

3D Viewing Transformations

Eric C. McCreath

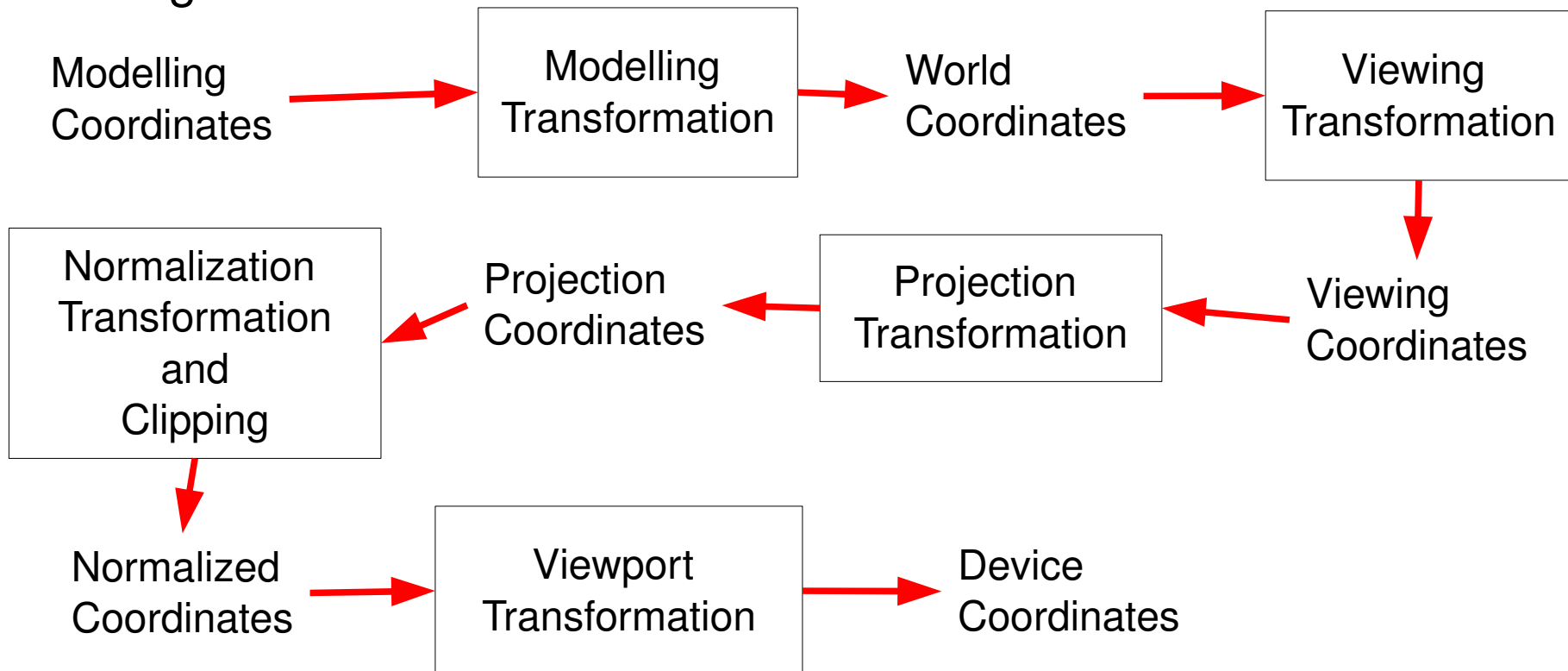
School of Computer Science
The Australian National University
ACT 0200 Australia

ericm@cs.anu.edu.au

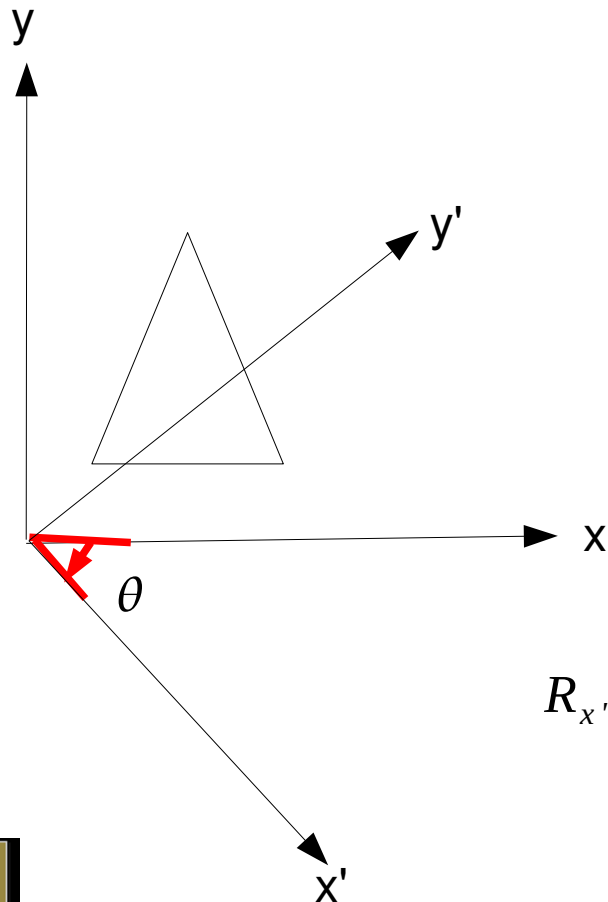
- 3D Matrix Transformations
- Model/World/Viewing/Project/Viewport Transformations
- Orthogonal Projects
- Perspective Projects
- OpenGL and the transformation matrix stack

Mostly from Chapter 7 of the textbook.

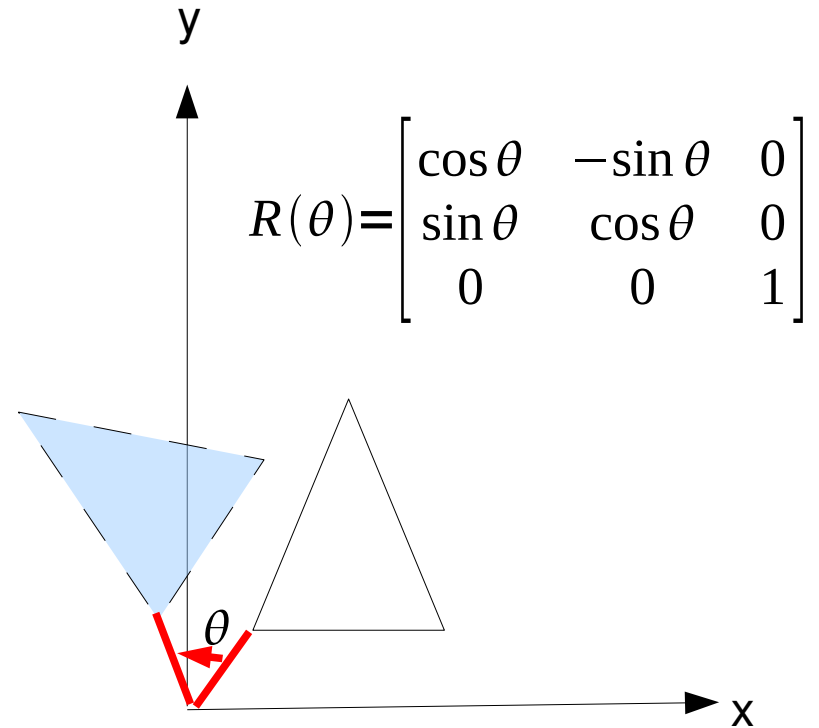
- Viewing processes for a 3D scene are in many ways similar to that of 2D. However, the extra dimension brings with it a host of complexities such as lighting and viewing projections.
- Coordinates undergo a series of transformations to produce the final image on the screen.



- 2D rotation can also be viewed as a change in basis vectors.



$$R_{x',y'} = \begin{bmatrix} x'_x & x'_y & 0 \\ y'_x & y'_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



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

- Translation and scaling in 3D are very similar to that of 2D.

Translation

$$T(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling

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

- 3D rotations can be specified by a sequence of rotations about the axes. (Pitch, Roll, and Yaw)

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

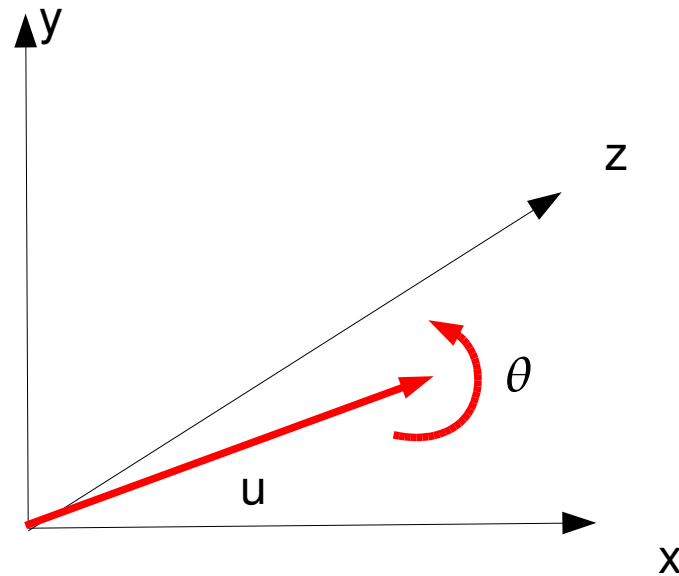
$$R_y(\theta) = \begin{bmatrix} \cos \theta & 0 & -\sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

Note – the inverse of a rotation matrix is its transpose.

$$R^{-1} = R^T$$

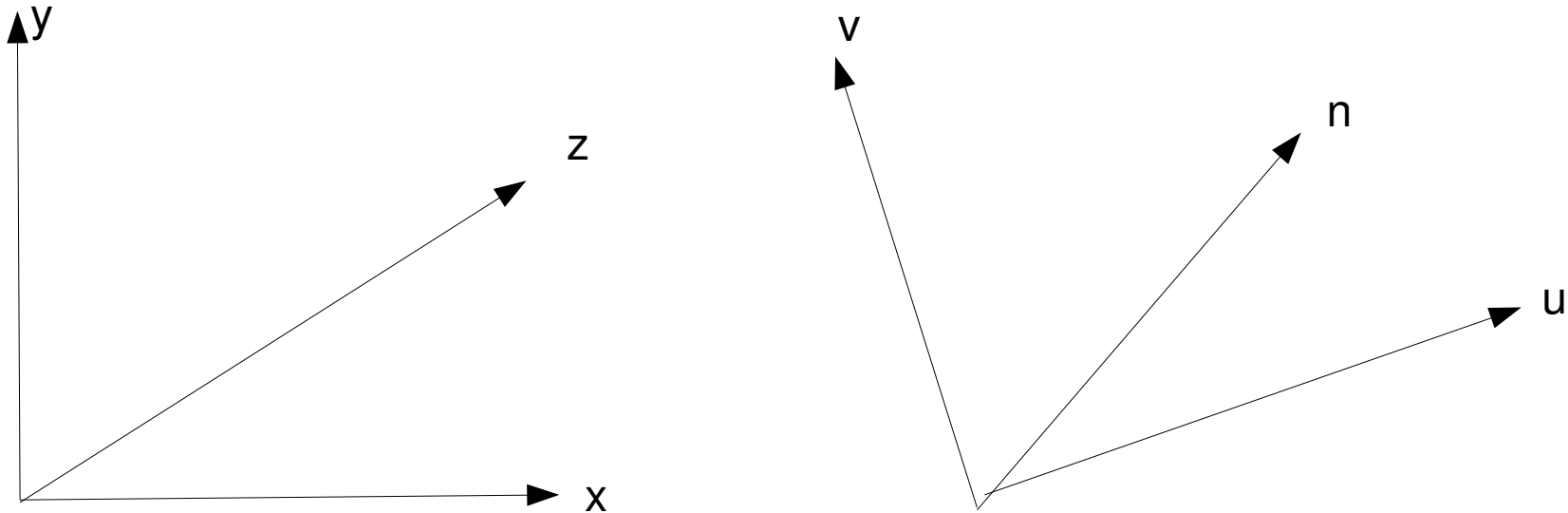
- 3D Rotations can be also specified in terms of a unit vector along with an angle of rotation.



$$R_u(\theta) = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} \sin \theta + I \cos \theta + uu^T (1 - \cos \theta)$$

$$u = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

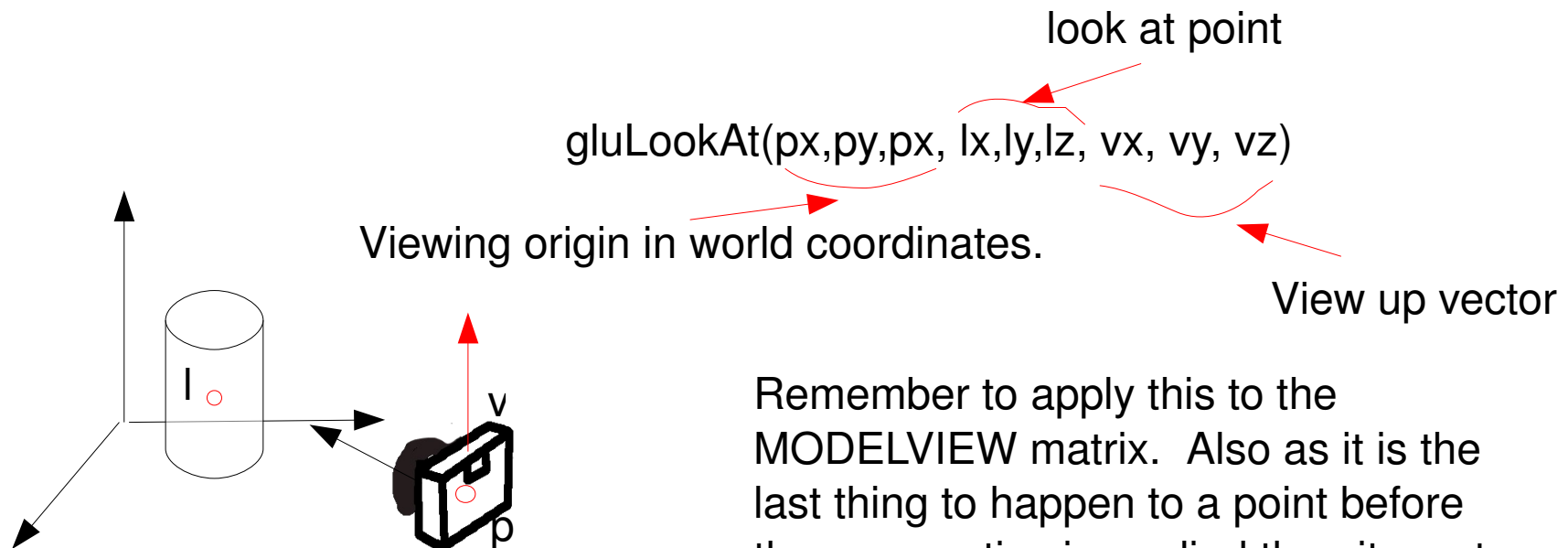
- We can also think of 3D rotations in terms of a change of basis.



$$R_{uvn} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

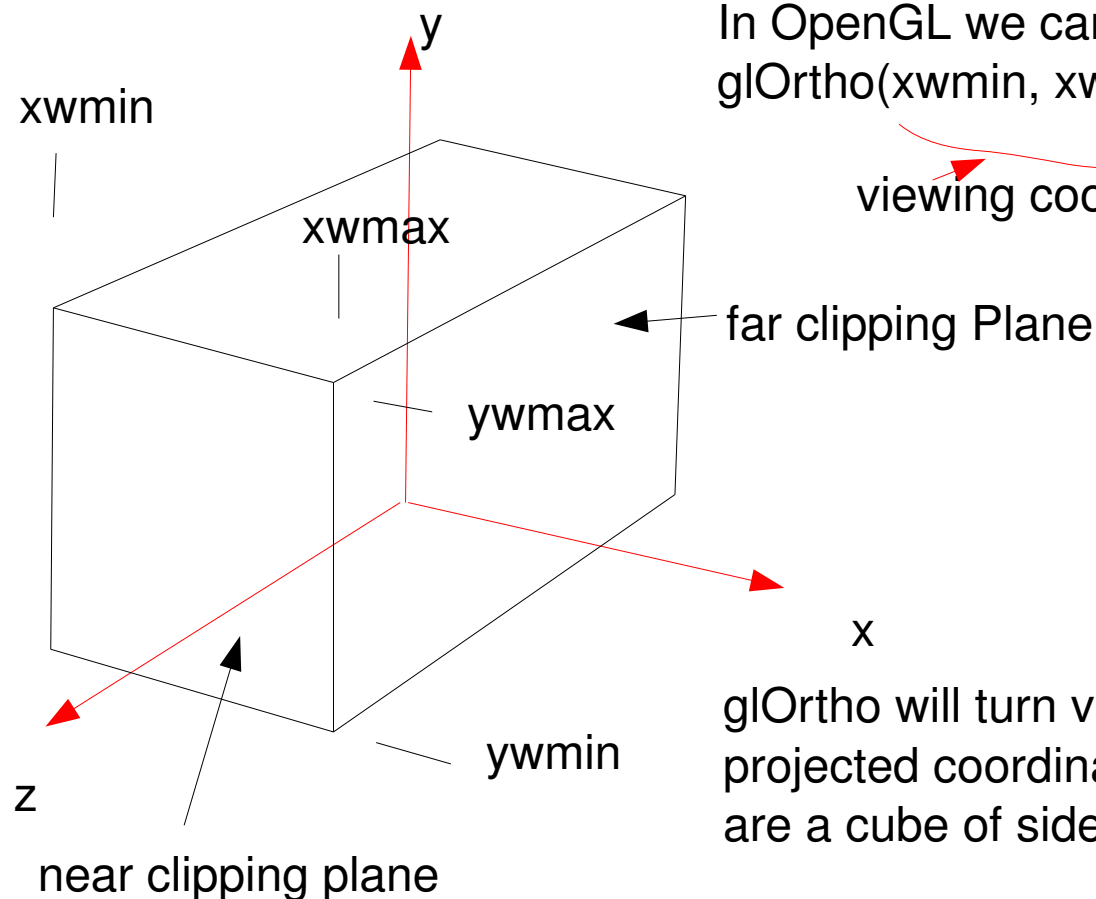
These are the unit vectors of the new basis' in the old basis' dimensions.

- Before any perspective transformations are applied to a scene world coordinates are transformed into viewing coordinates.
- This is done via a translation and then a rotation.
- OpenGL provides a method for doing this:



Remember to apply this to the MODELVIEW matrix. Also as it is the last thing to happen to a point before the perspective is applied then it must be the first operation you apply to the MODELVIEW matrix.

- Orthogonal projections are parallel projections in which objects appear the same size as their distance from the viewer changes.
- The view volume forms a box shape.



In OpenGL we can apply to the PROJECTION matrix:
`glOrtho(xwmin, xmax, ywmin, ymax, dnear, dfar)`

viewing coordinates

The scene is viewed in the negative z direction. Also dnear and dfar denote distances from the origin in this negative z direction.

glOrtho will turn viewing coordinates into normalised projected coordinates. These normalised coordinates are a cube of side length 2 centred on the origin.

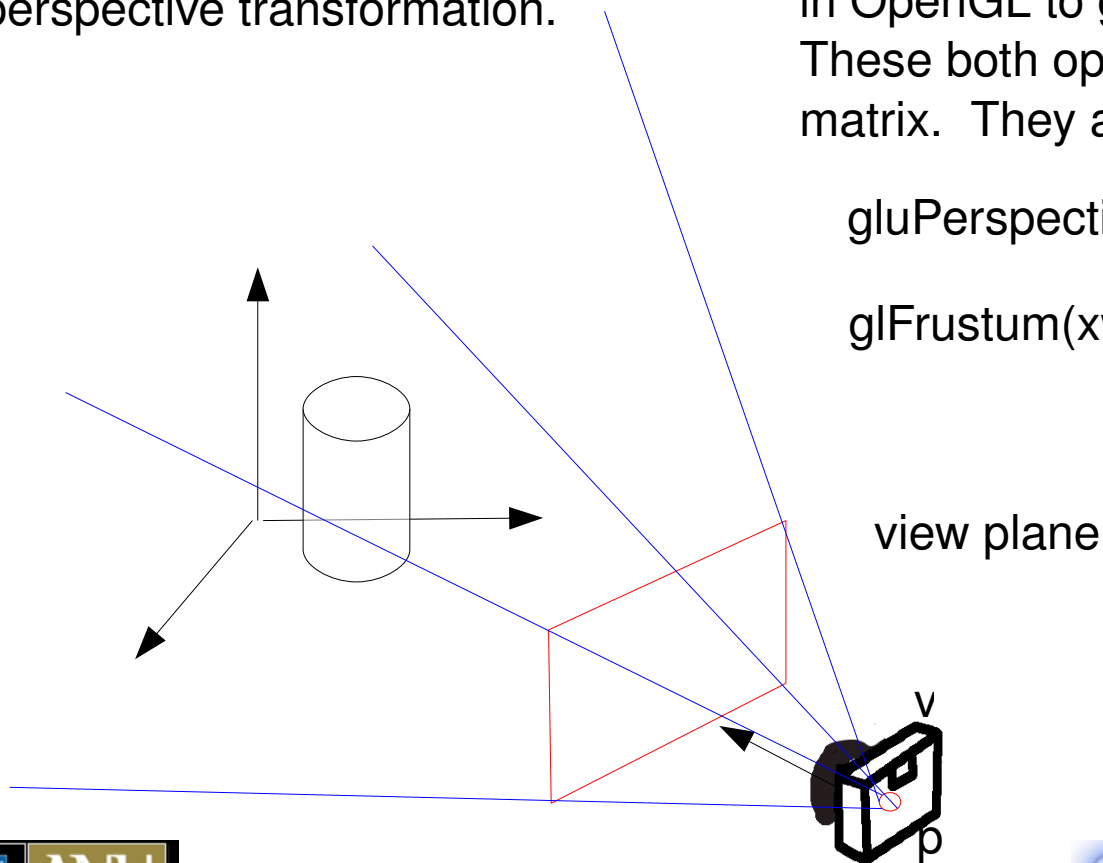
- Perspective projections provide a more realistic view of a scene.

A homogeneous matrix can be used to describe a perspective transformation.

There are two functions that can be used in OpenGL to give a perspective projection. These both operate on the PERSPECTIVE matrix. They are:

```
gluPerspective(theta, aspect, dnear, dfar);
```

```
glFrustum(xwmin, xwmax, ywmin, yzmax, dnear, dfar);
```



- The perspective transformations will generally leave coordinates in a 3D normalized box. These need to be mapped into screen coordinates. Information for each xy position can be stored in a color and depth buffer.
- In OpenGL this can be set with:
`glViewport(xvmin, yvmin, vpWidth, vpHeight)`
- By default the viewport is set to the size and position of the current display window.