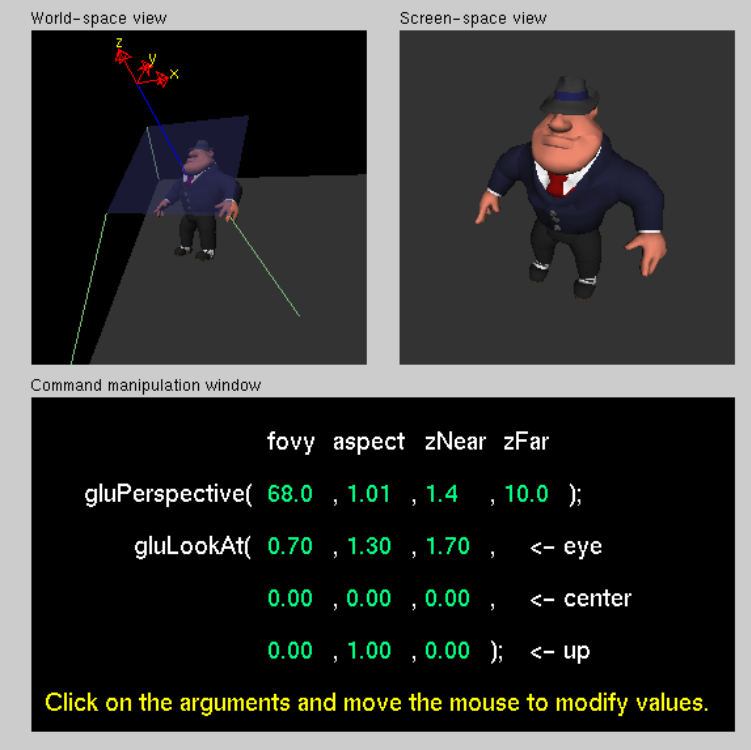
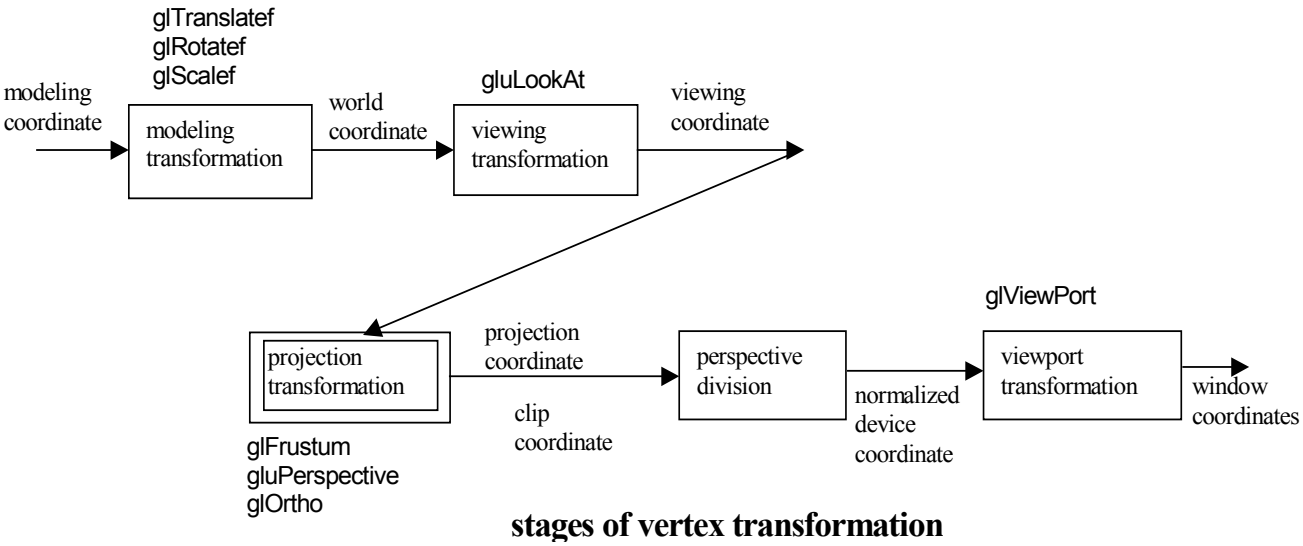
Note 3: Projection Transformation

Reading: Textbook 5

1. Introduction

In the last lecture, our concentration was on creating model and putting the model in the right place for viewing. Next we need to consider how we want the projection to be.





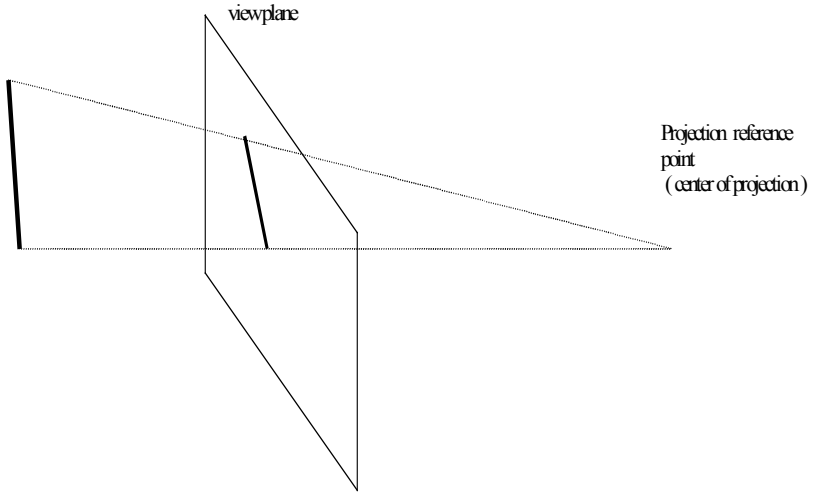
stages of vertex transformation

Demo program: projection from Nate Robins (Utah)

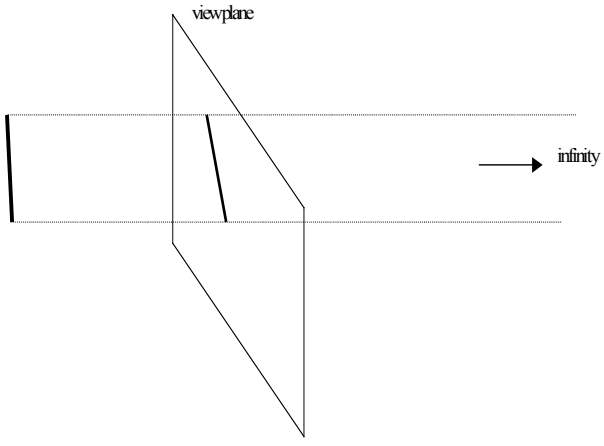
2. Simple Projections

There following are 2 main classes of planar geometric projections.

(i) perspective projection: object position are transformed to the view plane along the lines that converge to a point called the *center of projection*

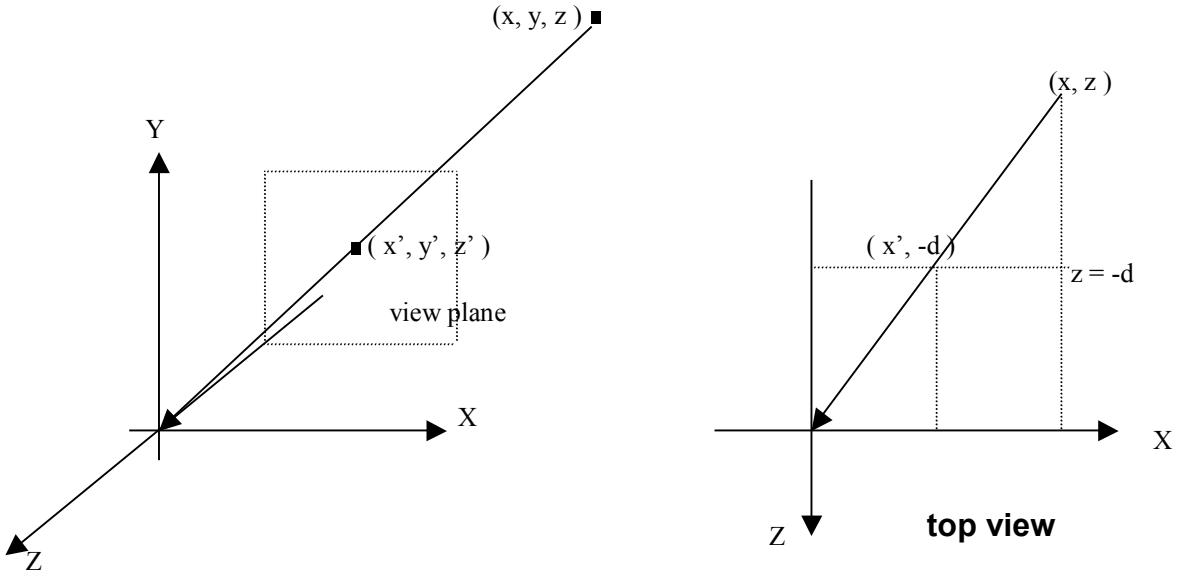


(ii) parallel projection: determined by Direction of projection (projectos are parallel—do not converge to an "eye" of COP…..some consider this as a special case of perspective projection with cop at infinity (along some direction)



- **orthogonal projection: the projections are perpendicular to the view plane**
- **oblique projection: the projection are not perpendicular to the view plan**

Perspective Case:



For sure, we have $z' = -d$

From similar triangle (top view), we get $\frac{x}{z} = \frac{x'}{-d} \Rightarrow x' = -\frac{x}{z/d}$

Similarly, we can get $y' = -\frac{y}{z/d}$

Thus, we have transformation matrix $\mathbf{M} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{pmatrix}$

Orthogonal Case:

x' = x
y' = y
z' = 0 (for projecting onto view plane at z=0)

We have transformation matrix $\mathbf{M} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

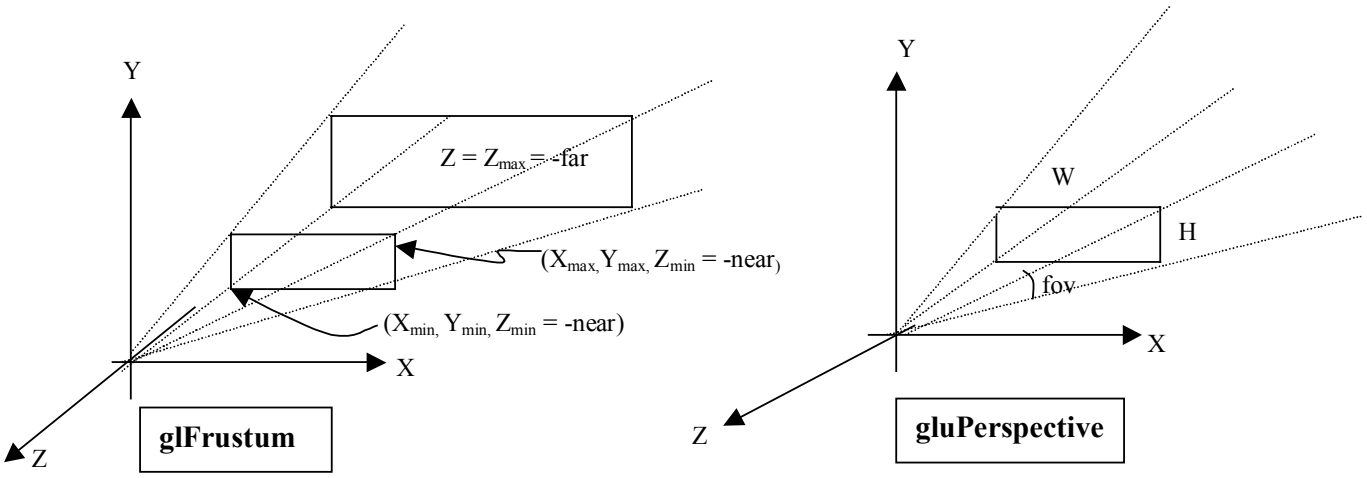
3 OpenGL Projection Transformation

Projection transformation defines a viewing volume that

- **determines how an object is projected onto the screen.**
- **defines which objects or portions of an object are clipped out of the final image.**

OpenGl provide the following to specify perspective view:

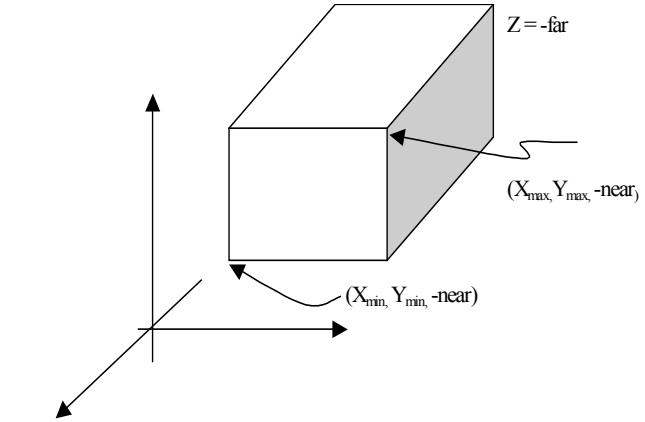
glFrustum (xmin, xmax, ymin, ymax, near, far)
gluPerspective (fovy, aspect, near, far)




```
void APIENTRY gluPerspective( GLdouble fovy, GLdouble aspect, GLdouble near, GLdouble far )
{  GLdouble xmin, xmax, ymin, ymax;
   ymax = near * tan( fovy * M_PI / 360.0 );          ymin = -ymax;
   xmin = ymin * aspect;                                xmax = ymax * aspect;
  glFrustum( xmin, xmax, ymin, ymax, near, far );
}
```

OpenGl provides the following to specify parallel view:

glOrtho (xmin, xmax, ymin, ymax, near, far)
gluOrtho2D (xmin, xmax, ymin, yman)

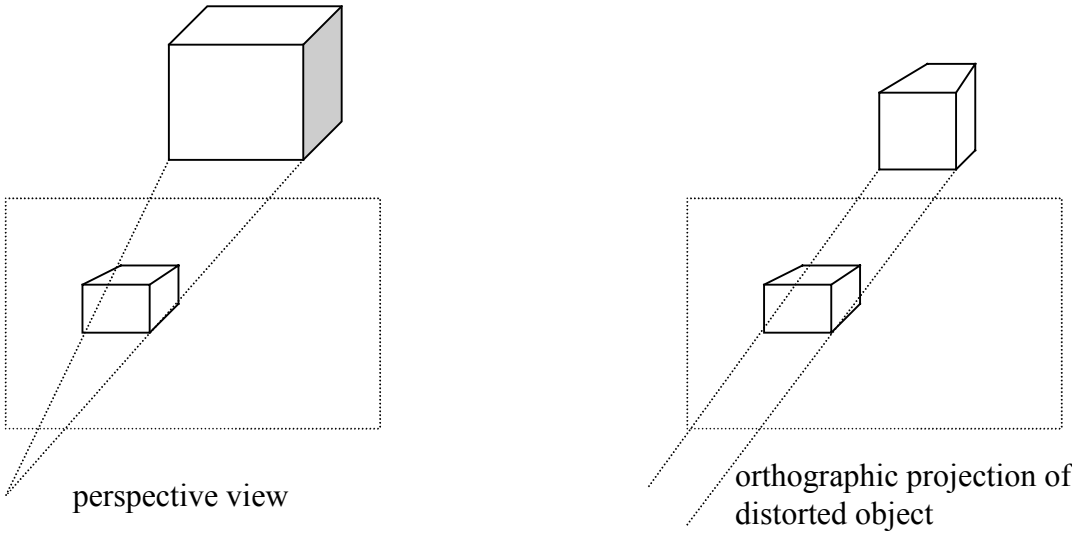



```
void APIENTRY gluOrtho2D( GLdouble left,GLdouble right, GLdouble bottom, GLdouble top )
{       glOrtho( left, right, bottom, top, -1.0, 1.0 );     }
```

4 Projection Matrices in OpenGL

Reference: Textbook 5.7.1.

- Approach is based on a technique - ***projection-normalization,*** that converts all projections into *orthogonal projections* by first distorting the objects such that the orthogonal projections of the distorted objects is the same as the desired projection of the original objects. (see Fig 5.29)



Distort (normalize) ➔ orthographic projection

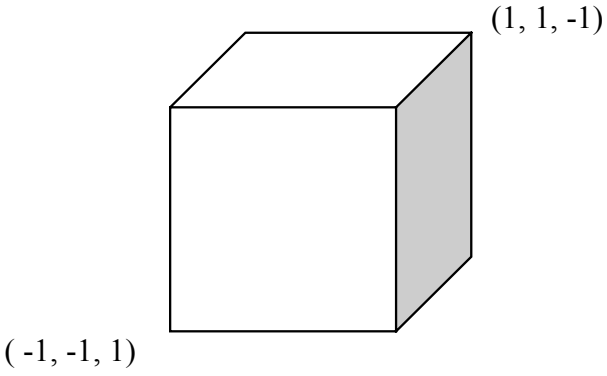
e.g.1. Orthogonal-Projection Matrix in OpenGL

(Reference: Textbook: 5.7.2)

For orthographic projections, the simplest clipping volume to deal with is a cube whose center is at the origin, whose sides are:

$x = \pm 1, \quad y = \pm 1, \quad z = \pm 1.$

This volume is called the ***canonical view volume*** and is defined by:



glOrtho(-1.,1.,-1.,1.,-1.,1.);

Now suppose we set the *glOrtho* parameters as:

$glOrtho(x_{min}, x_{max}, y_{min}, y_{max}, near, far);$

Projection matrix OpenGL sets up will convert the *vertices* that specify the objects to *vertices* within the *canonical view volume*, by scaling and translating them, then changing to a ***left-handed coord sys.***

Thus, the projection matrix that maps the specified volume to the canonical view volume may be represented by **P** where

$$\mathbf{P} = \mathbf{S}_{\mathbf{reflection}} \bullet \mathbf{S} \bullet \mathbf{T}$$

with

$$\mathbf{S}_{\mathbf{reflection}} = \mathbf{S}(1, 1, -1) ,$$

$$\mathbf{S} = \mathbf{S}(2/(x_{max} - x_{min}),\ \ 2/(y_{max} - y_{min}),\ \ 2/(far - near))$$

$$\mathbf{T} = \mathbf{T} (-(x_{max} + x_{min})/2,\ \ -(y_{max} + y_{min})/2,\ \ (far + near)/2)$$

T translates center of cube to origin, **S** scales the edge to length 2 and $\mathbf{S}_{\mathbf{reflection}}$ changes the right-hand coordinates to the left-handed coordinates.

Thus $\mathbf{P} = \mathbf{S}_{\mathbf{reflection}} \bullet \mathbf{S} \bullet \mathbf{T}$

$$P = \begin{bmatrix} \frac{2}{x\max - x\min} & 0 & 0 & \frac{-(x\max + x\min)}{x\max - x\min} \\ 0 & \frac{2}{y\max - y\min} & 0 & \frac{-(y\max + y\min)}{y\max - y\min} \\ 0 & 0 & \frac{-2}{far - near} & \frac{-(far + near)}{far - near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The matrix **P** is implemented in OpenGL by the function

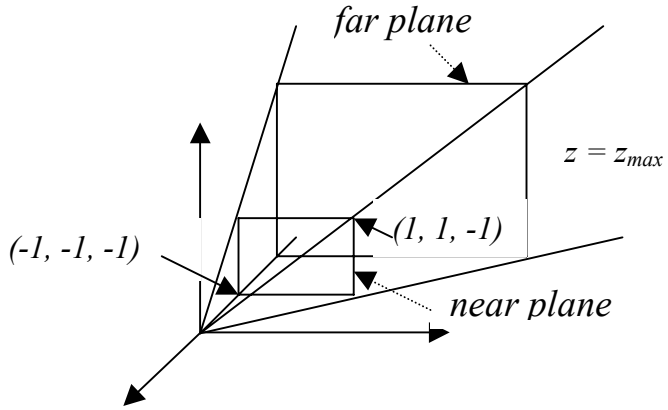
glOrtho (xmin, xmax, ymin, ymax, near, far)

e.g 2. Perspective-Projection Matrices (warm-up)

(Reference: Textbook: 5.8.1)

Consider first the matrix **N** and the simple perspective projection:
(fix the angle of view at 90 degrees by making the sides of the viewing volume intersect the projection plane at a 45-degree angle)

$$\mathbf{N} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -2 & -2 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$



$$\mathbf{N}\,[\,x \;\; y \;\; z \;\; 1\,]^{\mathbf{T}} = [\,x \;\; y \;\; (-2\text{-}2z) \;\; -z\,]^{\mathbf{T}}$$

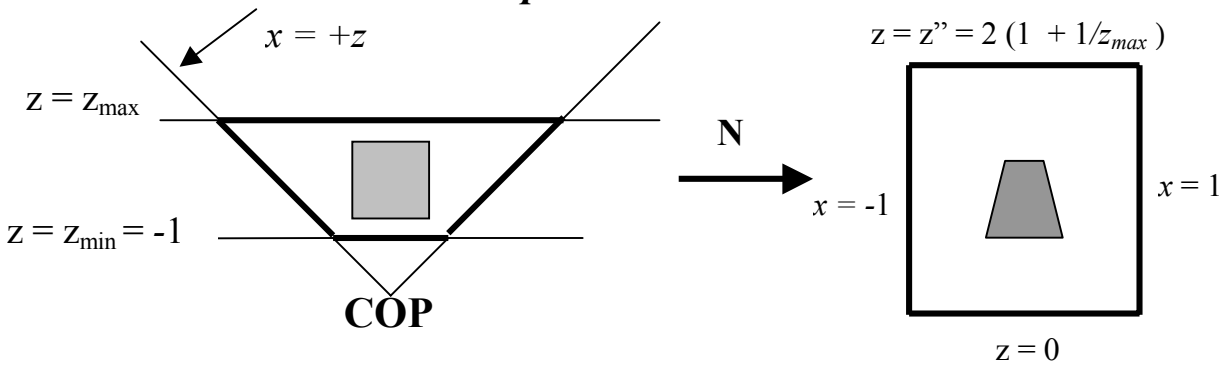
or

$$x'' = -x/z, \quad y'' = -y/z, \quad z'' = 2(1 + 1/z)$$

The matrix **N** has transformed the viewing volume into a new volume. Their original and new side planes given as follows:

original		new
$x = \pm z$	*into*	$x'' = -(\pm 1)$,
$y = \pm z$	*into*	$y'' = -(\pm 1)$,
$z = -1$	*into*	$z'' = 0$,
$z = z_{max}$	*into*	$z'' = 2\,(\,1 + 1/z_{max}\,)$

Top View :



Next consider an orthographic projection matrix $\mathbf{M}_{orth}$ with projection plane $z = 0$ and apply it after $\mathbf{N}$:

$$\mathbf{M}_{orth} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{M}_{orth}\,\mathbf{N} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$\mathbf{M}_{orth}\,\mathbf{N}$ becomes a simple projection matrix (with d=1)!

Checking:

$$\mathbf{p}_p = \mathbf{M}_{orth}\,\mathbf{N}\,\mathbf{p} = [\, x \;\; y \;\; 0 \;\; -z \,]^T$$

$$\Rightarrow \quad x_p = -x/z, \quad y_p = -y/z, \quad z_p = 0$$

- Thus $\mathbf{N}$ has transformed the view frustum into a right parallelepiped, and an orthographic projection in the transformed volume yields the same image as does the perspective projection.

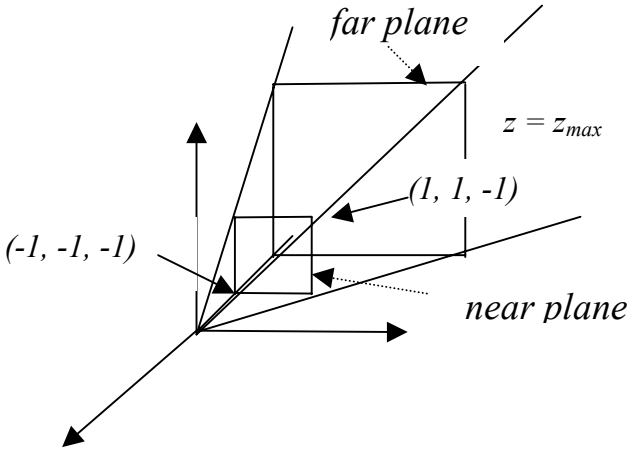
- $\mathbf{N}$ is called the ***perspective-normalization matrix*** **(converts a perspective projection to an orthogonal projection)**

$$\mathbf{N} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -2 & -2 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

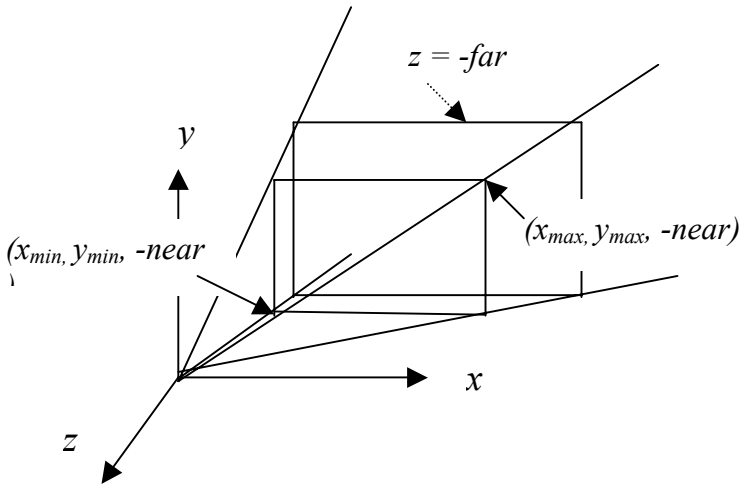
In the next example, we use a modified $\mathbf{N}$ for our purposes. The modification is necessary to move our *z"* to between –1 and +1.

e.g 3. OpenGL Perspective Transformation

Note: there are quite a bit of typos in the textbook on this topic



symmetric viewing frustum



OpenGL Perspective frustum

(1) Convert OpenGL frustum to the symmetric frustum by a shear transformation **H** (cot θ, cot ϕ). The skew (shear) angle is determined by our desire to shear the point ($(x_{min} + x_{max})/2$, $(y_{min} + y_{max})/2$, *-near*) to (*0, 0, -near*)

$$\mathbf{H}(\cot\theta, \cot\phi) = \begin{bmatrix} 1 & 0 & \frac{x_{min} + x_{max}}{2\,near} & 0 \\ 0 & 1 & \frac{y_{min} + y_{max}}{2\,near} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The resultant symmetric frustum is described by the planes:

$$x = \pm\left(\frac{x_{max} - x_{min}}{2\,near}\right) z, \quad y = \pm\left(\frac{y_{max} - y_{min}}{2\,near}\right) z, \quad z = -far, \quad z = -near.$$

(2) Want to scale the sides to :

$x = \pm z$, $y = \pm z$, $z = -far$, $z = -near$.

Required scaling matrix **S** is therefore**:**

S = **S** [$2\,near/(x_{max} - x_{min})$, $2\,near/(y_{max} - y_{min})$, 1]

(3) To transform the far plane to the plane $z = +1$ and the near plane to $z = -1$ using the projection normalization, we must make a slight change to the ***projection-normalization matrix*** **N** by adjusting the parameters α and β :

$$\mathbf{N} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

The effect of **N** on *x, y,* and *w* components of a point **p** remains the same as before, but the new *z* is now given by: $z' = -(\alpha + \frac{\beta}{z})$.

We require near plane: $z = -near$ be moved to $z' = -1$,

far plane: $z = -far$ be moved to $z' = +1$.

Thus: $\alpha = \dfrac{-(far + near)}{far - near}$ $\beta = \dfrac{-2\,far * near}{far - near}$

With these new values for the ***projection-normalization matrix*** **N,** the required projection matrix is therefore

$$\mathbf{P} = \mathbf{NSH} = \begin{bmatrix} \dfrac{2near}{(x_{max} - x_{min})} & 0 & \dfrac{x_{max} + x_{min}}{x_{max} - x_{min}} & 0 \\ 0 & \dfrac{2near}{y_{max} - y_{min}} & \dfrac{y_{max} + y_{min}}{y_{max} - y_{min}} & 0 \\ 0 & 0 & \dfrac{-(far + near)}{far - near} & \dfrac{-2\,far * near}{far - near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$