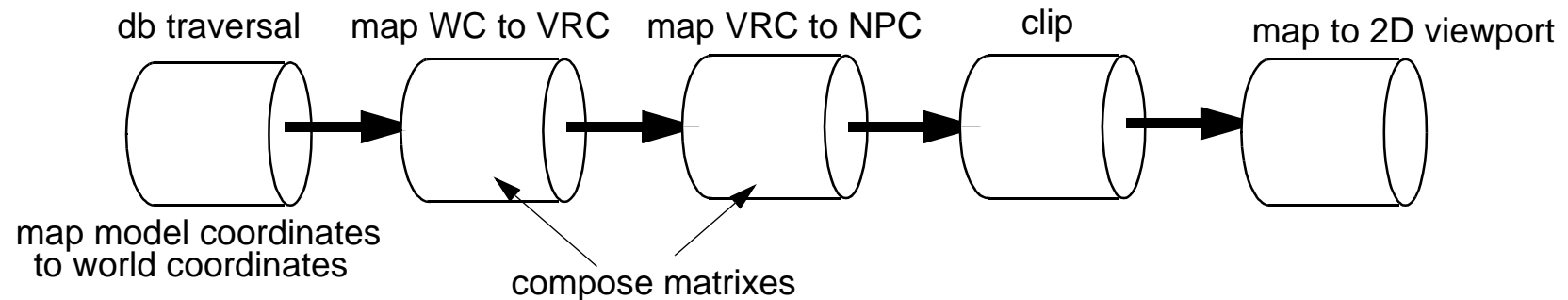


## 6 The Viewing Pipeline

- Viewing Transformations
- The Problem
- View Reference Coordinates
- Parallel Projections
- Perspective Projections
- Viewport Transformations
- OpenGL example code

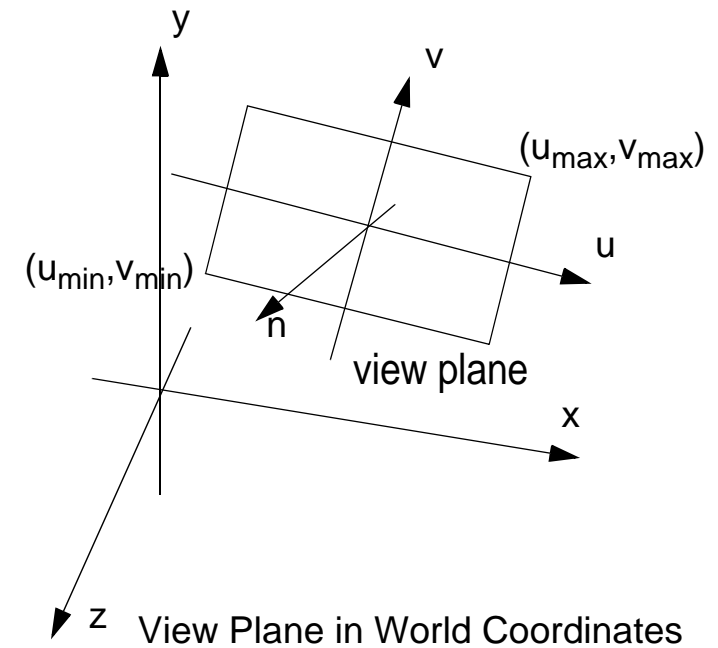
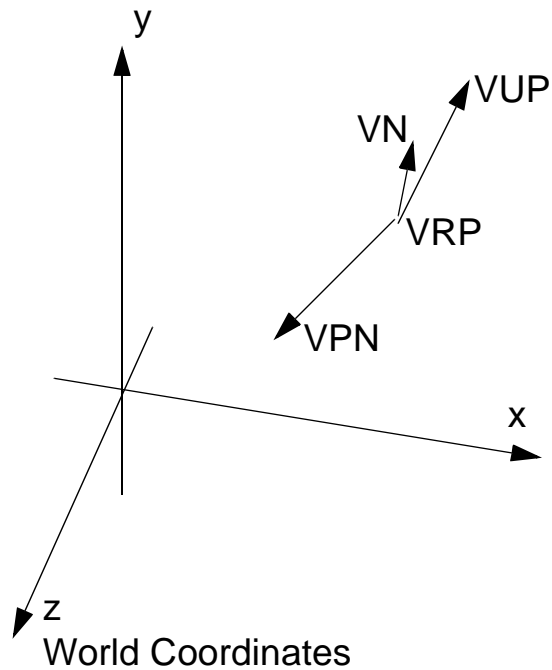
## Viewing Transformations

- viewing pipeline: mapping view volume to device (screen) coordinates
  - derive transformation matrix
  - two cases: parallel and perspective



- WC: world coordinates
- VRC: view reference coordinates
- NPC: normalised projection coordinates used in the canonical view volume (CVV)
- each mapping corresponds to matrix
- matrixes may be combined to accelerate process

## Overview of the Problem



- VRP: View Reference Point
- VN: View Normal
- VUP: View up vector
- VPN: View Plane Normal

$$\mathbf{n} \parallel \mathbf{VPN}$$

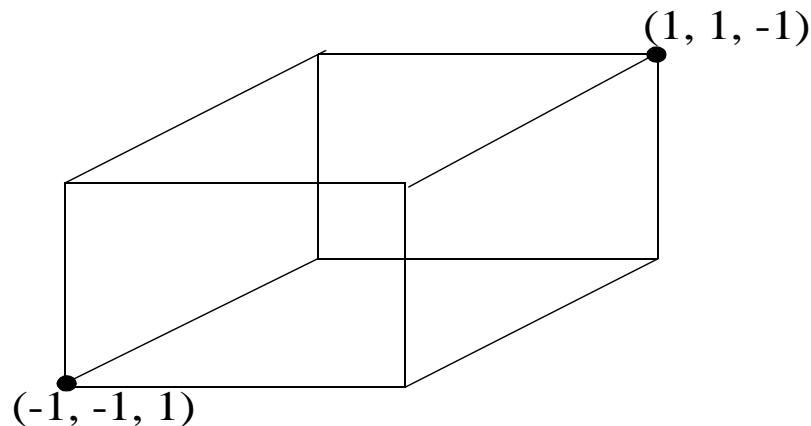
$$\mathbf{v} \parallel \mathbf{VN}$$

$$\mathbf{u} \parallel \mathbf{v} \times \mathbf{n}$$

## Overview of the Problem (cont)

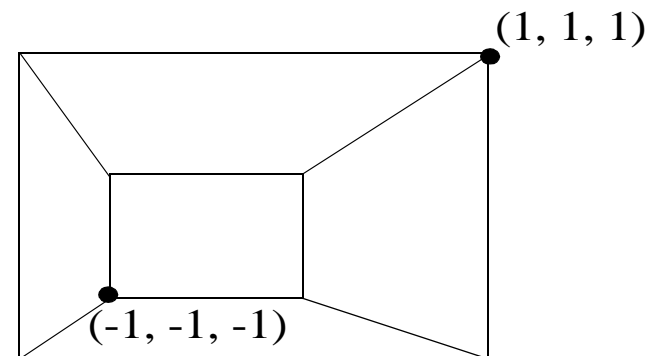
We are given a system of world coordinates in which are defined the view reference point, the view normal and the view up vector. In the view reference coordinates we are given: the projection reference point; the location of the window in the view plane; and the distance of the front and back clipping planes from the projection reference point.

We wish to find transformations that yield a coordinate system such that the front bottom left is at  $(-1, -1, -1)$  and the back top right is at  $(1, 1, 1)$ , called a canonical view volume (CVV).



Parallel Projection

(Right handed coordinates)



Perspective Projection

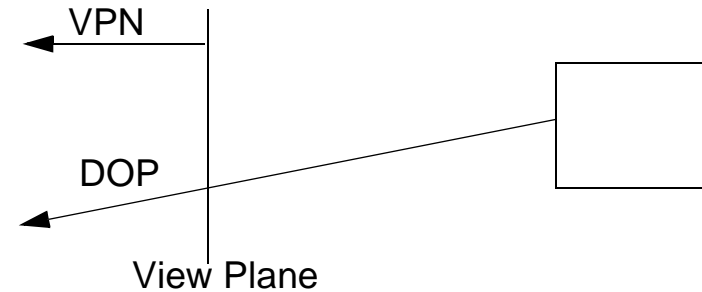
(Left handed coordinates)

## Orthographic & Oblique Projections (Reminder)



### Orthographic

View plane normal parallel to the direction of projection



### Oblique

View plane normal not parallel to the direction of projection.

Not implemented by OpenGL, do yourself using `glMulMatrix` function.

## View Reference Coordinates 1: Translate VRP to Origin

The first stage of transforming the world coordinate system from world coordinates to view reference coordinates is a translation of the view reference point to the origin of the world system. The translation can then be applied to each model.

$$T = \begin{bmatrix} 1 & 0 & 0 & -\text{VRP}_x \\ 0 & 1 & 0 & -\text{VRP}_y \\ 0 & 0 & 1 & -\text{VRP}_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

## View Reference Coordinates 2: Rotate (u, v, n) into (x, y, z)

The second stage of the transformation is to rotate the axes of the view reference coordinate system into the axes of the world coordinate system. Applying this transformation after the translation in the previous slide allows the models to be transformed into view reference coordinates.

First determine the **u**, **v** and **n** in world coordinates.

$$\mathbf{n} = \frac{\mathbf{V} \cdot \mathbf{P} \cdot \mathbf{N}}{\|\mathbf{V} \cdot \mathbf{P} \cdot \mathbf{N}\|} \quad \mathbf{u} = \frac{\mathbf{V} \cdot \mathbf{U} \cdot \mathbf{P} \cdot \mathbf{N}}{\|\mathbf{V} \cdot \mathbf{U} \cdot \mathbf{P} \cdot \mathbf{N}\|} \quad \mathbf{v} = \mathbf{n} \times \mathbf{u}$$

Form the rows of the rotation matrix out of the components of the 3 vectors.

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

## View Reference Coordinates: OpenGL Commands

In OpenGL the view reference coordinates are part of the ‘model view’ matrix. There are 2 ways of setting up the coordinates.

In GL the commands for translation and rotation can be used:

```
void glTranslatef(GLfloat x, GLfloat y, GLfloat z)
void glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z)
```

Alternatively in the utility toolkit GLU there is:

```
void gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez,
               GLdouble centerx, GLdouble centery, GLdouble centerz,
               GLdouble upx, GLdouble upy, GLdouble upz)
```

This eye coordinates specify a look-from point, the centre coordinates a look-at point and the up coordinates the head-up vector.

The Matrix stack currently used is chosen with:

```
void glMatrixMode()
```

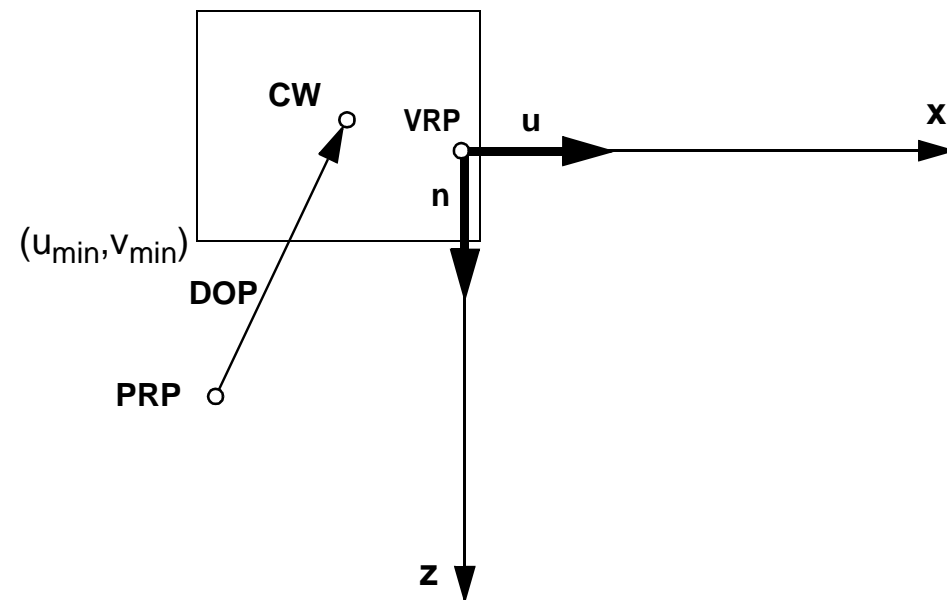
## Parallel Projection 1: Bring DOP parallel to Z (if oblique projection)

First the direction of projection (DOP) must be made to lie parallel with the **n** axis. This is achieved by a shear.

CW = Center of Window

$$DOP = CW - PRP = \begin{bmatrix} DOP_u \\ DOP_v \\ DOP_n \end{bmatrix}$$

$$M_0 = \begin{bmatrix} 1 & 0 & DOP_u / DOP_n & 0 \\ 0 & 1 & DOP_v / DOP_n & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Note if the projection is orthogonal, then the PRP is on the **n** axis and the shear is zero.

## Parallel Projection2: Translate Centre of Front Clipping Plane to Origin

This is a simple translation of the window centre to the z axis.

$$T = \begin{bmatrix} 1 & 0 & 0 & -\frac{(u_{\max} + u_{\min})}{2} \\ 0 & 1 & 0 & -\frac{(v_{\max} + v_{\min})}{2} \\ 0 & 0 & 1 & -\frac{(n_{\max} + n_{\min})}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

## Parallel Projection: Scale into Canonical View Volume

The final stage is to scale the coordinates so that the volume extends from  $(-1, -1, 1)$  to  $(1, 1, -1)$ .

$$S = \begin{bmatrix} \frac{2}{u_{\max} - u_{\min}} & 0 & 0 & 0 \\ 0 & \frac{2}{v_{\max} - v_{\min}} & 0 & 0 \\ 0 & 0 & \frac{-2}{n_{\max} - n_{\min}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The x, y and z axis are scaled by half the width and height and depth respectively.

## The OpenGL Commands

OpenGL only implements orthographic projections, if you need other oblique projections you must form the  $M_0$  matrix yourself and use `glMultMatrix()` to modify the projection matrix.

GL provides the function:

```
void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)
```

The matrix it produces is:

$$M = ST = \begin{bmatrix} \frac{2}{u_{\max} - u_{\min}} & 0 & 0 & \frac{-(u_{\max} + u_{\min})}{u_{\max} - u_{\min}} \\ 0 & \frac{2}{v_{\max} - v_{\min}} & 0 & \frac{-(v_{\max} + v_{\min})}{v_{\max} - v_{\min}} \\ 0 & 0 & \frac{-2}{n_{\max} - n_{\min}} & \frac{-(n_{\max} + n_{\min})}{n_{\max} - n_{\min}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

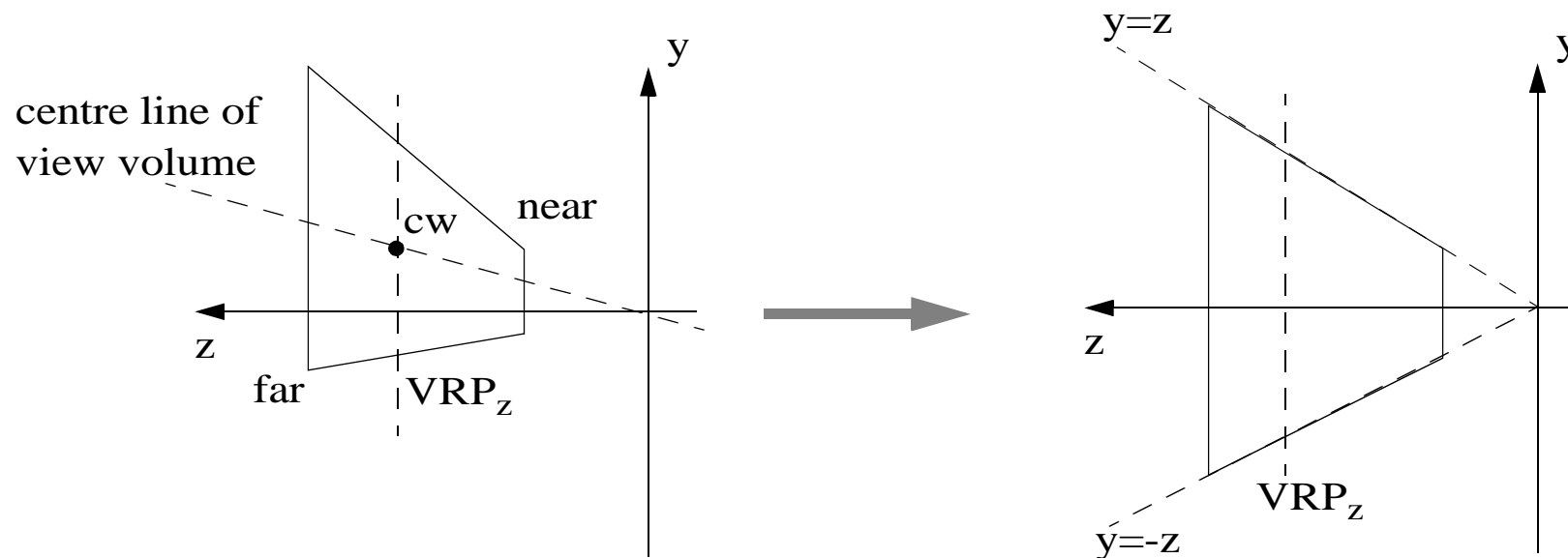
## Parallel Projection: OpenGL Example

The following code sets up an orthogonal projection and then returns to model view mode. The `glLoadIdentity()` function is needed to replace the existing matrix with an identity matrix as the `glOrtho()` function premultiplies what ever is at the top of the matrix stack.

```
glMatrixMode(GL_PROJECTION);
    glPushMatrix();          /* optional, preserves a previous matrix */
    glLoadIdentity();       /* sets top of stack to id matrix */
    glOrtho(Xmin, Xmax, Ymin, Ymax, nearClip, farClip);
glMatrixMode(GL_MODELVIEW);
```

## Perspective Projection 1: Scale and Shear View volume

A shear must be applied to bring the centre line of the view volume parallel to the z axis, the x and y coordinates scaled so that the sides of the frustum lie on the lines  $x = \pm z$  and  $y = \pm z$ .



## Perspective Projection 1: Scale and Shear View volume (cont)

Asymmetry is corrected by a shear and the sides are fixed by scaling. For the x axis the combined operation is:

$$x \rightarrow \frac{2Nx + (x_{\max} + x_{\min})z}{\text{width}}$$

$$\text{width} = (x_{\max} - x_{\min}) \text{ and asymmetry} = (x_{\max} + x_{\min})$$

Remember that if the frustrum is symmetric  $x_{\max} = -x_{\min}$  so the asymmetry is zero and the width is  $2x_{\max}$ .

$$P_1 = \begin{bmatrix} \frac{2N}{(x_{\max} - x_{\min})} & 0 & \frac{(x_{\max} + x_{\min})}{(x_{\max} - x_{\min})} & 0 \\ 0 & \frac{2N}{(y_{\max} - y_{\min})} & \frac{(y_{\max} + y_{\min})}{(y_{\max} - y_{\min})} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

N = distance to near clip plane

## Perspective Projection 2: Simple Perspective

This is carried out in 2 stages: first scale the x and y coordinates, since the scale must be inversely proportional to the z coordinate this is a shear along the z axis.

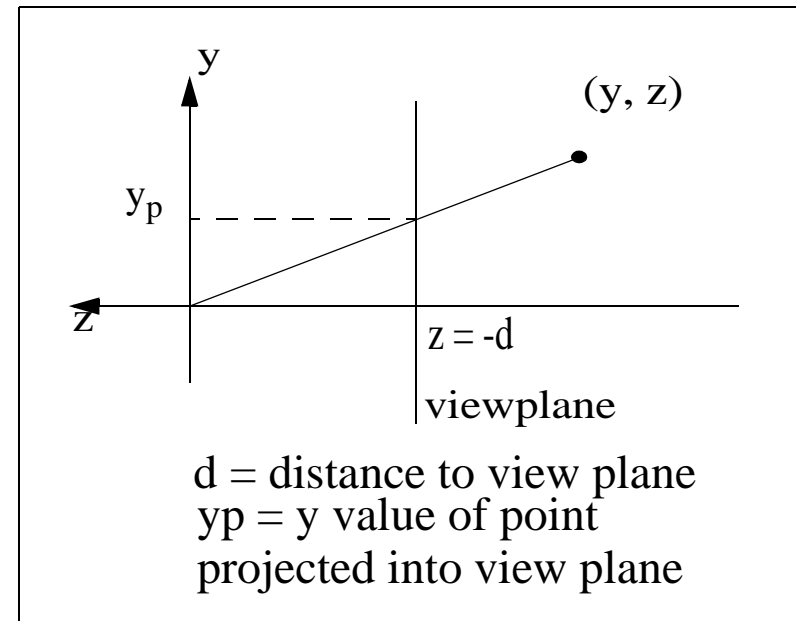
Similar triangles

$$\frac{y}{z} = \frac{y_p}{-d} \quad \text{so} \quad y_p = \frac{-dy}{z} = \frac{y}{z/(-d)}$$

So transformation

$$P_s = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/(-d) & 0 \end{bmatrix}$$

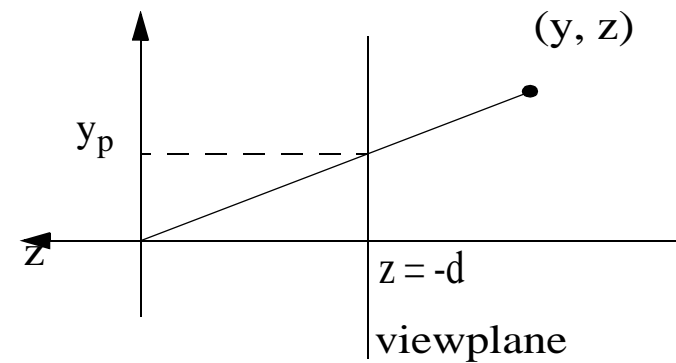
When vector renormalised



## Perspective Projection 2: Simple Perspective (cont)

Remember the vector is renormalised after the transformation is applied. So

$$P_s \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/(-d) \end{bmatrix} = \begin{bmatrix} \frac{x}{z/(-d)} \\ \frac{y}{z/(-d)} \\ -d \\ 1 \end{bmatrix}$$



Note that depth data is lost as z coordinates are compressed onto plane  $z = -d$ .

## Perspective Projection: OpenGL Perspective

Easy part put projection plane at -1 so transformation becomes

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

We need to preserve depth information so that we can carry out hidden surface removal also we want the sides of the frustum to be at  $\pm 1$ . Given the sides are already at  $\pm z$  we need a transformation such that:  $x = \pm z \quad y = \pm z \rightarrow x = \pm 1 \quad y = \pm 1$ .

$$P_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{-(F+N)}{F-N} & \frac{2FN}{F-N} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$N =$  distance to near clip plane,  $N > 0$

$F =$  distance to far clip plane,  $F > 0$

## Perspective Projection: OpenGL Perspective (cont)

The full OpenGL perspective matrix is given by the composition of  $P_1$  and  $P_2$ .

$$P = P_2 P_1 = \begin{bmatrix} \frac{2N}{(x_{\max} - x_{\min})} & 0 & \frac{(x_{\max} + x_{\min})}{(x_{\max} - x_{\min})} & 0 \\ 0 & \frac{2N}{(y_{\max} - y_{\min})} & \frac{(y_{\max} + y_{\min})}{(y_{\max} - y_{\min})} & 0 \\ 0 & 0 & \frac{-(F + N)}{F - N} & \frac{2FN}{F - N} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

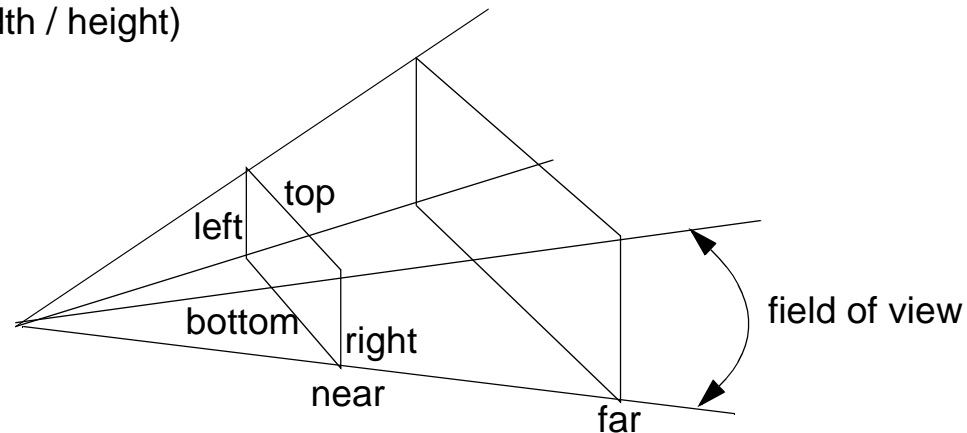
## Perspective Projection: OpenGL Commands

The OpenGL perspective matrix can be generated by using the GL command:

```
void glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);  
left, right = max and min of x  
top, bottom = max and min of Y
```

There is a simplified version, for symmetric viewports, in the utility library GLU:

```
void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble near, GLdouble far)  
fovy = vertical field of view  
aspect = aspect ratio (width / height)
```



## Map Into Viewport

The final stage is to map the window on the view plane into the view port on the screen. To do this a transformation is applied that maps the bottom left corner of the window into the pixel location of the bottom left of the viewport. The height and width of the window are also transformed to match the viewports pixel height and width.

GL provides the command:

```
void glViewport(GLint x, GLint y, GLsizei width, GLsizei height)
```

Note: if the aspect ratio of the viewport is different from the window on the view plane then distortions will occur.

## Perspective OpenGL Example

First set up projection matrix:

```
glViewport(0, 0, w, h);
glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(120, w/ h, nearClip, farClip); /* reuse w and h to maintain aspect ratio */
glMatrixMode(GL_MODELVIEW);
```

Then transform to view reference coordinates and draw your world:

```
glLoadIdentity();
    gluLookAt( viewer[0],viewer[1],viewer[2], /* look from point or, eye point*/
              0.0, 0.0, 0.0,             /* look-at point */
              0.0, 1.0, 0.0 );          /* head-up vector */
glColor3fv(col1);                       /* set a colour */
glPushMatrix();                          /* push the model view matrix to preserve it, copy stays on stack top*/
glRotatef(rot0, 0.0, 1.0, 0.0);          /* premultiply the copy of the model view matrix by the a rotation */
glTranslatef( 5.5, 3.7 0.0);             /* premultiply the stack top by a translateion */
glCallList( thing1 );                   /* call a display list for an object, the current stack top transforms it */
glPopMatrix();                           /* pop the original model view matrix back */

/* repeat for next thing */
```