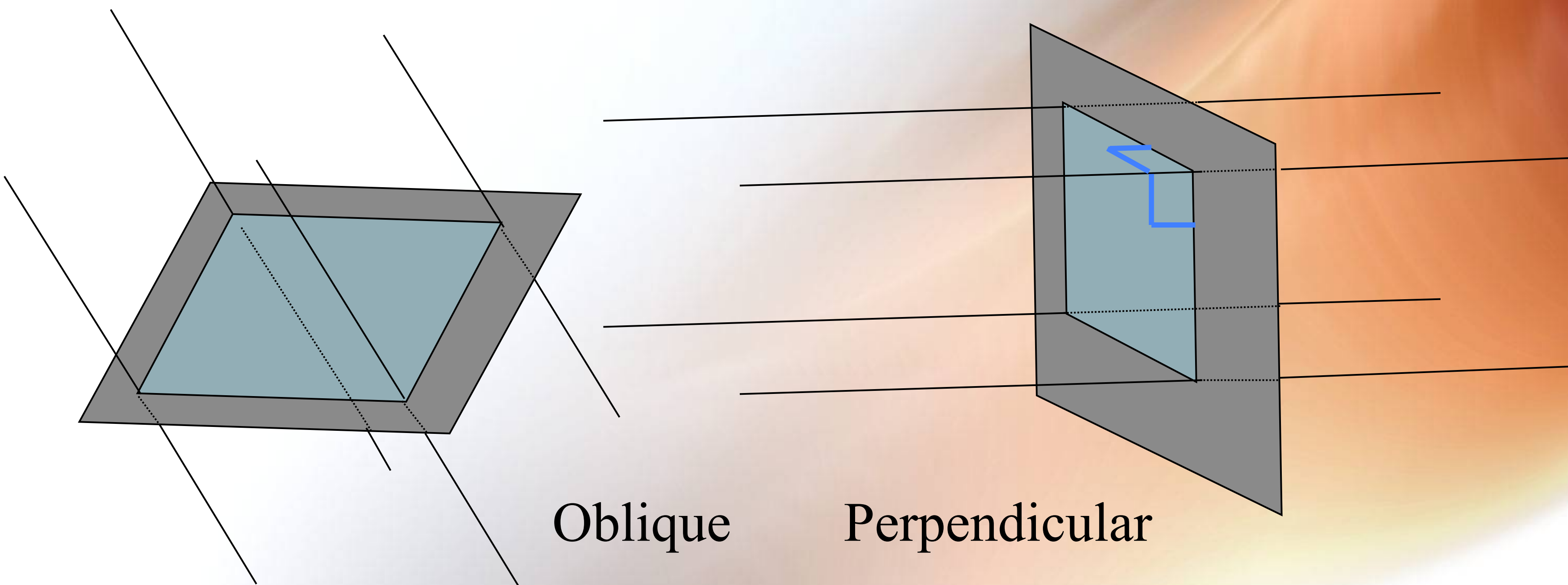
Computer Graphics

Viewing Transformations and Projection

Based on slides by Dianna Xu, Bryn Mawr College

Parallel Projection Clipping View Volume

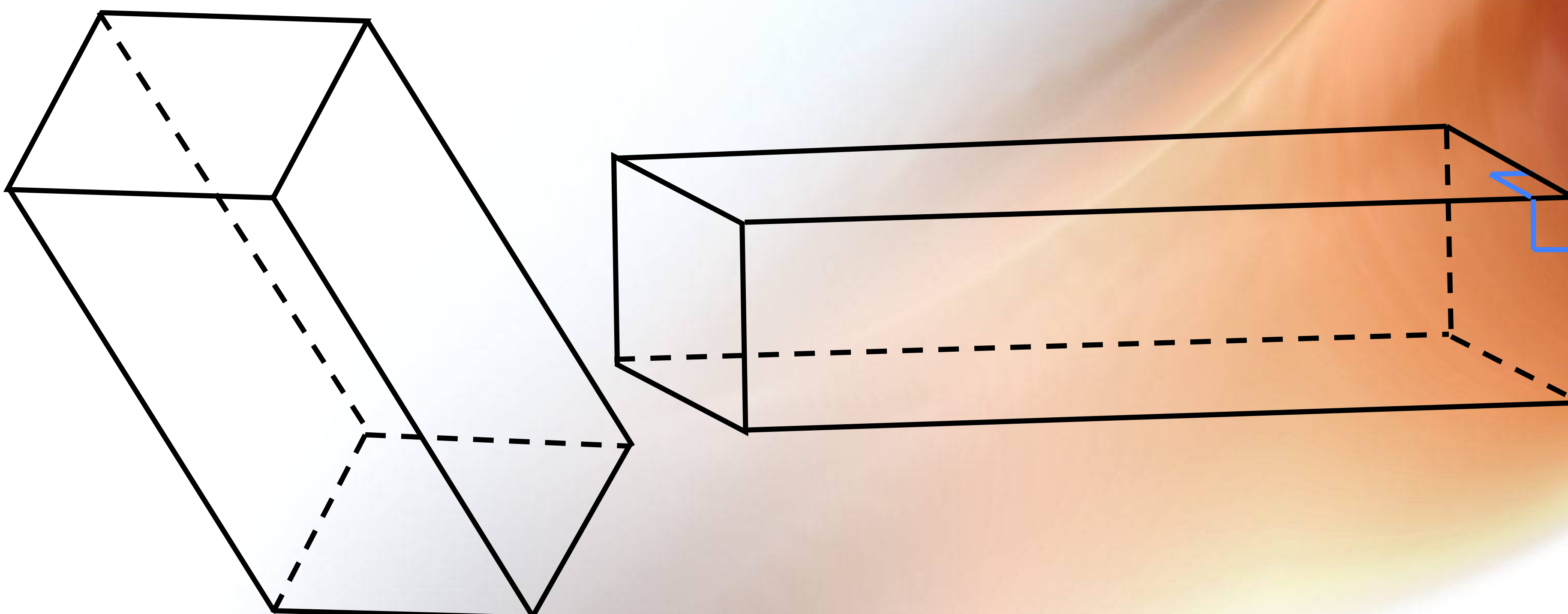
- View Volume determined by the direction of projection and the window



Oblique   Perpendicular

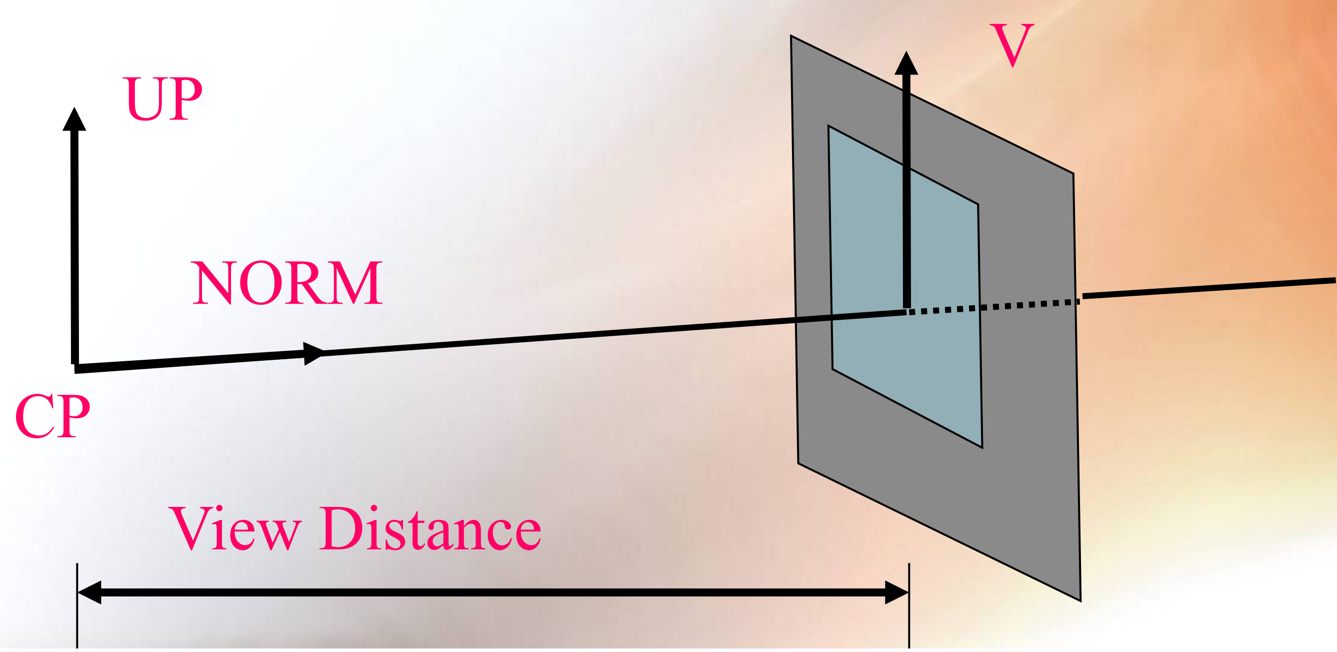
Parallel Projection View Volume

- **View Volume is now a** parallelopiped**.**

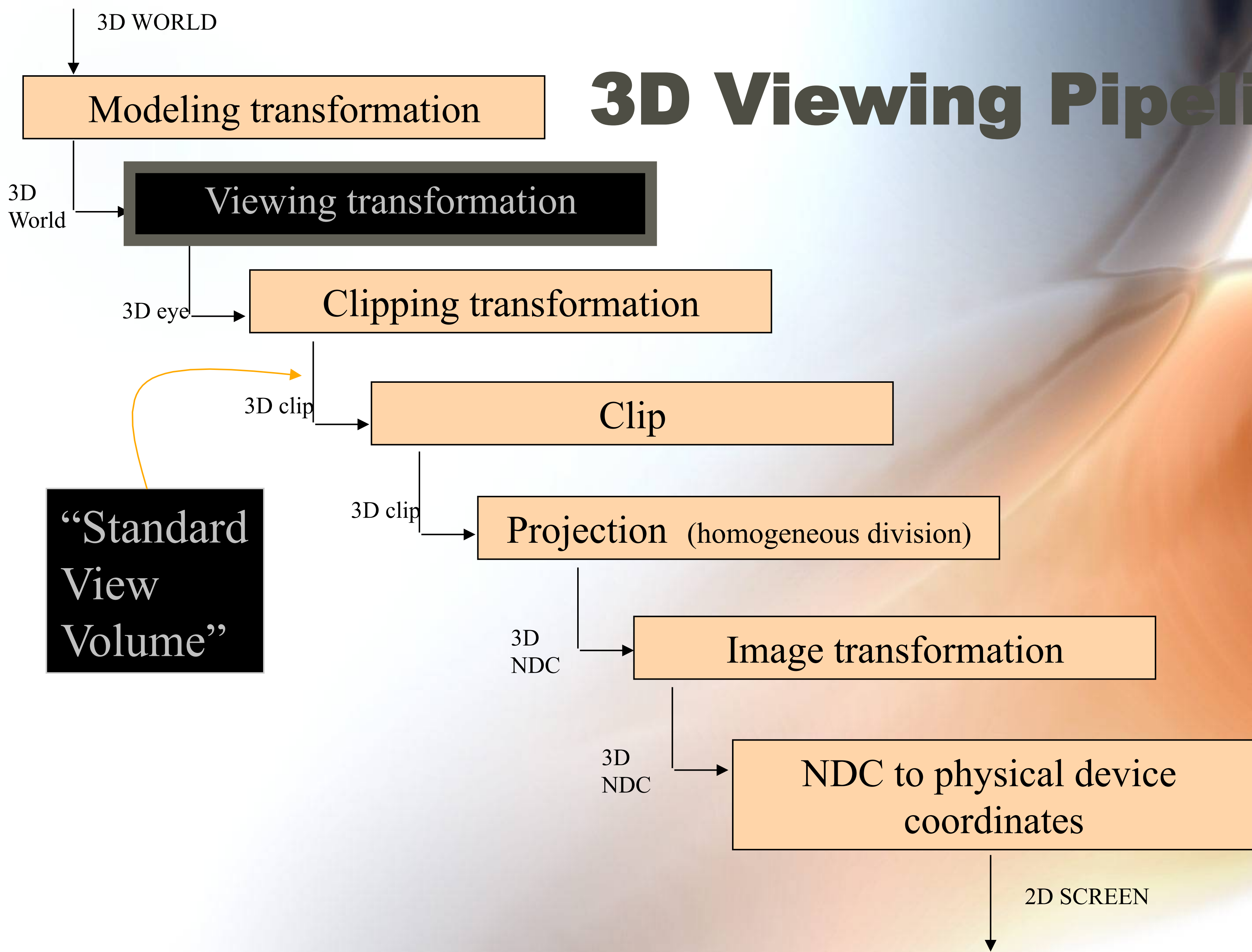


The Synthetic Camera

- Translated via CP changes.
- Rotated via UP changes.
- Redirected via View Plane Normal changes (e.g. panning).
- Zoom via changes in View Distance



3D Viewing Pipeline

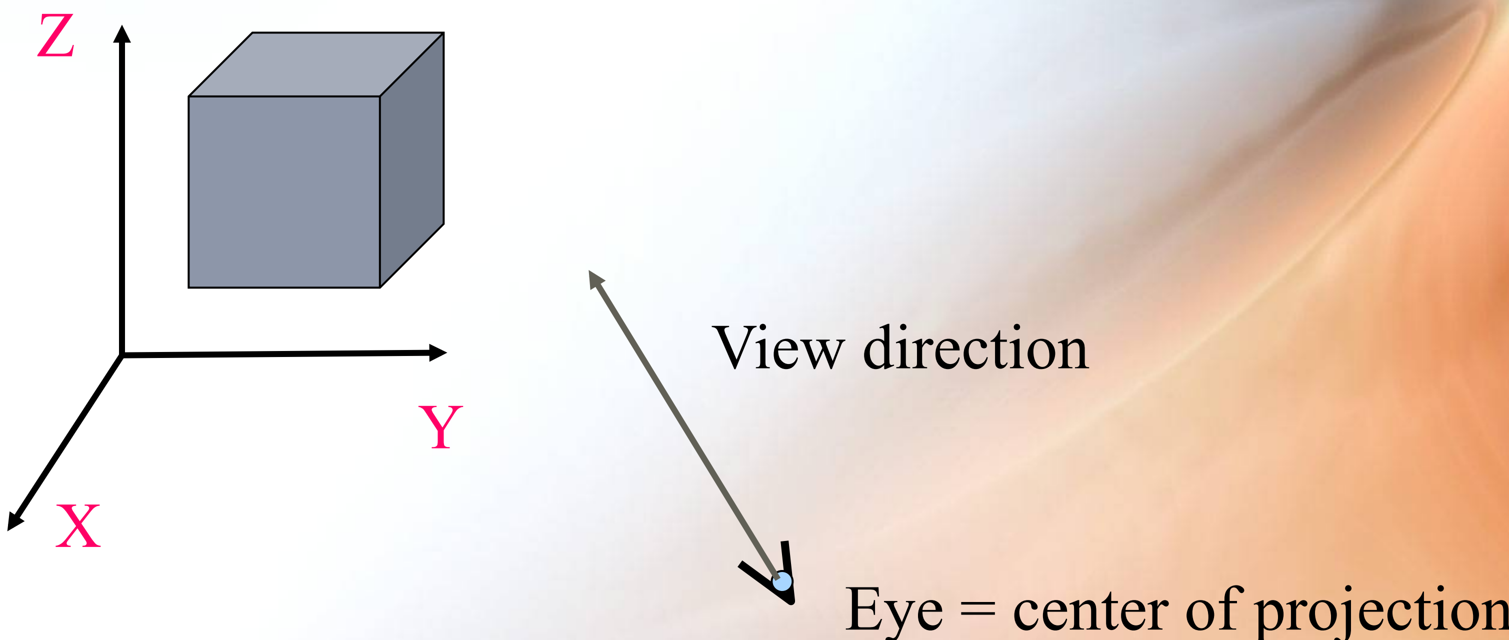


Transform World Coordinates to Eye Coordinates

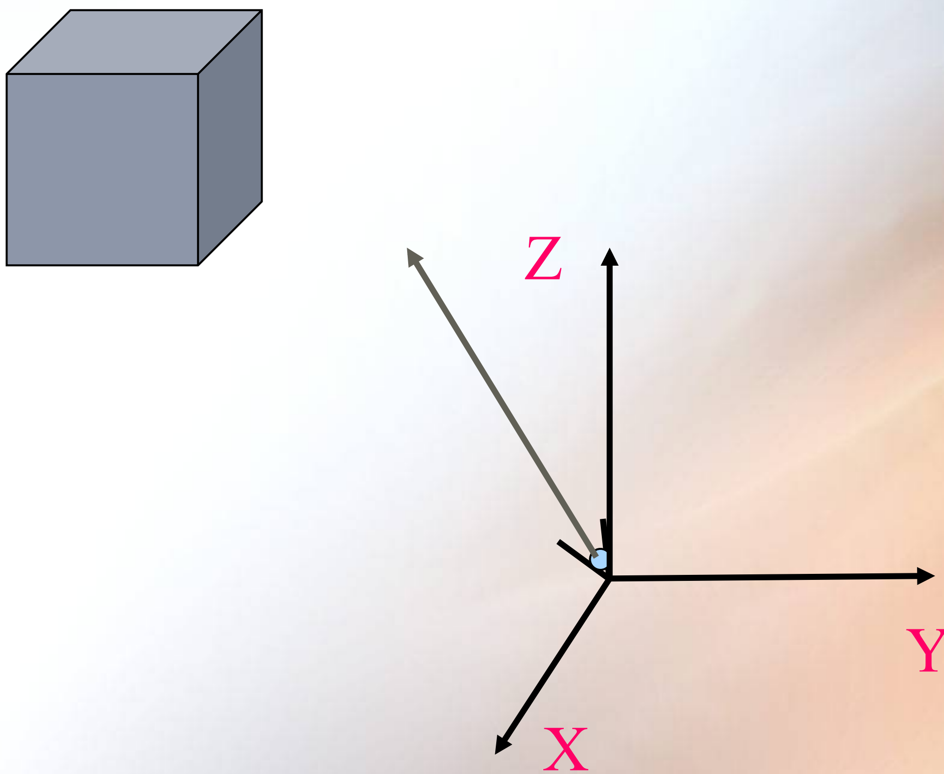
Approximate steps:

- Put eye (center of projection) at (0, 0, 0).
- Make X point to right.
- Make Y point up.
- Make Z point forward (away from eye in depth).
- (This is now a *left-handed* coordinate system!)

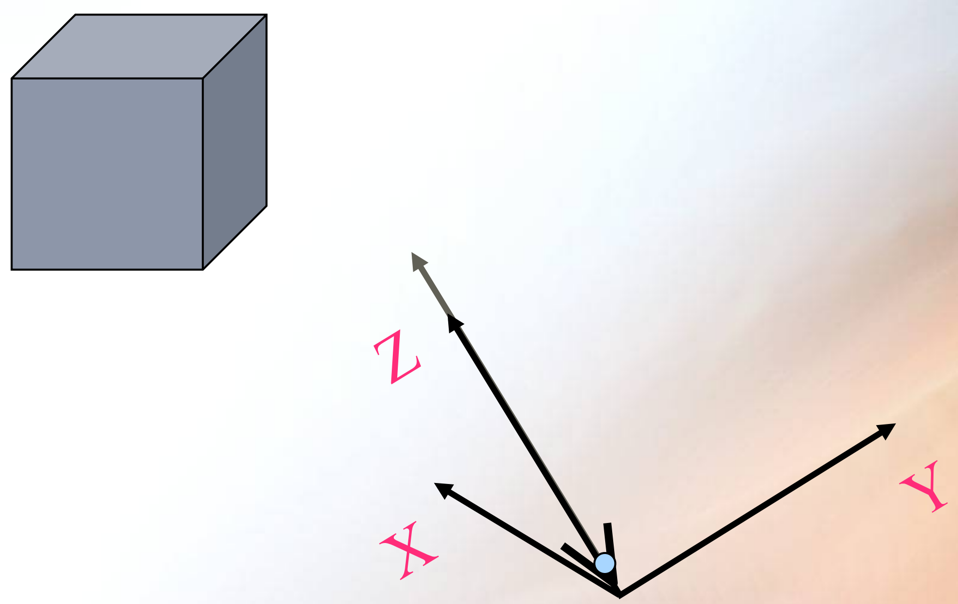
World to Eye Transformation START



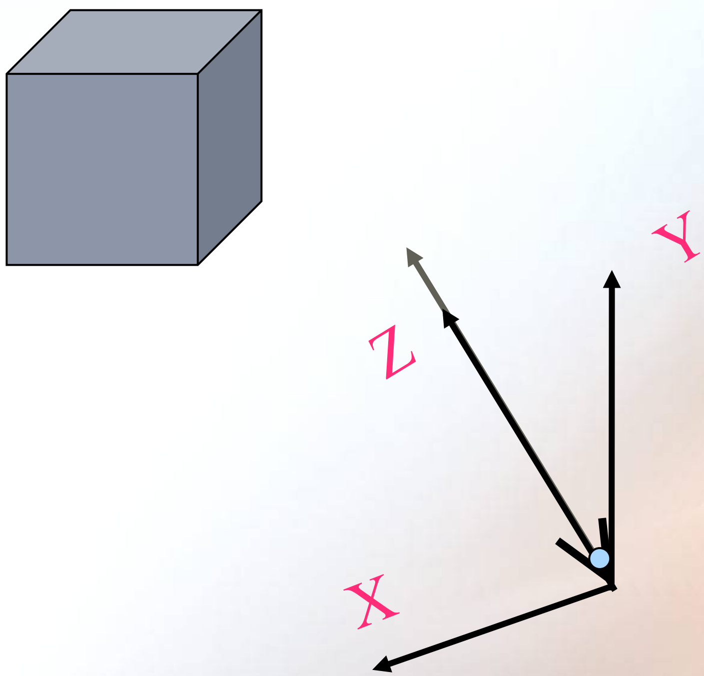
World to Eye Transformation
Translate eye to (0, 0, 0)



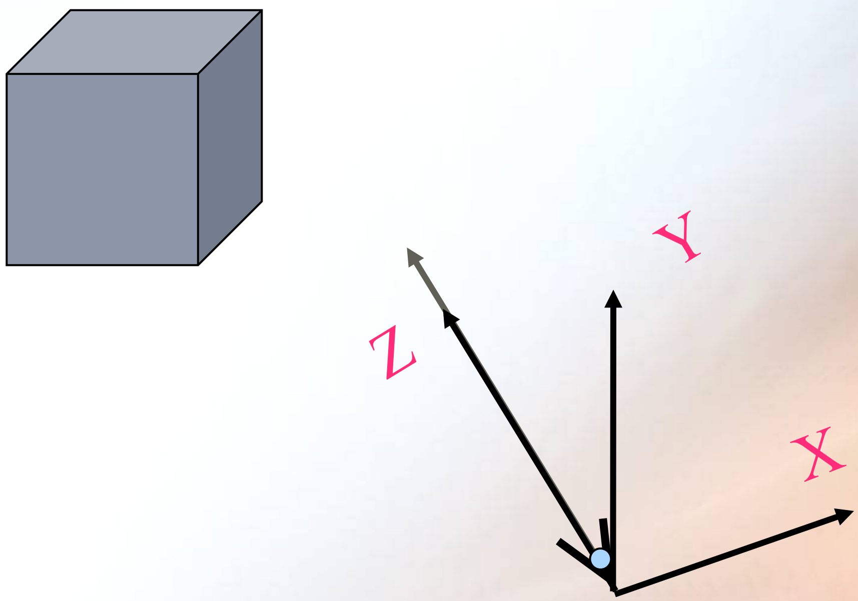
World to Eye Transformation
Align view direction with +Z



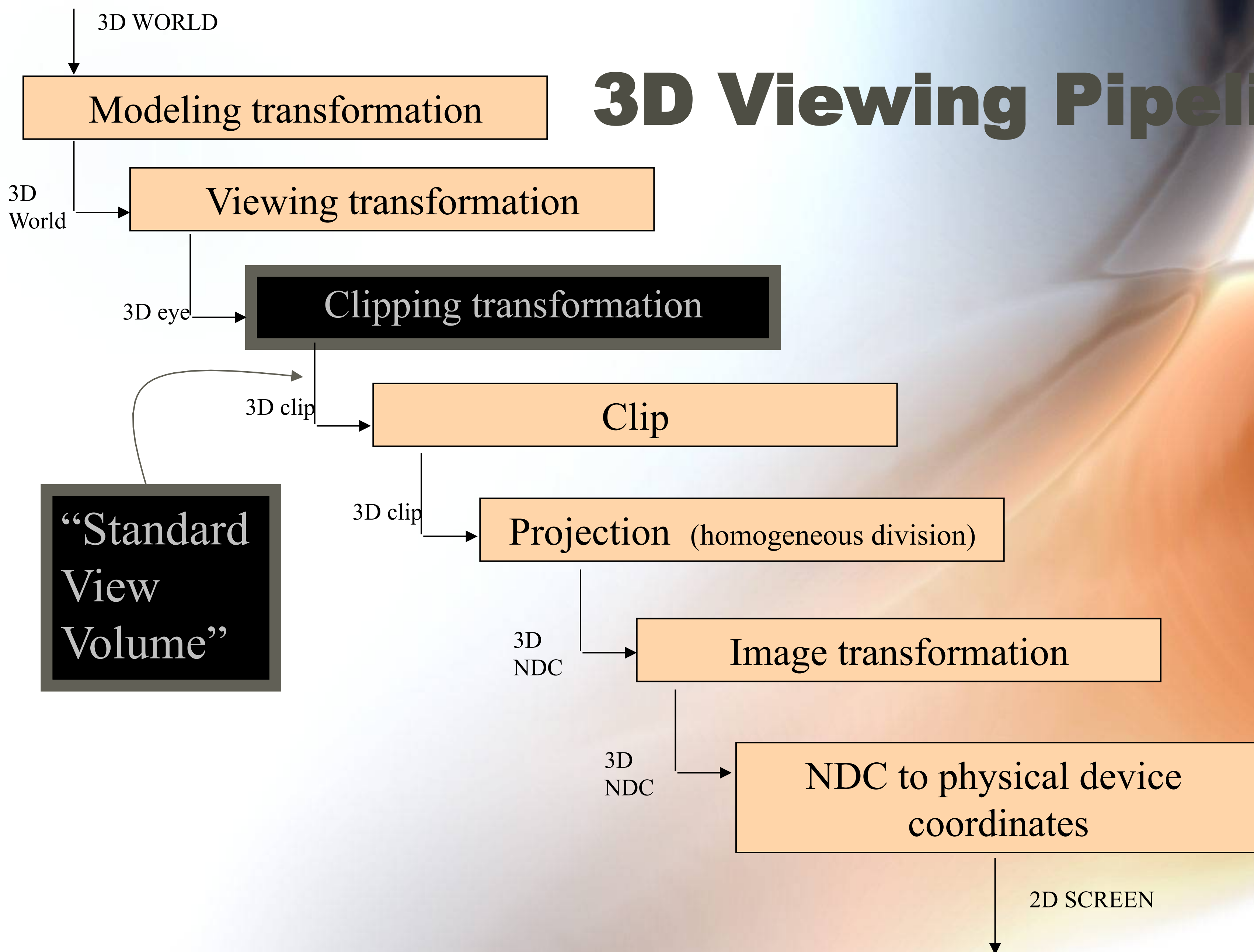
World to Eye Transformation
Align VUP direction with +Y



World to Eye Transformation Scale to LH coordinate system

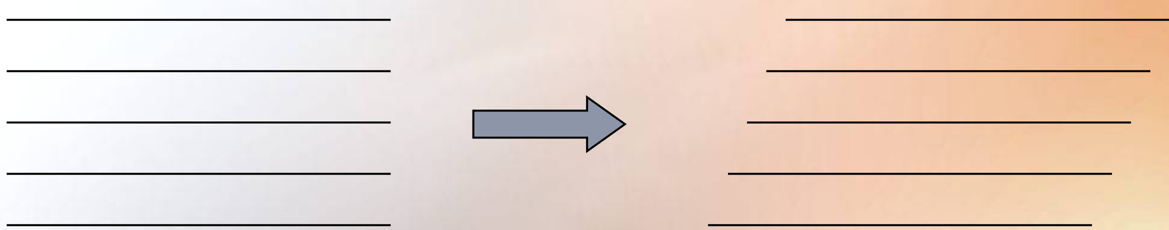


3D Viewing Pipeline

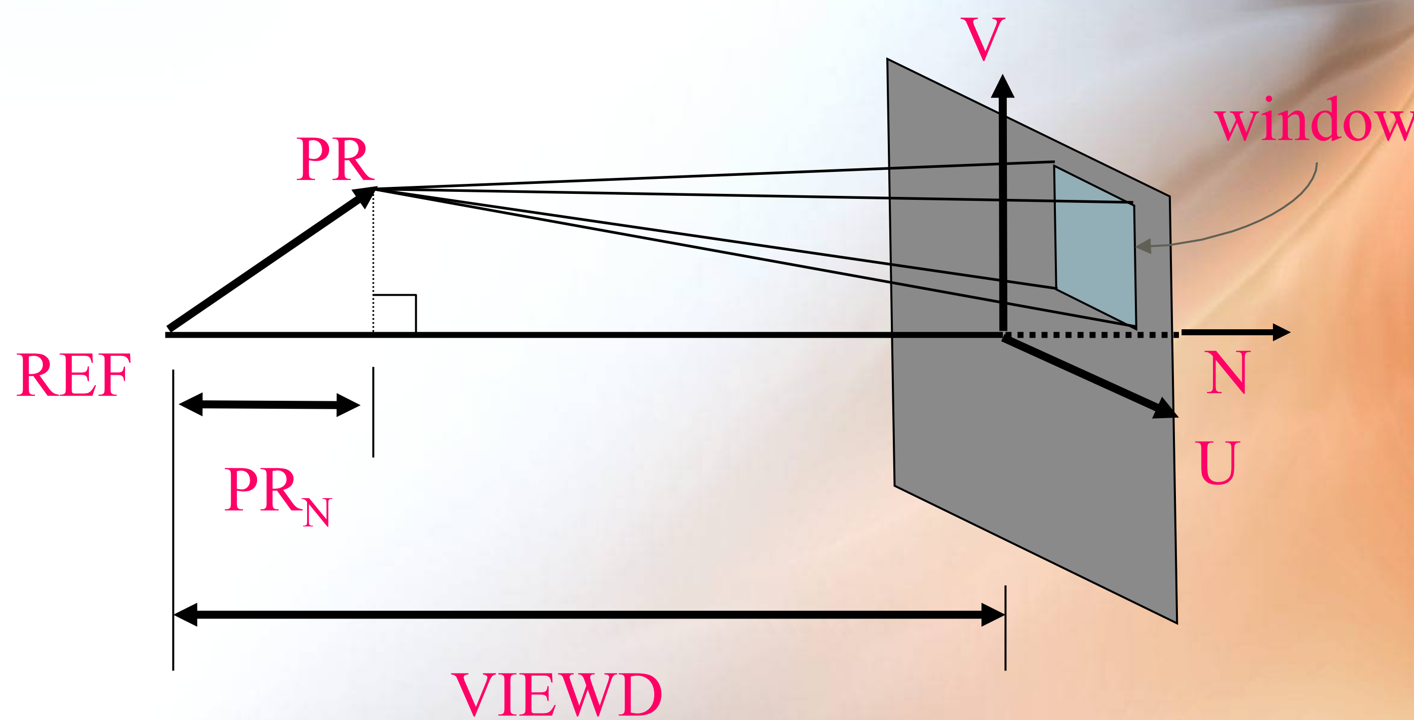


On to the Clipping Transformation

- It remains to do the transformations that put these coordinates into the clipping coordinate system
- We have to shear it to get it upright

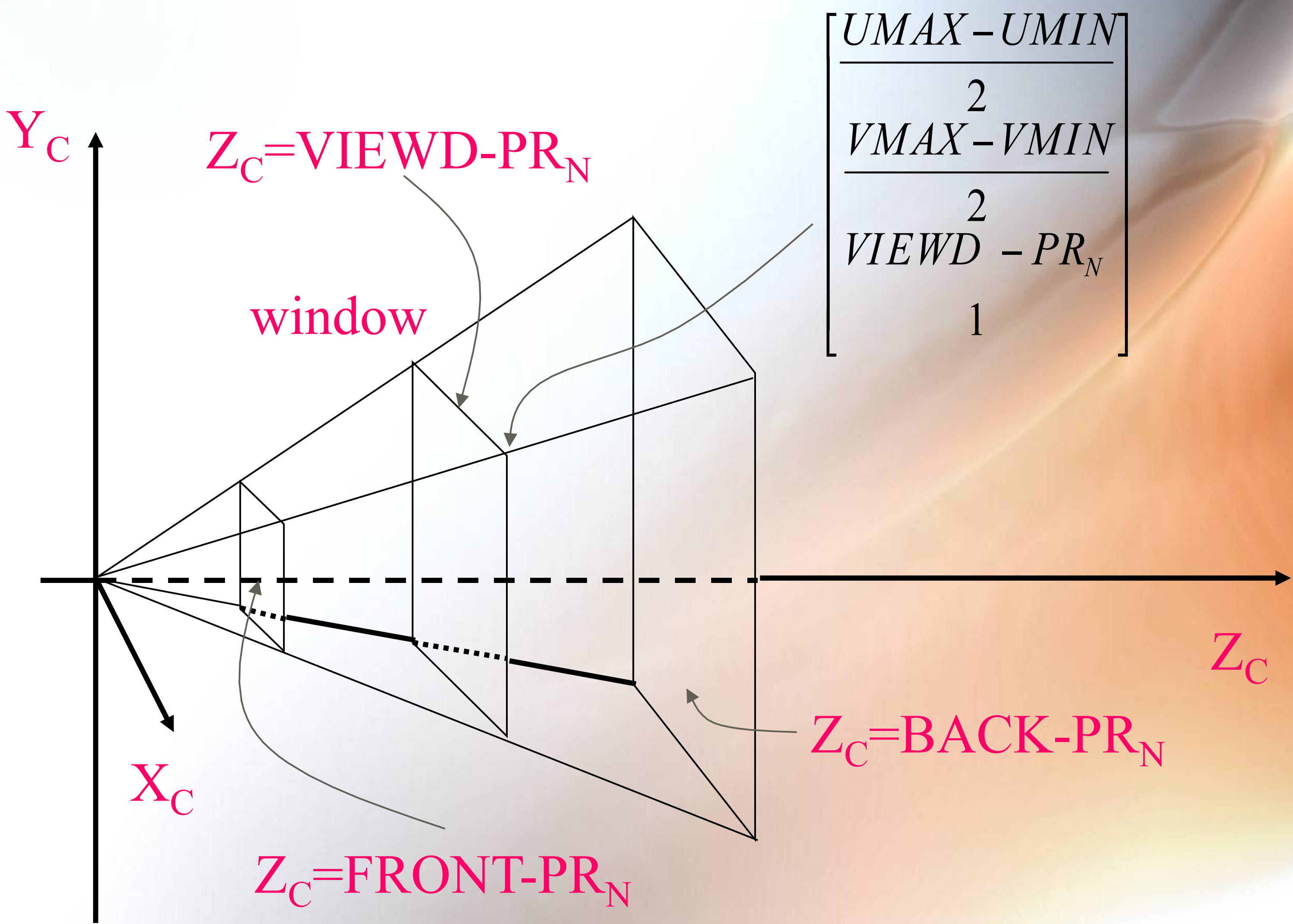


Shear Layout

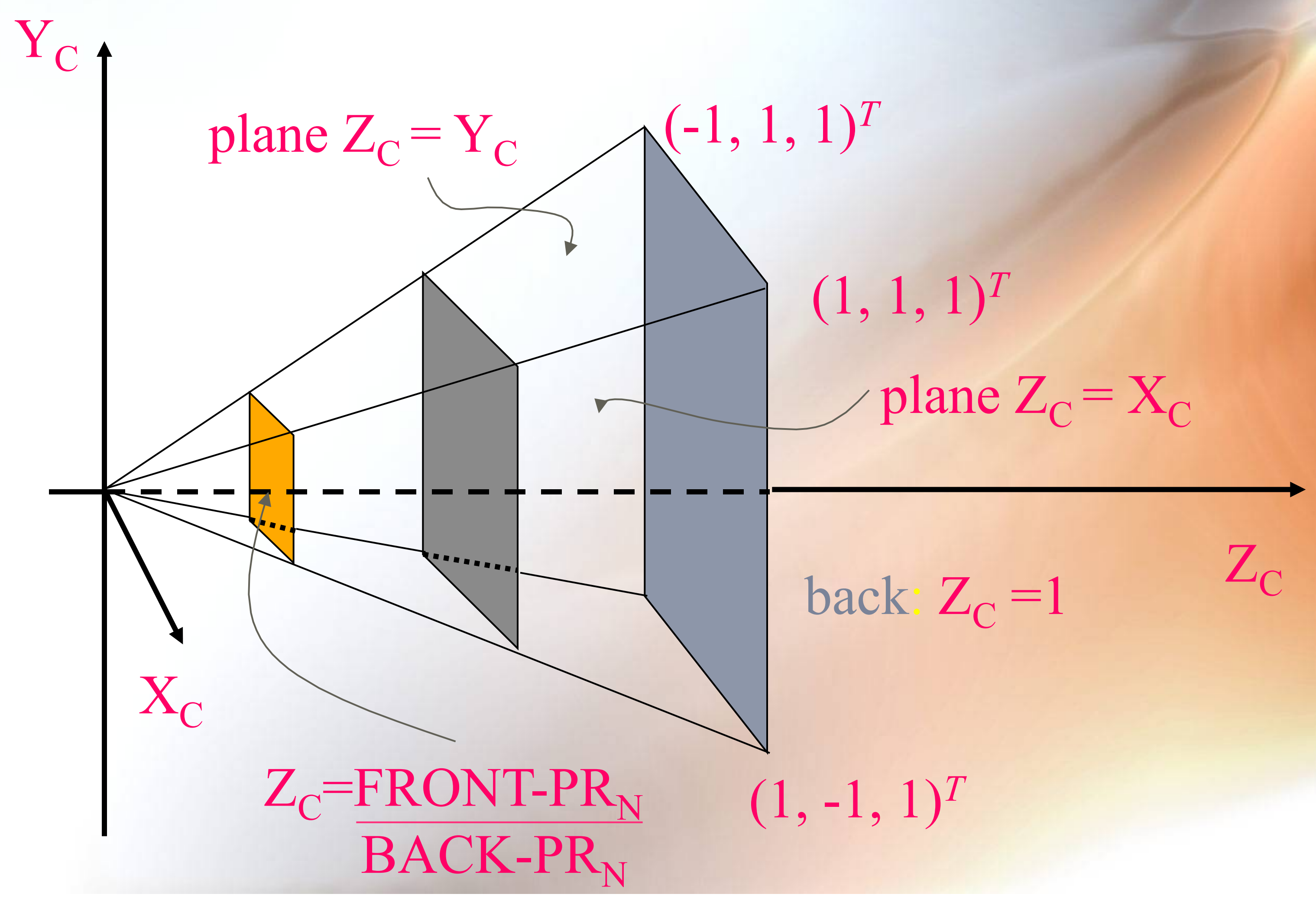


Notice that the view pyramid is not a right pyramid.
We must make it so with the shear transformation

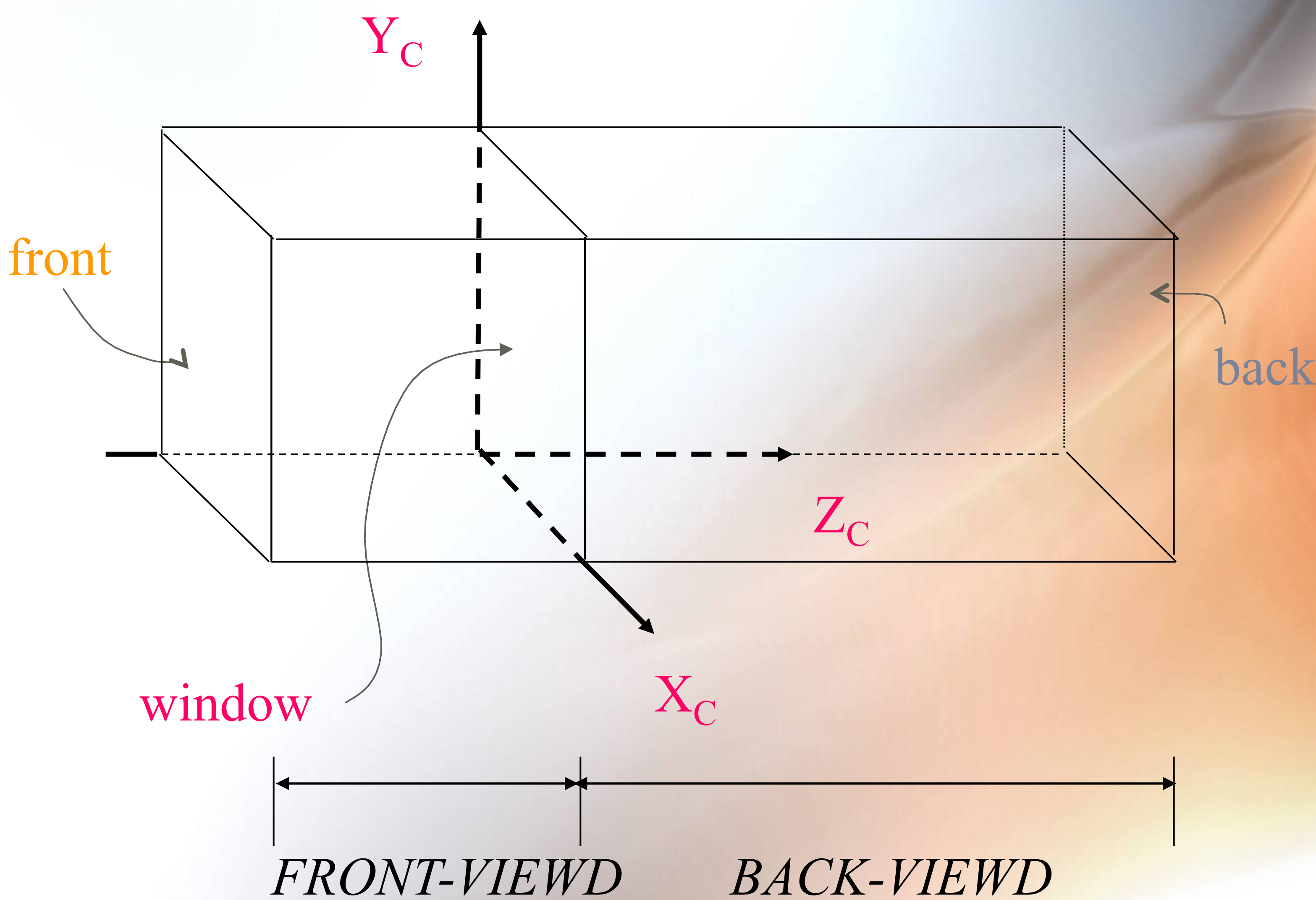
Scaling to Standard View Volume



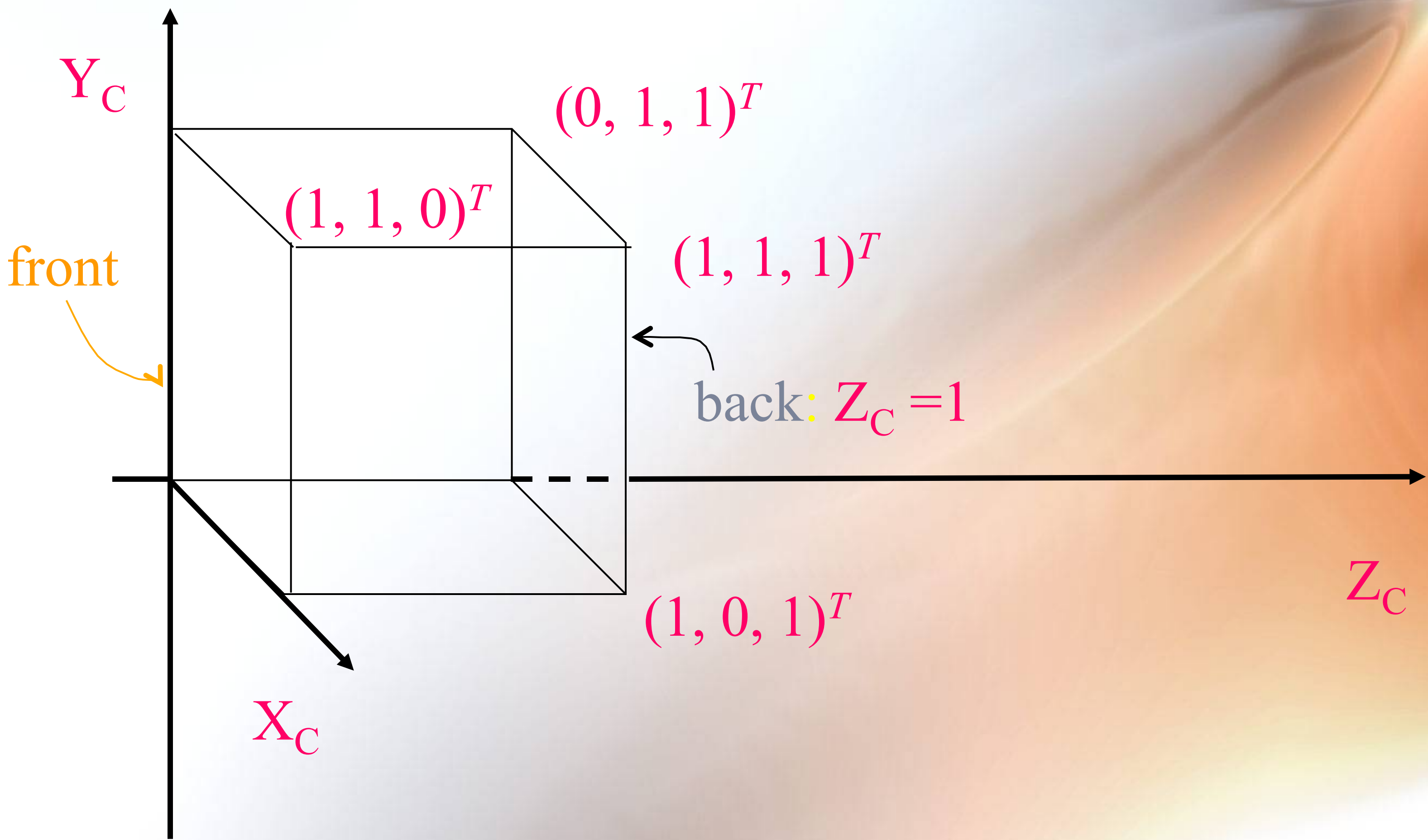
The Standard View Volume for Perspective Case



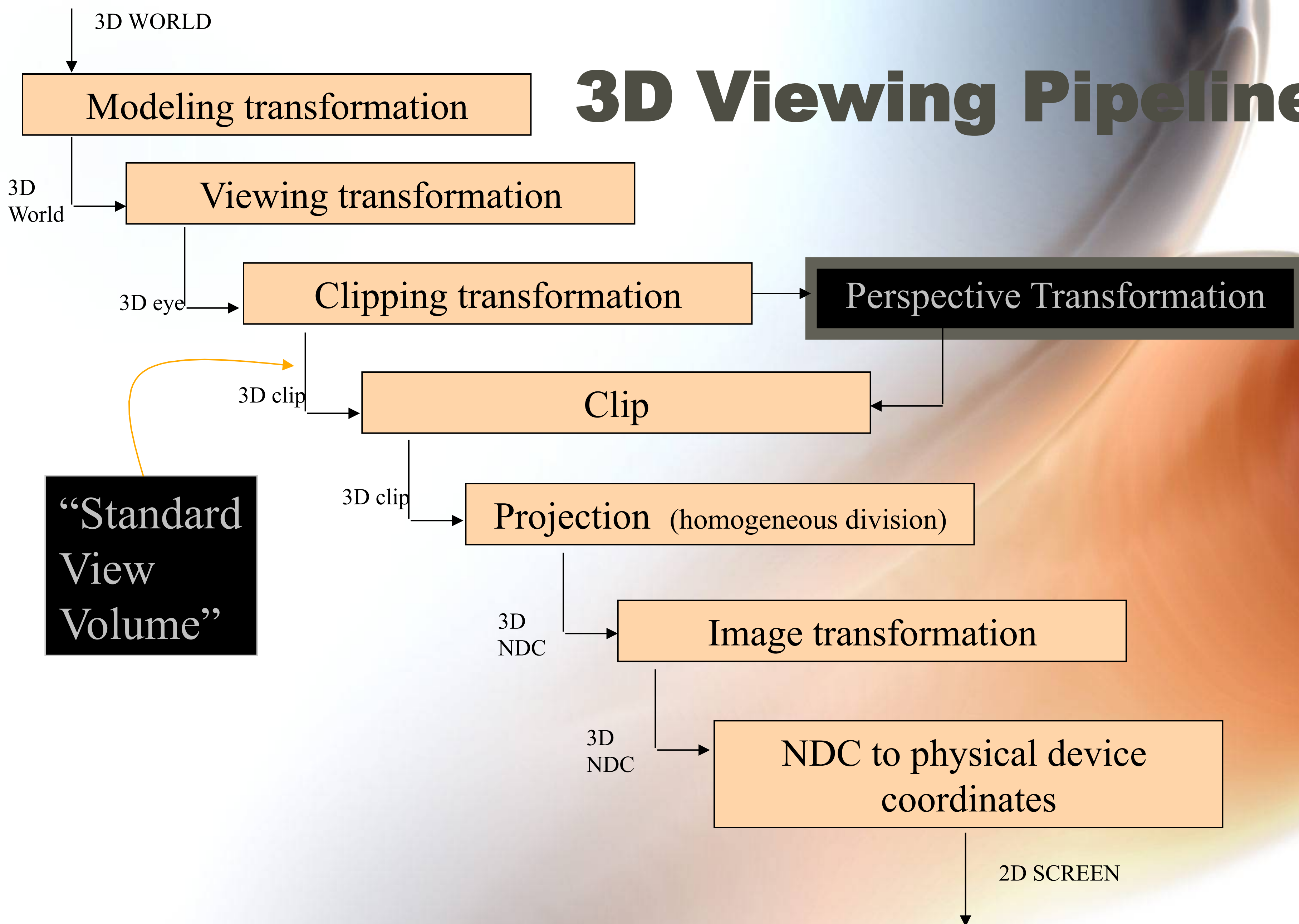
Scaling to Standard View Volume: Parallel



The Standard View Volume for Parallel: The Unit Cube $[0, 1]^3$



3D Viewing Pipeline



Projections

- The default projection in the eye (camera) frame is orthogonal
- For points within the default view volume

$$x_p = x$$
$$y_p = y$$
$$z_p = 0$$

Normalization

- **Most graphics systems use *view normalization***
  - **All other views are converted to the default view by transformations that determine the projection matrix**
  - **Allows use of the same pipeline for all views**

Homogeneous Coordinate Representation

$\mathbf{x_p = x}$
$\mathbf{y_p = y}$
$\mathbf{z_p = 0}$
$\mathbf{w_p = 1}$

default orthographic projection

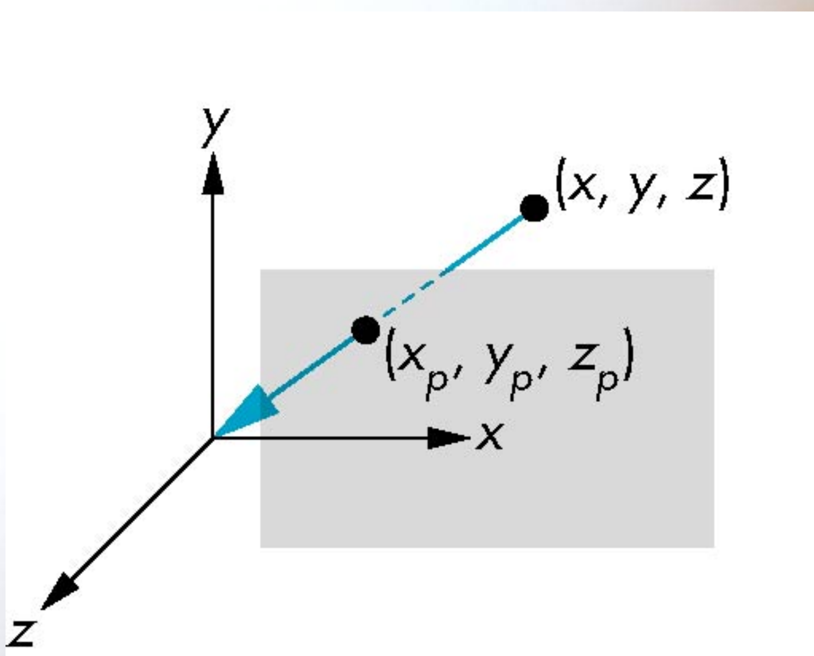
$$\mathbf{p}_\mathrm{p} = \mathbf{Mp}$$

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In practice, we can let $\mathbf{M} = \mathbf{I}$ and set the z term to zero later

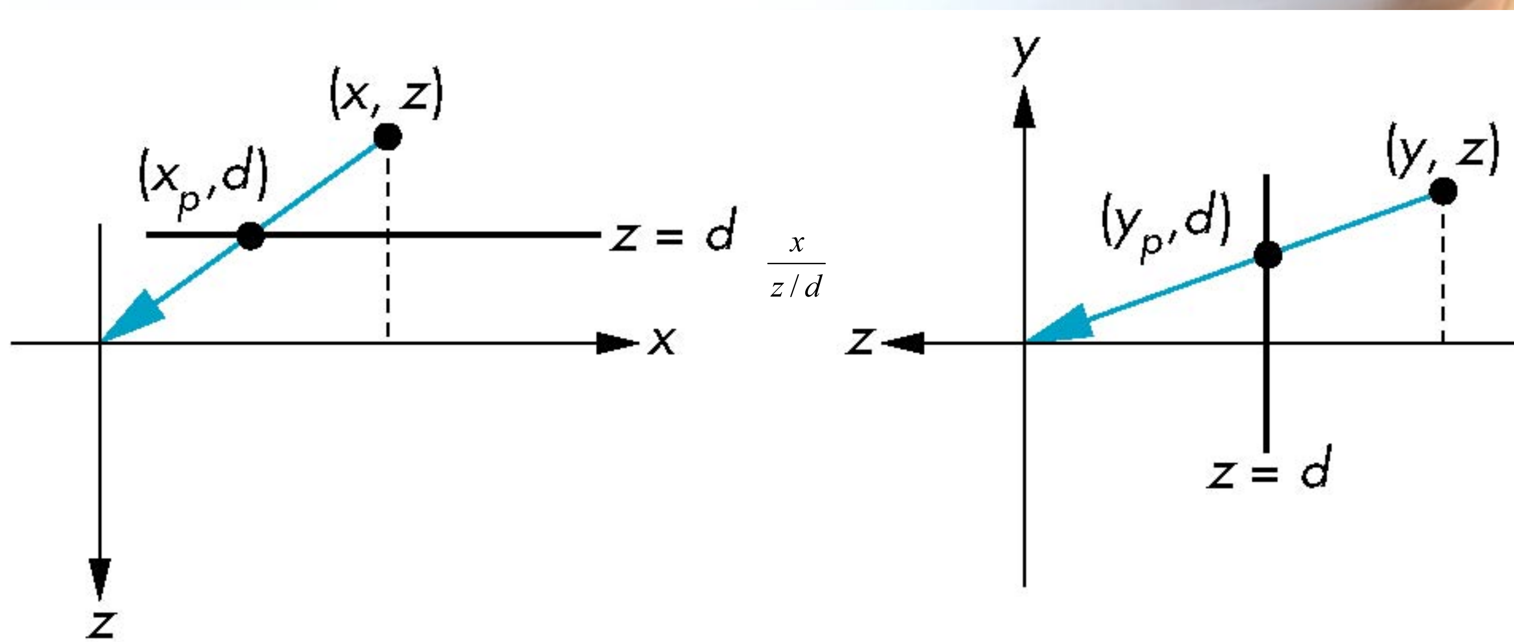
Simple Perspective

- Center of projection at the origin
- Projection plane $z = d$, $d < 0$



Perspective Equations

Consider top and side views



$$x_p = \frac{x}{z/d} \qquad y_p = \frac{y}{z/d} \qquad z_p = d$$

Normalize Homogeneous Coordinates (Perspective Only)

$$x' = \frac{x}{w}$$

$$y' = \frac{y}{w}$$

$$z' = \frac{z}{w}$$

provided $w \neq 0$

Returns x' and y' in range [-1, 1]
z' in range [0, 1]

Homogeneous Coordinate Form

consider $q = Mp$ **where**

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

$$\mathbf{q} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} \Rightarrow \quad \mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Perspective Division

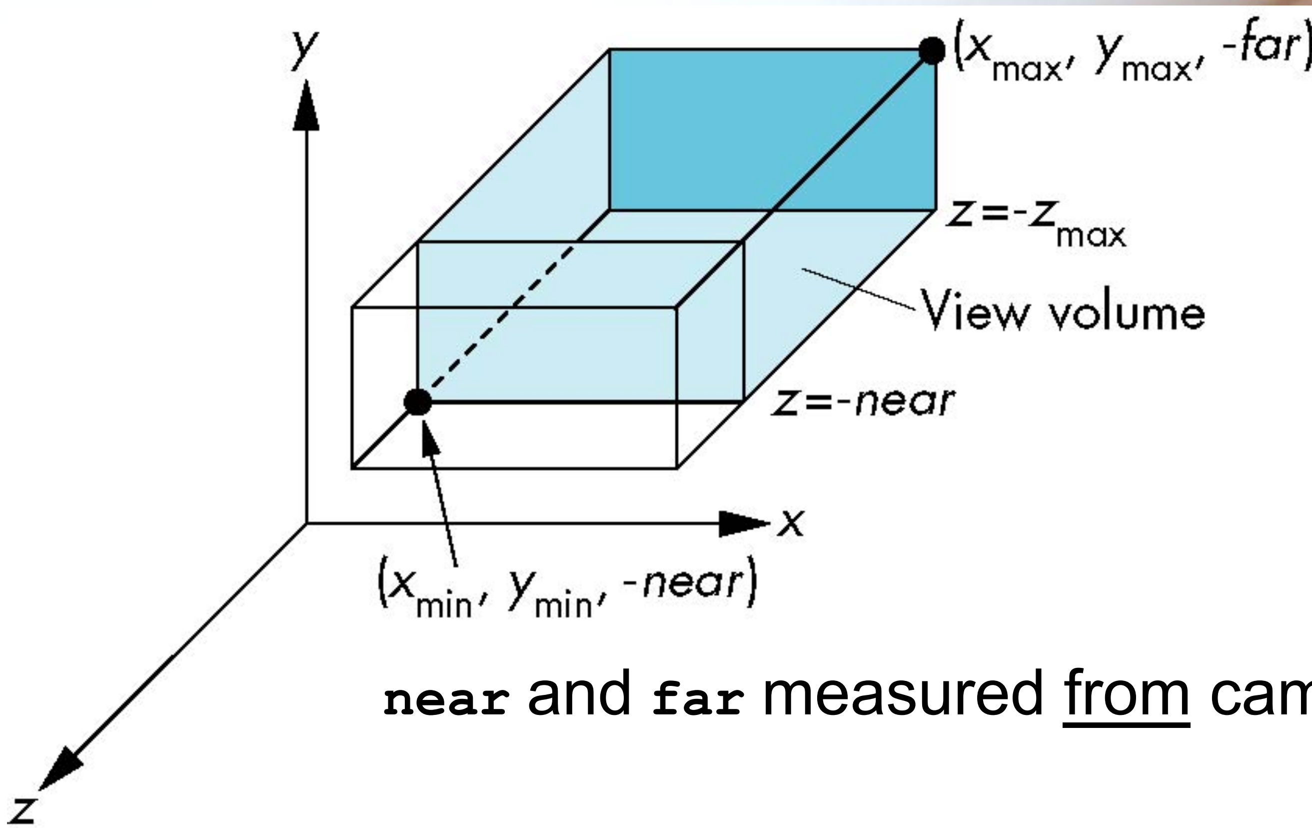
- However $w \neq 1$, so we must divide by w to return from homogeneous coordinates
- This *perspective division* yields

$$x_p = \frac{x}{z/d} \qquad y_p = \frac{y}{z/d} \qquad z_p = d$$

the desired perspective equations
- We will consider the corresponding clipping volume with the OpenGL functions

OpenGL Orthogonal Viewing

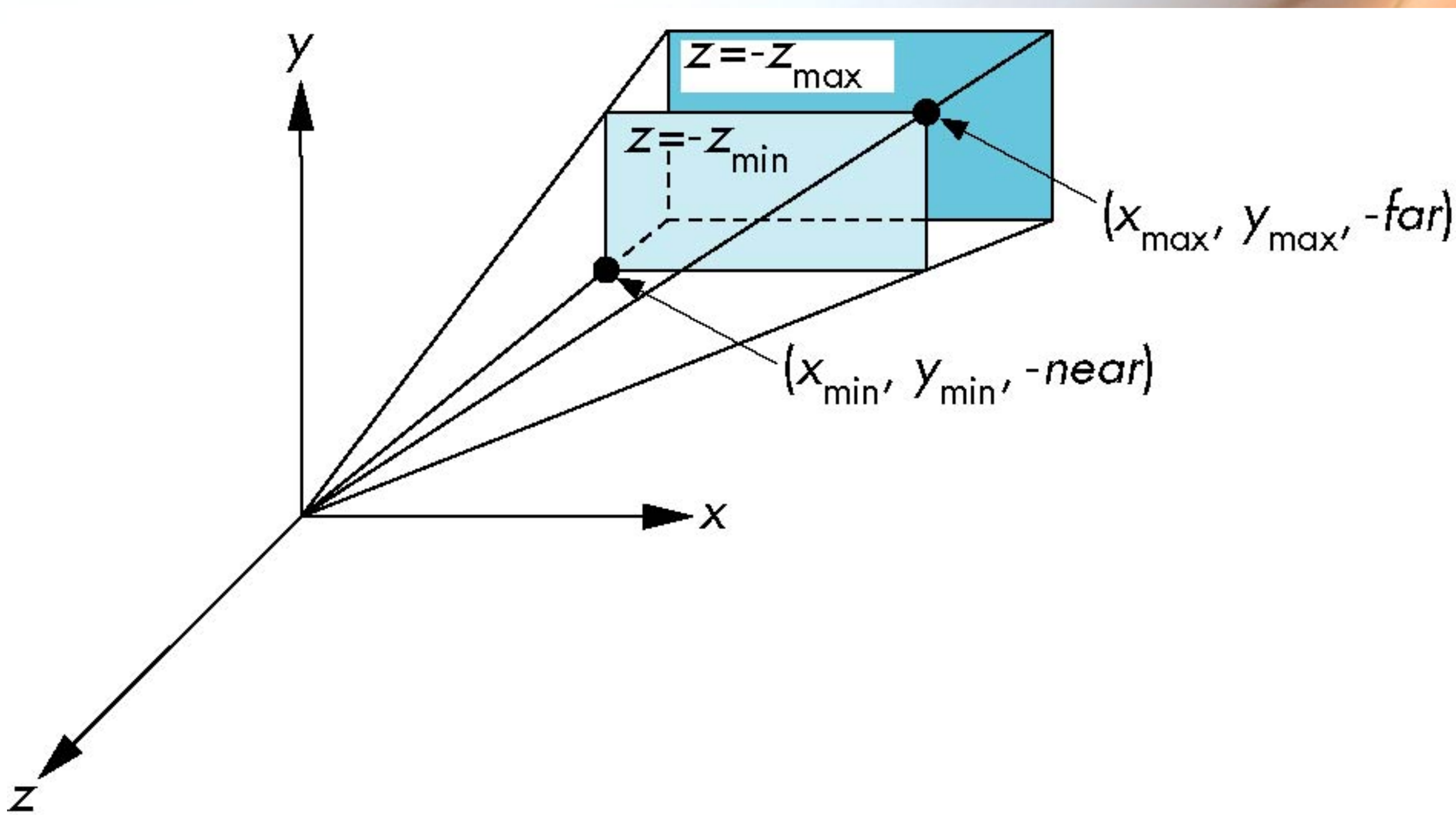
```
glOrtho(xmin,xmax,ymin,ymax,near,far)
glOrtho(left,right,bottom,top,near,far)
```



`near` and **`far`** measured <u>from</u> camera

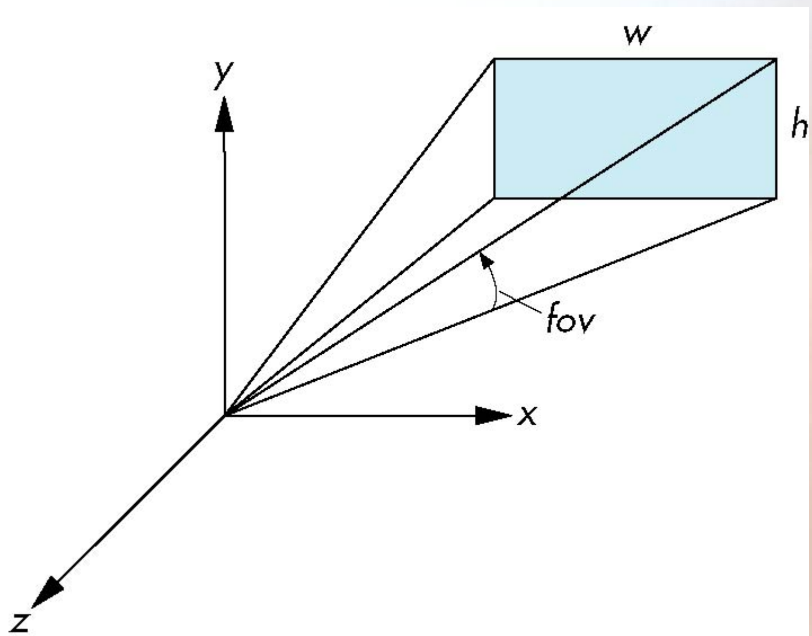
OpenGL Perspective

`glFrustum(xmin,xmax,ymin,ymax,near,far)`



Using Field of View

- **With `glFrustum` it is often difficult to get the desired view**
- **`gluPerpective(fovy, aspect, near, far)` often provides a better interface**



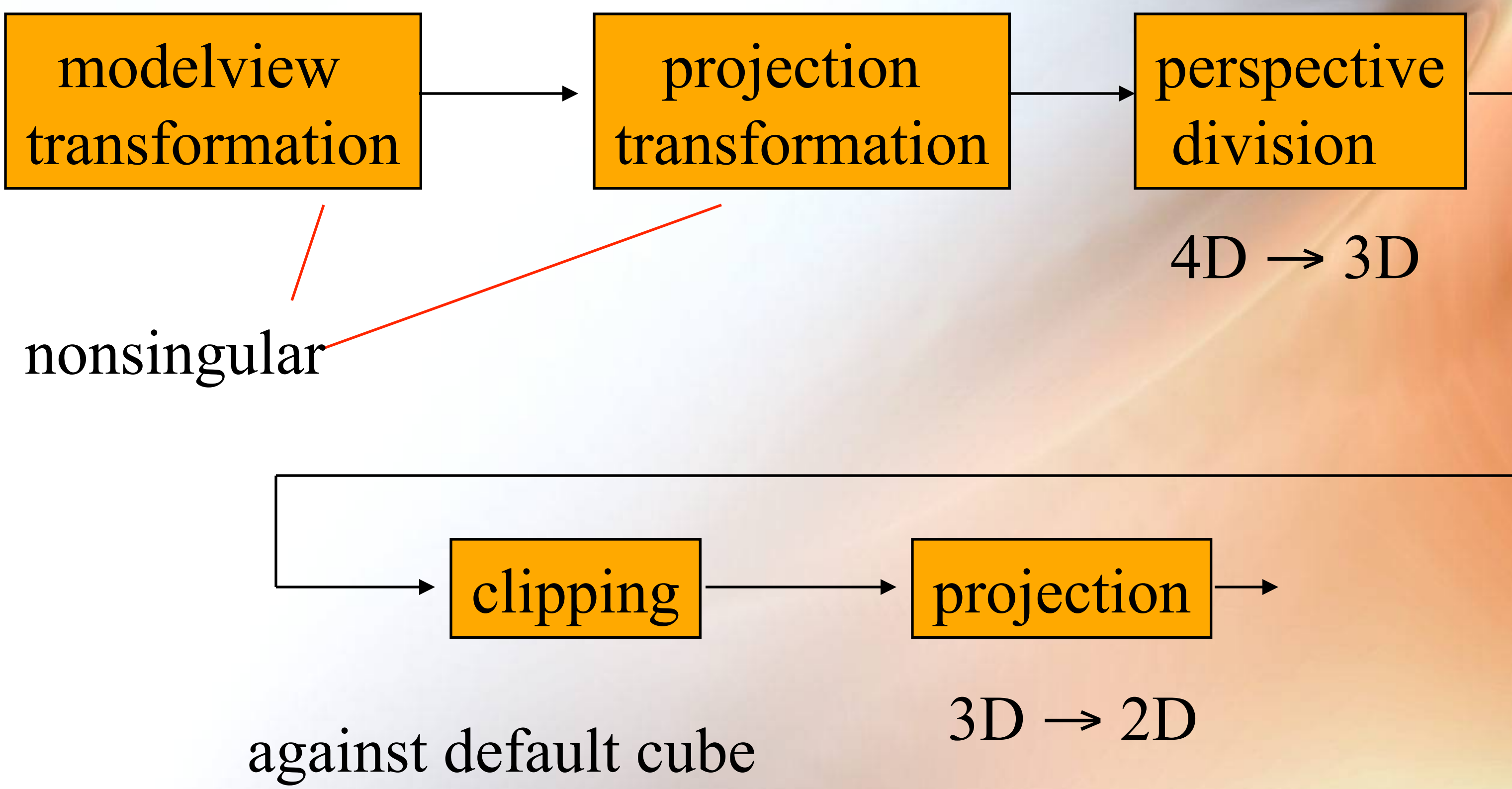
front plane

`aspect = w/h`

Normalization

- Rather than derive a different projection matrix for each type of projection, we can convert all projections to orthogonal projections with the default view volume
- This strategy allows us to use standard transformations in the pipeline and makes for efficient clipping

Pipeline View



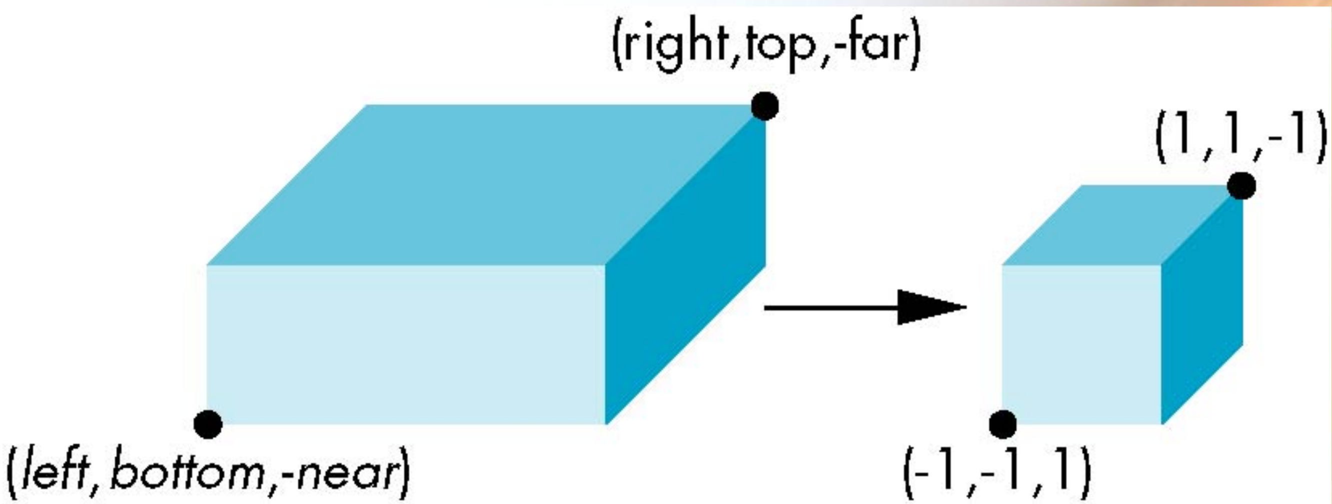
Notes

- We stay in 4D homogeneous coordinates through both the modelview and projection transformations
  - Both these transformations are nonsingular
  - Default to identity matrices (orthogonal view)
- Normalization lets us clip against simple cube regardless of type of projection
- Delay final projection until end
  - Important for hidden-surface removal to retain depth information as long as possible

Orthogonal Normalization

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

normalization ⇒ find transformation to convert specified clipping volume to default



Orthogonal Matrix

- Two steps
  - Move center to origin

    `T(-(right+left)/2, -(top+bottom)/2,(near+far)/2))`
  - Scale to have sides of length 2

    `S(2/(right-left),2/(top-bottom),2/(near-far))`

$$\mathbf{P} = \mathbf{ST} = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{2}{near-far} & \frac{far+near}{near-far} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Final Projection

- **Set $z = 0$**
- **Equivalent to the homogeneous coordinate transformation**

$$\mathbf{M}_{\text{orth}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **Hence, general orthogonal projection in 4D is**

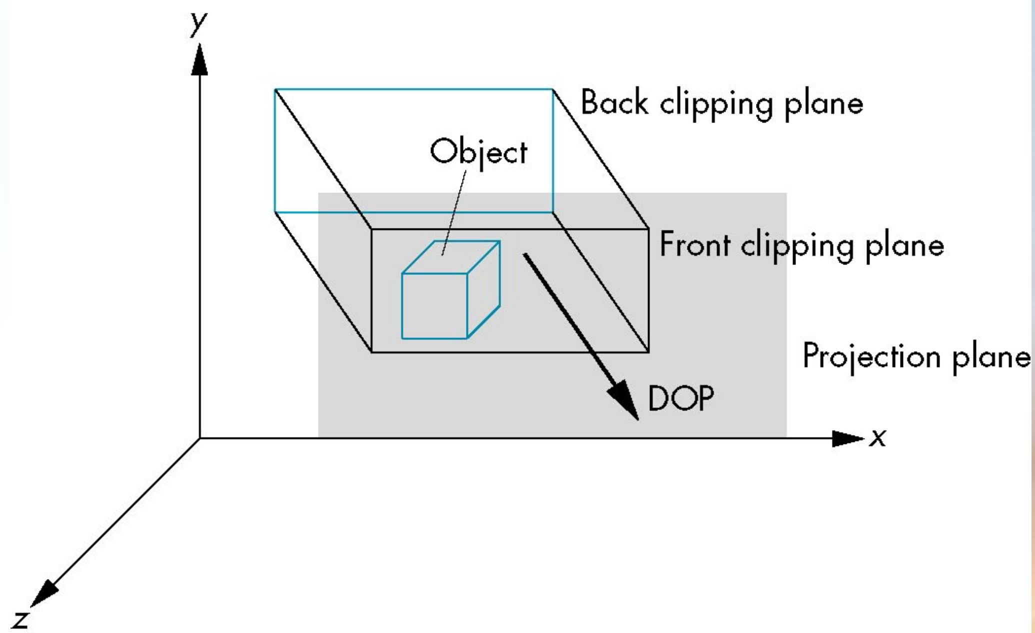
$$\mathbf{P} = \mathbf{M}_{\text{orth}}\mathbf{S}\mathbf{T}$$

Oblique Projections

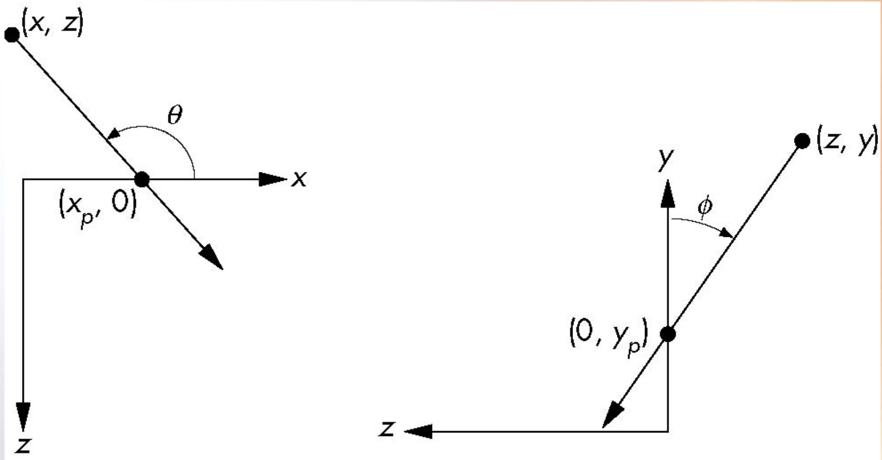
- The OpenGL projection functions cannot produce general parallel projections such as

- However if we look at the example of the cube it appears that the cube has been sheared
- Oblique Projection = Shear + Orthogonal Projection

General Shear



top view



side view

Shear Matrix

xy shear (*z* values unchanged)

$$\mathbf{H}(\theta,\phi) = \begin{bmatrix} 1 & 0 & -\cot\theta & 0 \\ 0 & 1 & -\cot\varphi & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

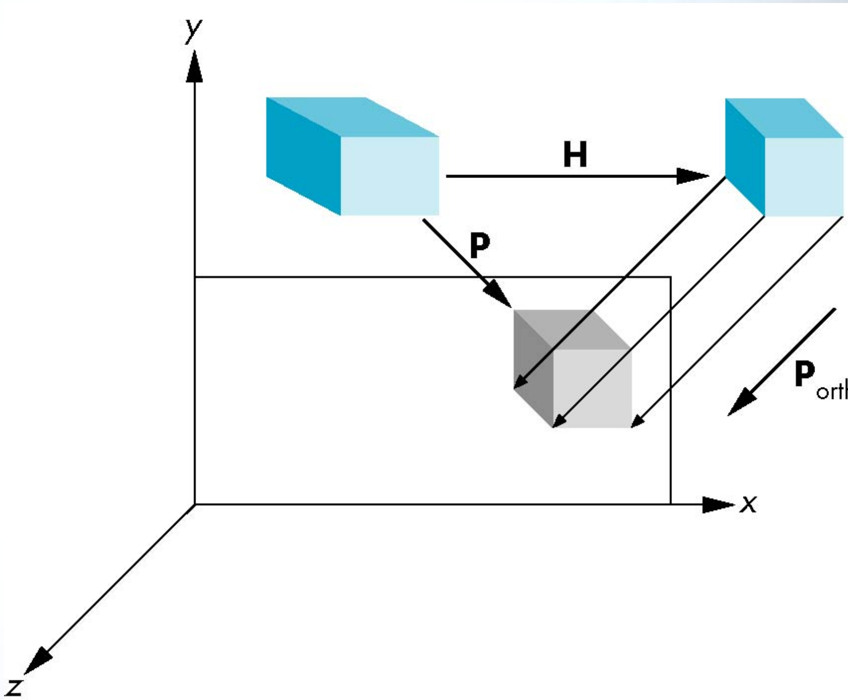
Projection matrix

$$\mathbf{P} = \mathbf{M}_{\text{orth}}\, \mathbf{H}(\theta,\phi)$$

General case:

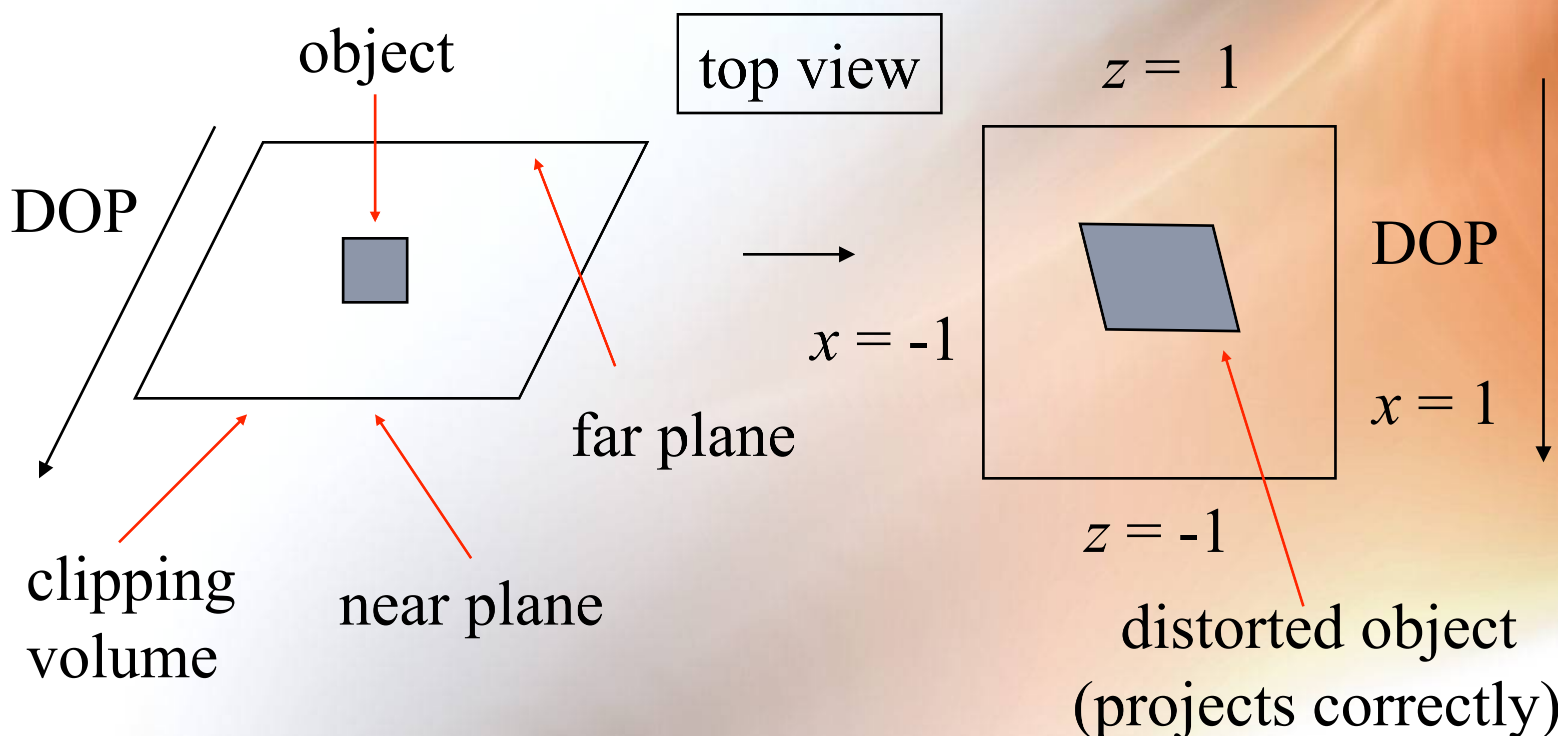
$$\mathbf{P} = \mathbf{M}_{\text{orth}}\, \mathbf{STH}(\theta,\phi)$$

Equivalency



Effect on Clipping

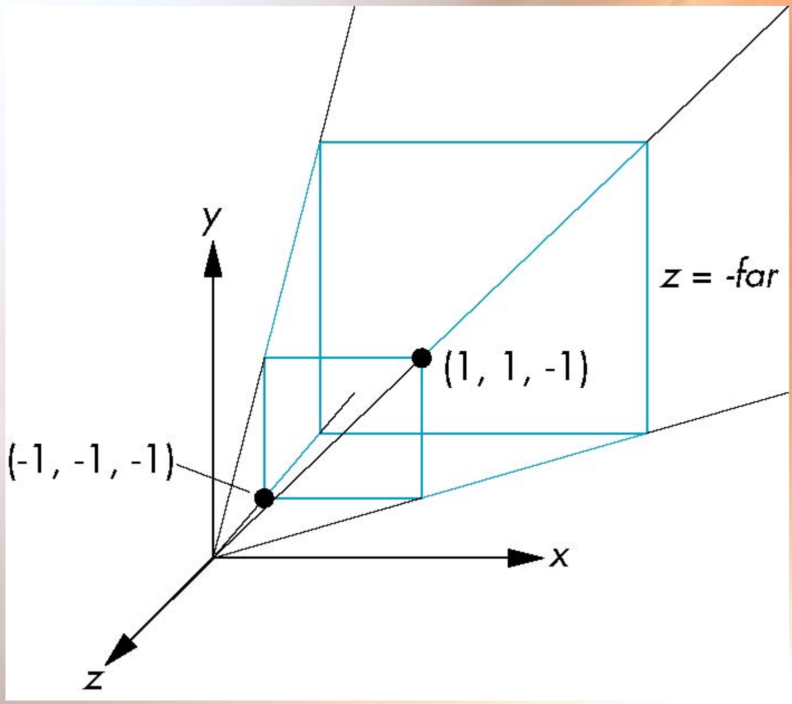
- **The projection matrix** P = STH **transforms the original clipping volume to the default clipping volume**



Simple Perspective

Consider a simple perspective with the COP at the origin, the near clipping plane at z = -1, and a 90 degree field of view determined by the planes

$x = \pm z,\ y = \pm z$



Perspective Matrices

Simple projection matrix in homogeneous coordinates

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

Note that this matrix is independent of the far clipping plane

Generalization

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

after perspective division, the point $(x, y, z, 1)$ goes to

$$x'' = -x/z$$
$$y'' = -y/z$$
$$z'' = -(\alpha+\beta/z)$$

which projects orthogonally to the desired point regardless of α and β

Picking α and β

If we pick

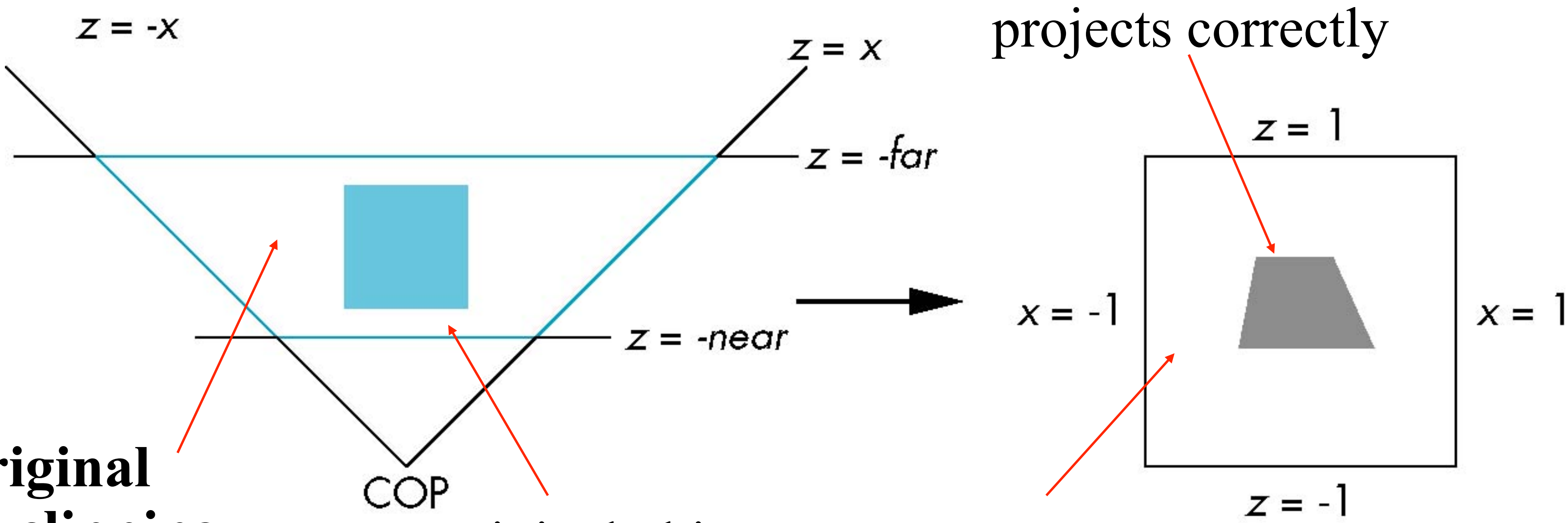
$$\alpha = \frac{\text{near} + \text{far}}{\text{near - far}}$$

$$\beta = \frac{2\text{near} * \text{far}}{\text{far - near}}$$

the near plane is mapped to $z = -1$
the far plane is mapped to $z = 1$
and the sides are mapped to $x = \pm 1, y = \pm 1$

Hence the new clipping volume is the default clipping volume

Normalization Transformation



distorted object projects correctly

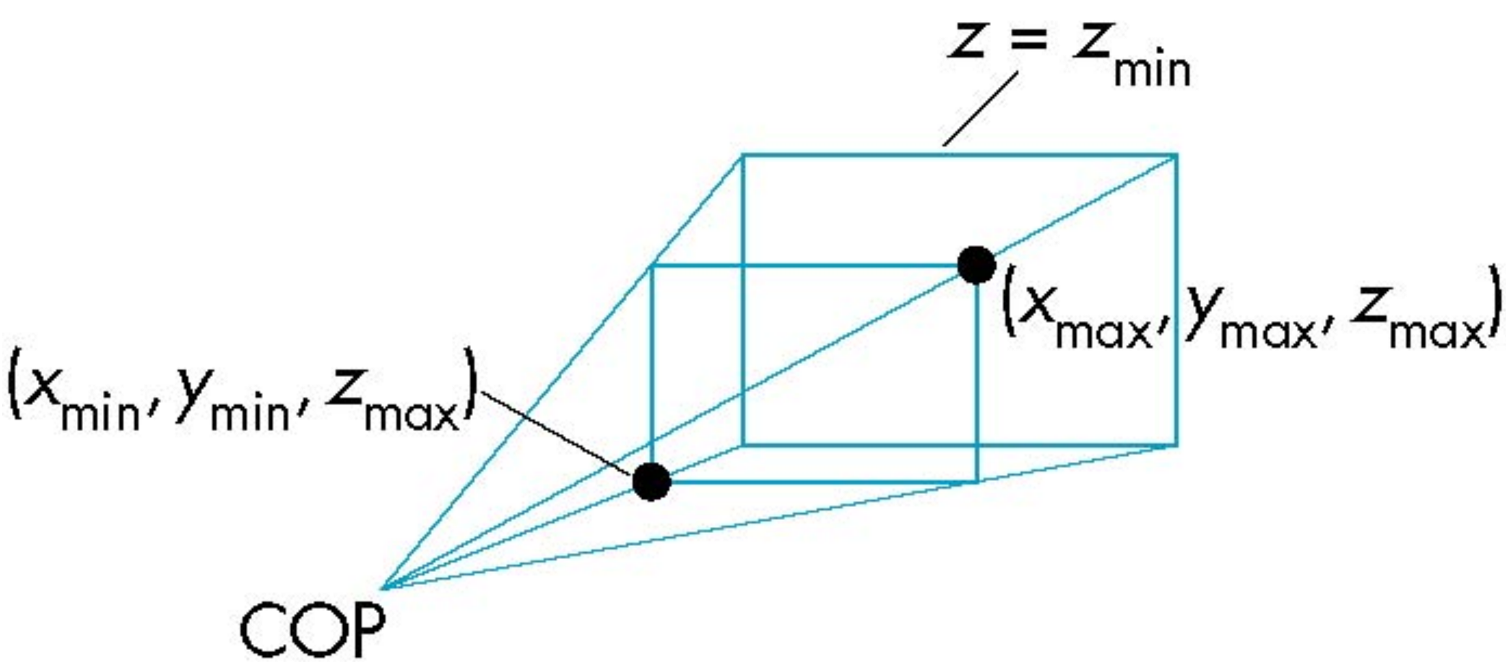
original clipping volume

original object

new clipping volume

OpenGL Perspective

- `glFrustum` allows for an asymmetric viewing frustum (although `gluPerspective` does not)



OpenGL Perspective Matrix

- The normalization in `glFrustum` requires an initial shear to form a right viewing pyramid, followed by a scaling to get the normalized perspective volume. Finally, the perspective matrix results in needing only a final orthogonal transformation

$$P = NSH$$

our previously defined perspective matrix

shear and scale

OpenGL Perspective Matrix

- H (shear): skew the point ((left+right)/2, (top +bottom)/2, -near) to (0, 0, -near)
- S (scale): scale the sides to $x = \pm z$, $y = \pm z$
- N (normalization): get the far plan to z = -1 and the near plane to z = 1

$$P = NSH = \begin{bmatrix} \frac{-2 * near}{right - left} & 0 & \frac{right + left}{right - left} & 0 \\ 0 & \frac{-2 * near}{top - bottom} & \frac{top + bottom}{top - bottom} & 0 \\ 0 & 0 & \frac{far + near}{near - far} & \frac{2\,far * near}{far - near} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Why do we do it this way?

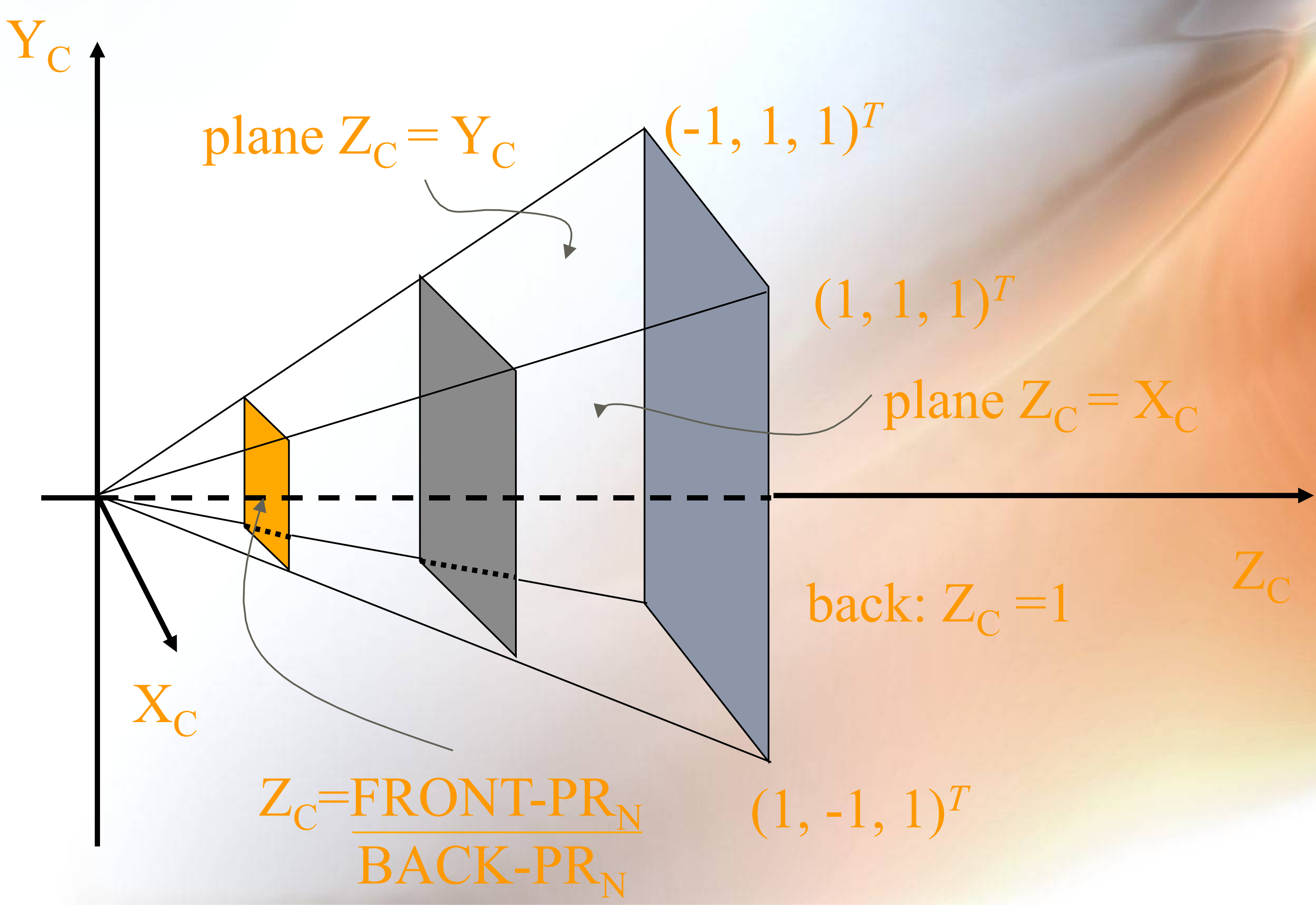
- Normalization allows for a single pipeline for both perspective and orthogonal viewing
- We keep in four dimensional homogeneous coordinates as long as possible to retain three-dimensional information needed for hidden-surface removal and shading
- We simplify clipping

View Volume Clipping Limits

	Parallel	Perspective
Above	y > 1	y > w
Below	y < 0	y < -w
Right	x > 1	x > w
Left	x < 0	x < -w
Behind (yon)	z > 1	z > w
In Front (hither)	z < 0	z < 0

A point (x, y, z) is in the view volume if and only if it lies inside these 6 planes.

Recall the Standard View Volume:



The Perspective Transformation (for the Perspective Case only)

- **Now that we have a normalized perspective view volume we apply one more matrix to it in order to permit simple depth comparisons**

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{1}{1 - Zfront} & \dfrac{-Zfront}{1 - Zfront} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

- **Where *Zfront* is the value of the front clipping z coordinate after viewing transformation --**

$(Front - PR_N) / (Back - PR_N)$

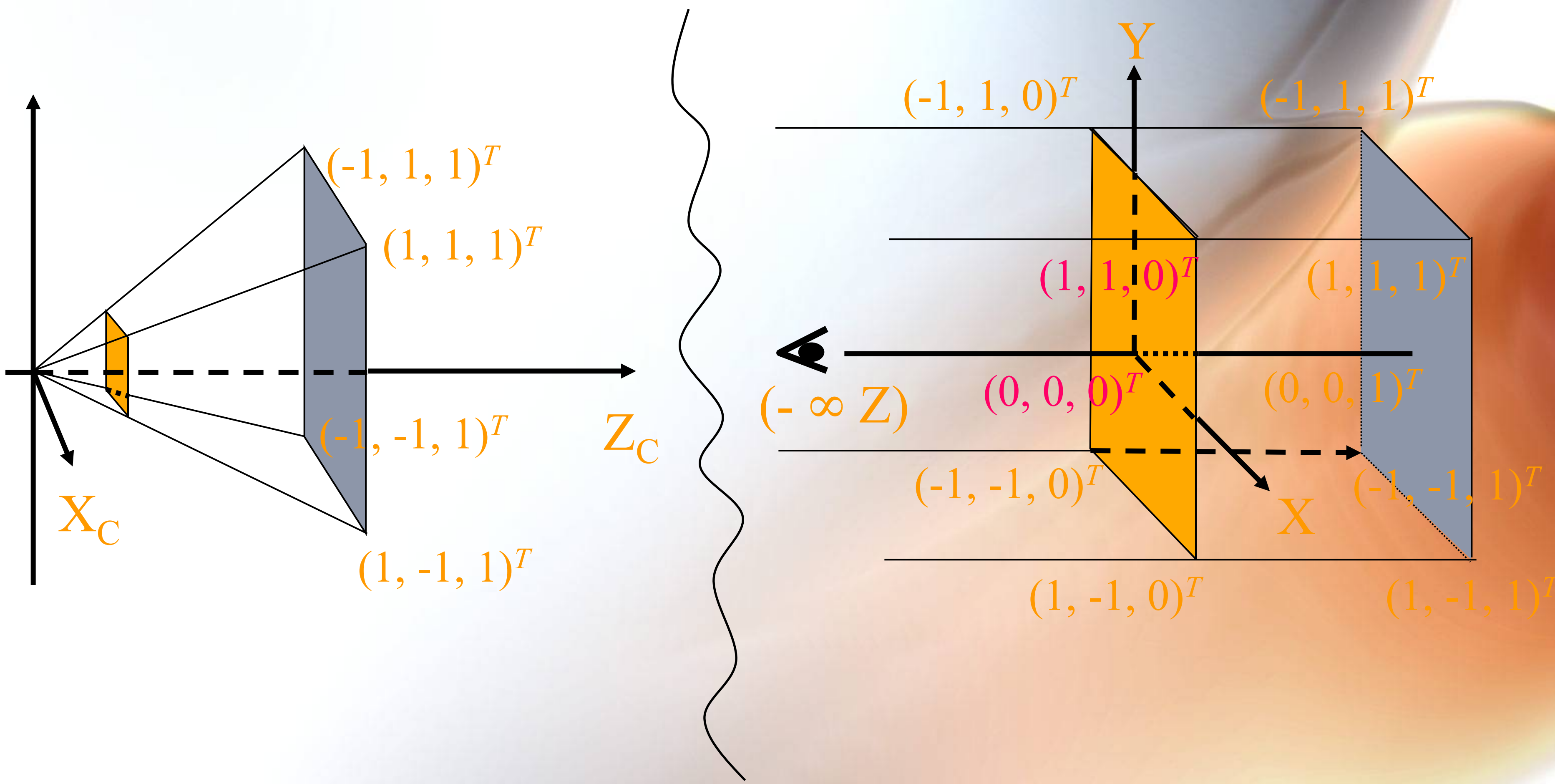
What does *M* do?

- Notice that *M* does not affect the x or y coordinates.
- *M* sets the homogeneous coordinate

  $w \leftarrow z$.
- z is changed to lie in the range [0, 1].
- Check: if $z = Zfront$ then new $z \leftarrow 0$;

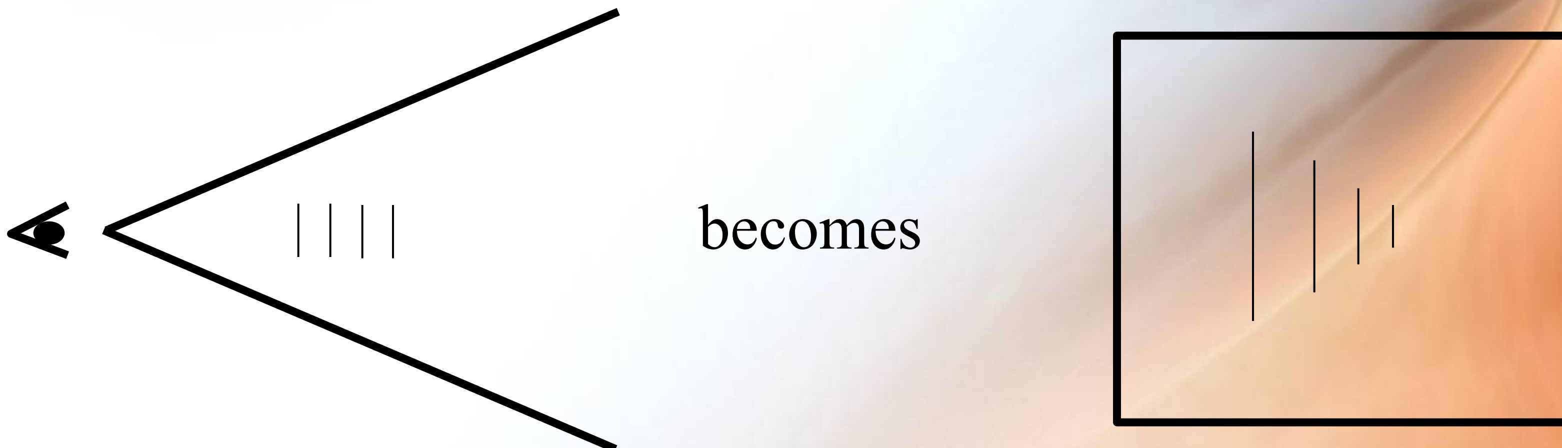
  if $z = 1$ then new $z \leftarrow 1$.

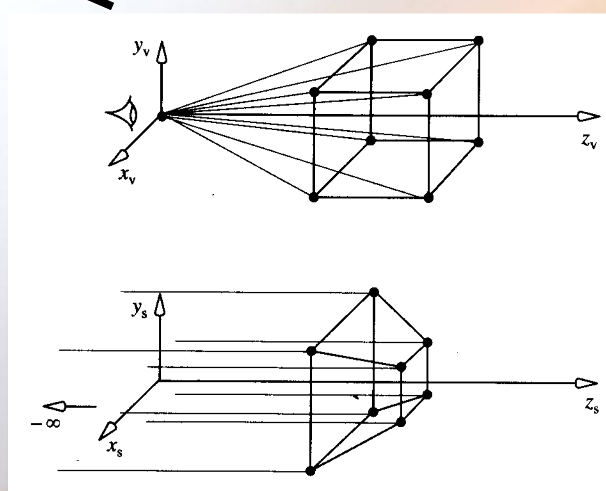
M Changes Standard Pyramid to This…



M Creates the Foreshortening Effect

- **Thus *M* makes the projectors parallel, allowing later depth comparisons:**





The Perspective Transformation *M*

- Preserves relative depth.
- Preserves linearity ("straightness").
- Preserves planarity.
- Produces perspective foreshortening.
- Still permits clipping -- just use W coordinate.