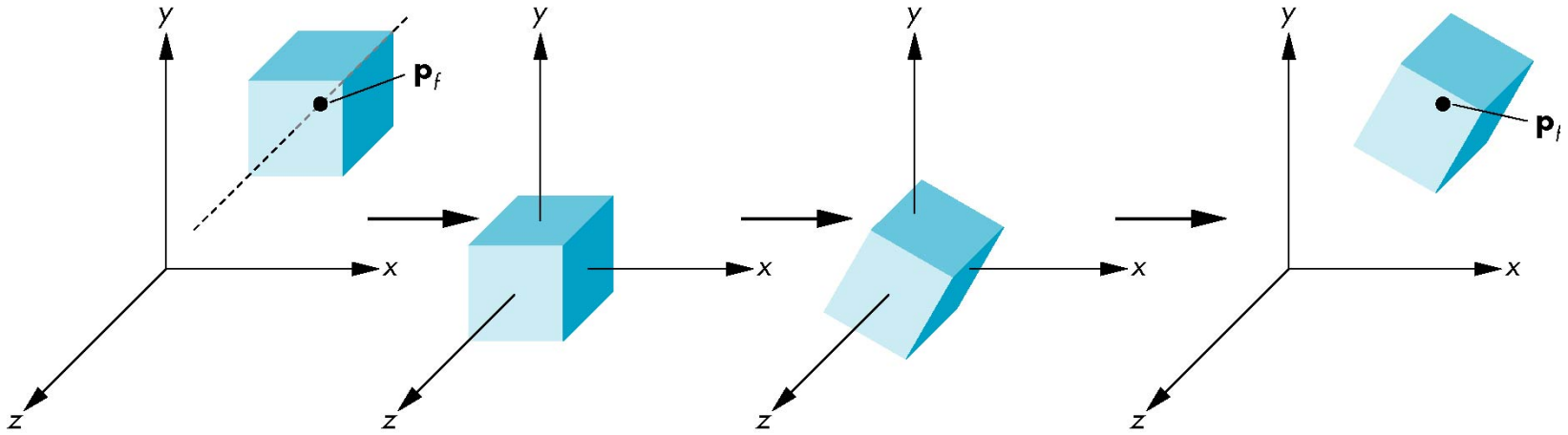
- Rotate about X to bring it to XoZ plane
- Clockwise rotate about Y to align it with Z axis
- Rotate about Z is equivalent to rotation about U
- Undo rotations about Y and X
- Undo the beginning translation

$$\mathbf{T}^{-1}\cdot\mathbf{R}_x^{-1}(\alpha)\cdot\mathbf{R}_y^{-1}(\beta)\cdot\mathbf{R}_z(\theta)\cdot\mathbf{R}_y(\beta)\cdot\mathbf{R}_x(\alpha)\cdot\mathbf{T}$$

Rotation About a Fixed Point other than the Origin

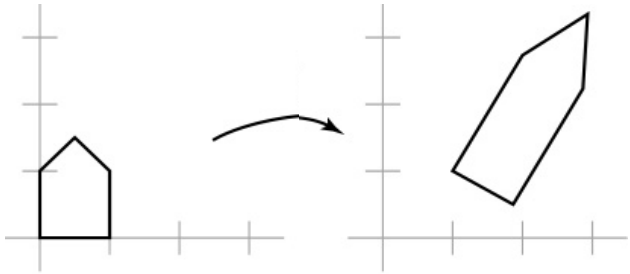
- Move fixed point p_f to origin → $\mathbf{T}(-p_f)$
- Rotate → $\mathbf{R}(\theta)$
- Move fixed point back → $\mathbf{T}(p_f)$

Hence, $\mathbf{M} = \mathbf{T}(p_f)\,\mathbf{R}(\theta)\,\mathbf{T}(-p_f)$

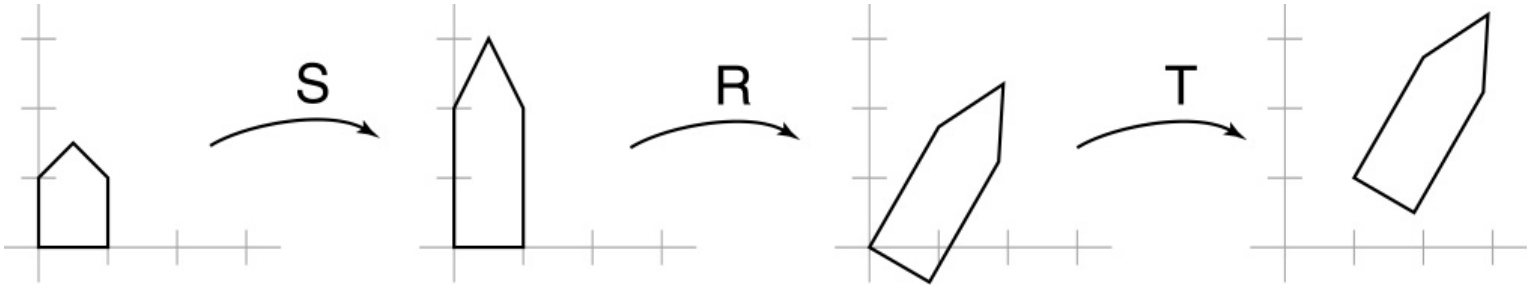


Composite Transformations

- To transform an object like this:



- It will take several transformations, each represented as a matrix:



Composite Transformations

- In matrix form:

$$P' = T_1 * T_2 * \ldots * T_n * P \text{ (vector)}$$

- The last matrix is applied to each vertex (column vector) of the object first (last come first serve)

$$T1 \,.\, T2 \,.\, T3 \ldots Tn \,. \begin{matrix} x \\ y \\ z \end{matrix}$$

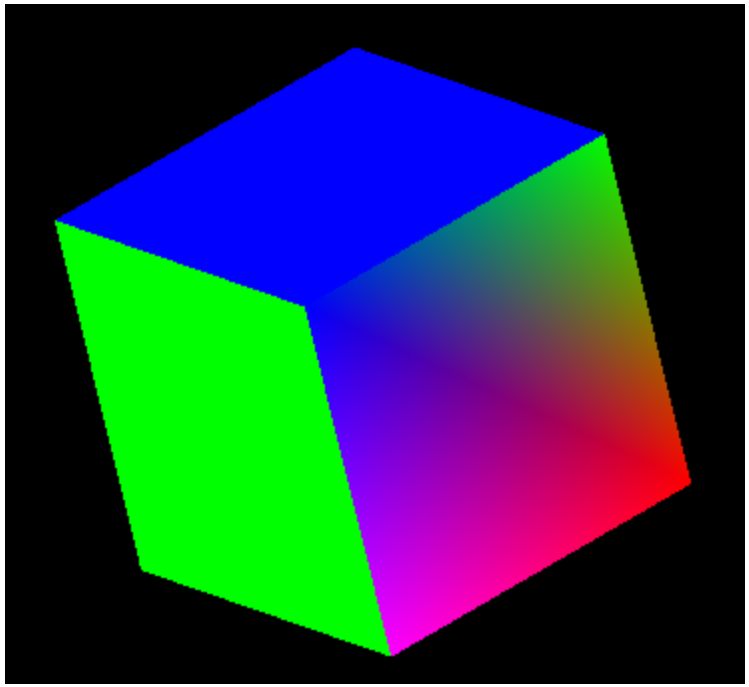
- Can also multiply all transformation matrices to get a composite matrix, and use it to transform the object: **(ABCD)X = (A (B (C (DX))))**
- The order of the transformations is critical,

Transformation in OpenGL

- FRONT - Multicolored
- BACK - White
- TOP - Blue
- BOTTOM - Red
- RIGHT - Purple
- LEFT – Green

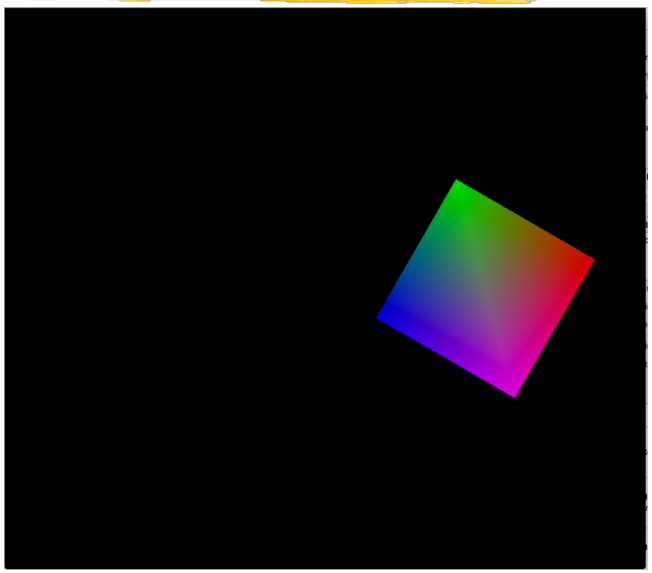
- glTranslatef(0, 0, -500.0);
- glScalef(100, 100, 100);

- glRotated(30, 0.0, 0.0, **1.0**);
- glRotated(30, 0.0, **1.0**, 0.0);
- glRotated(30, **1.0**, 0.0, 0.0);

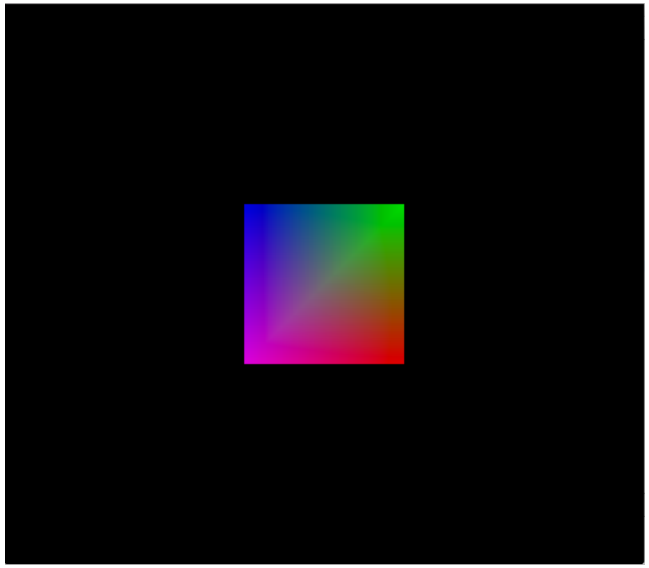


Transformation in OpenGL

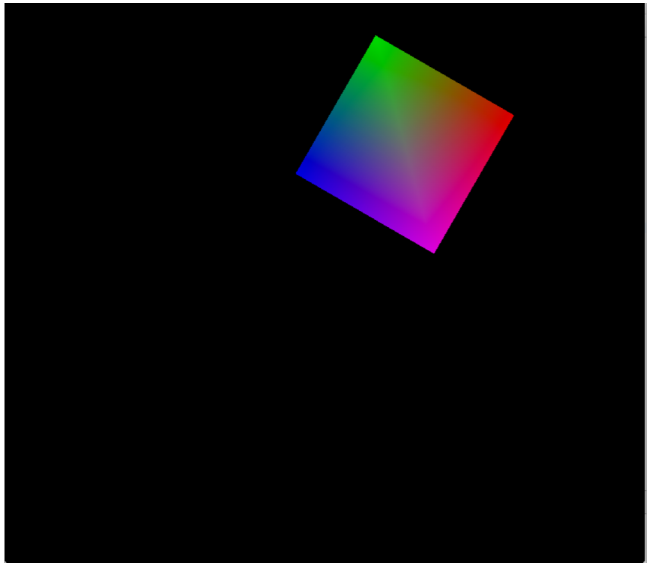
glTranslatef(2.0, 0.0, 0.0);
glRotatef(60.0, 0, 0, 1);



Original



glRotatef(60.0, 0, 0, 1);
glTranslatef(2.0, 0.0, 0.0);



OpenGL Matrix Stacks

- There are 3 **transformation matrix stacks:**
  - Model-View - glMatrixMode(**GL_MODELVIEW**);
  - Projection - glMatrixMode(**GL_PROJECTION**);
  - Texture - glMatrixMode(**GL_TEXTURE**);
- When Model-View mode starts, usually an **identity** matrix is created, which becomes the 'current matrix' and sits on top of the matrix stack.
- When other transformations are called later in the program, the current matrix is updated, e.g., right multiplied by new matrices corresponding to the transformations called.

OpenGL Matrix Stacks

- Say you call glMatrixMode(GL_MODELVIEW), and glLoadIdentity(), OpenGL will create an **identity matrix** (1s on diagonal line 0s on others)
- If you then call glTranslatef(0,0,-250), the current matrix will be

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -250 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -250 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- If you call glTranslatef(0,0,-250) again, current matrix will be:

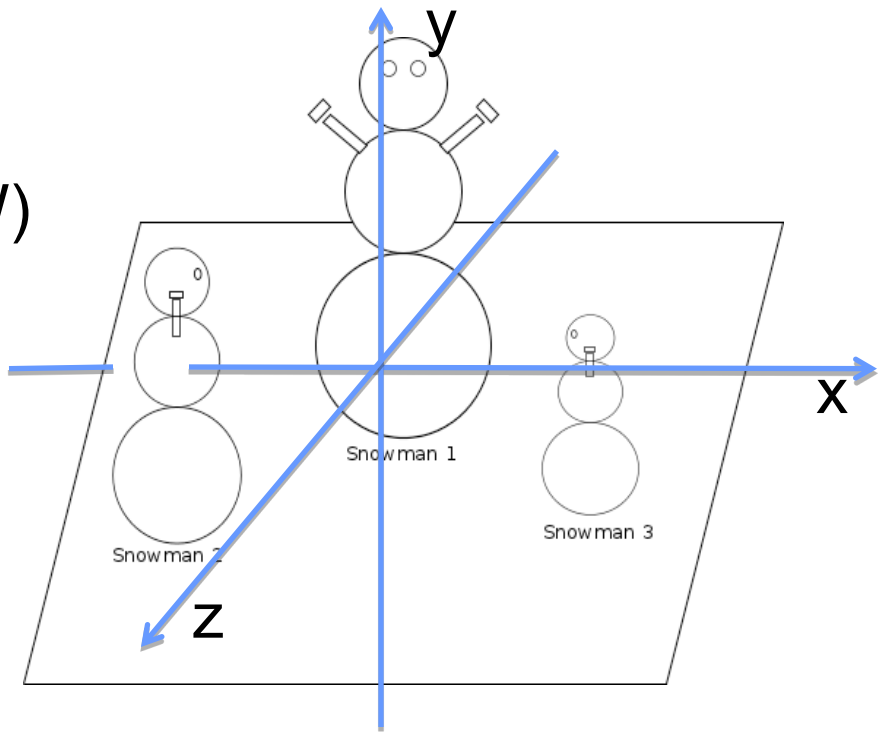
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -250 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -250 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -500 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- If you then call DrawCube() – the cube will be drawn at a distance of 500 from the origin, in the -Z direction

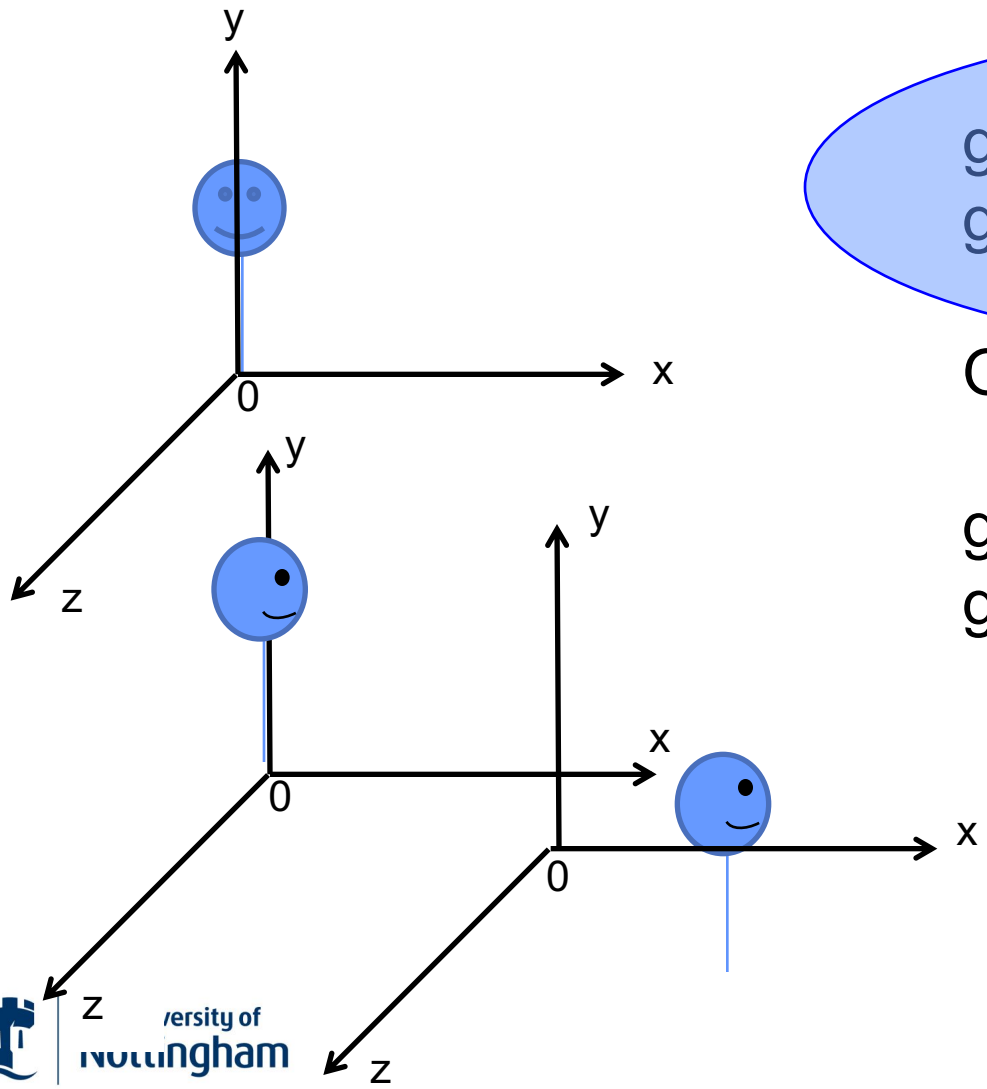
Transformation in OpenGL

- **Snowman 1** is at the coordinate origin, looking along the **z-axis**. Now draw **Snowman 2**, positioned 8 units along the negative x-axis, 6 units along the positive z-axis, half the size of Snowman 1, and looks down the x-axis.

- glMatrixMode(GL_MODELVIEW)
- glTranslatef(-8.0, 0.0, 6.0);
- glRotatef(90.0, 0.0, 1.0, 0.0);
- glScalef(0.5, 0.5, 0.5);
- drawSnowman();



Transformation in OpenGL




```
glTranslatef(5.0, 0.0, 5.0);
glRotatef(90.0, 0.0, 1.0, 0.0);
```

Or

```
glRotatef(90.0, 0.0, 1.0, 0.0);
glTranslatef(5.0, 0.0, 5.0);
```

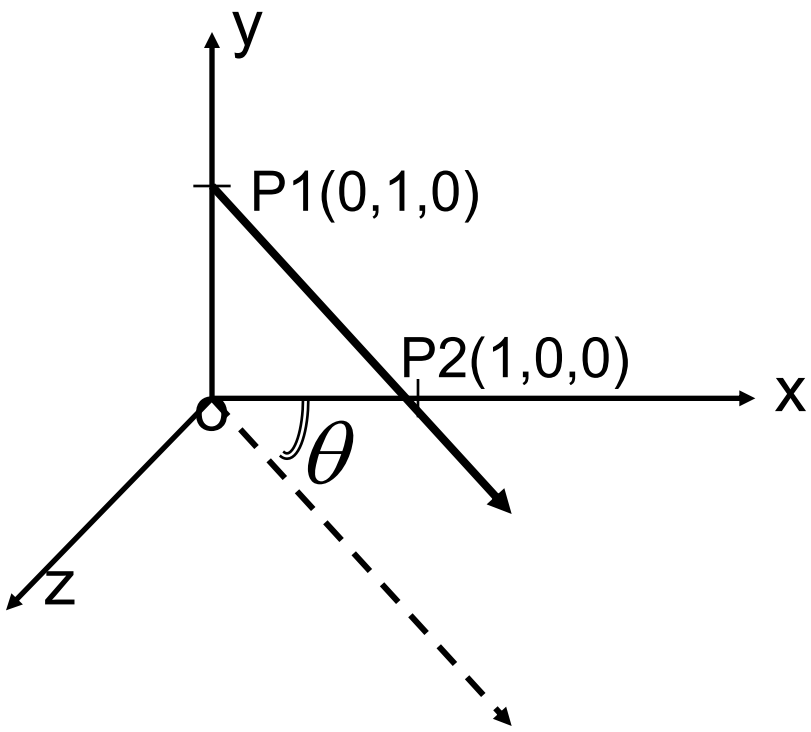
T * R * point

Or R * T * point

Exercise - (Question)

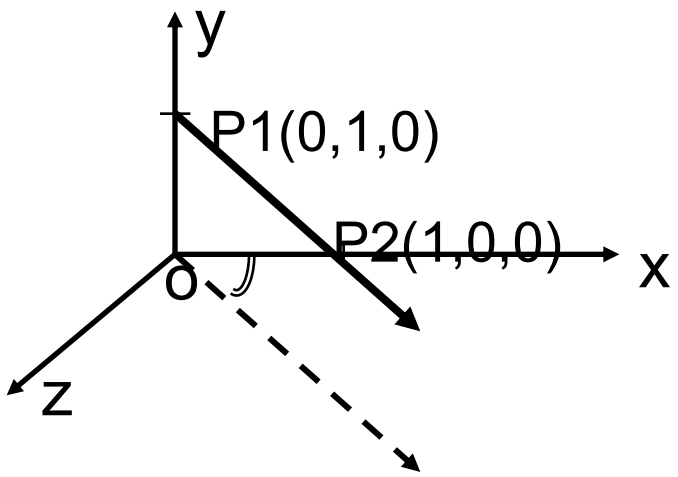
- Write down the transformation that rotates an object by **45 degrees** about vector U
- Vector U originates from P1(0,1,0) on the XoY plane and passes point P2(1,0,0) on the X axis

Question: What's θ, and rotation should be about which axis?



Exercise – (Solution)

- P1 and P2 are on XoY plane, so vector P1P2 is on XoY plane
- After translation to the origin vector P1'P2' is still on the XoY plane, and can be rotated by 45 around Z-axis to align with X-axis
- The composite matrix for rotation around P1P2 is



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(-45) & -\sin(-45) & 0 & 0 \\ \sin(-45) & \cos(-45) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos 45 & -\sin 45 & 0 \\ 0 & \sin 45 & \cos 45 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 45 & -\sin 45 & 0 & 0 \\ \sin 45 & \cos 45 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- translate to move the end of the vector to origin,
- rotate about z to align the vector with x, then rotate about x,
- undo rotation about z-axis and undo the translation