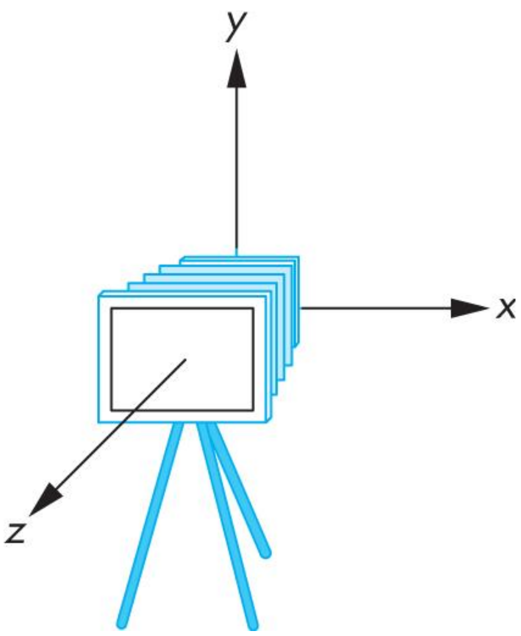
Perspective Matrix and its implementation

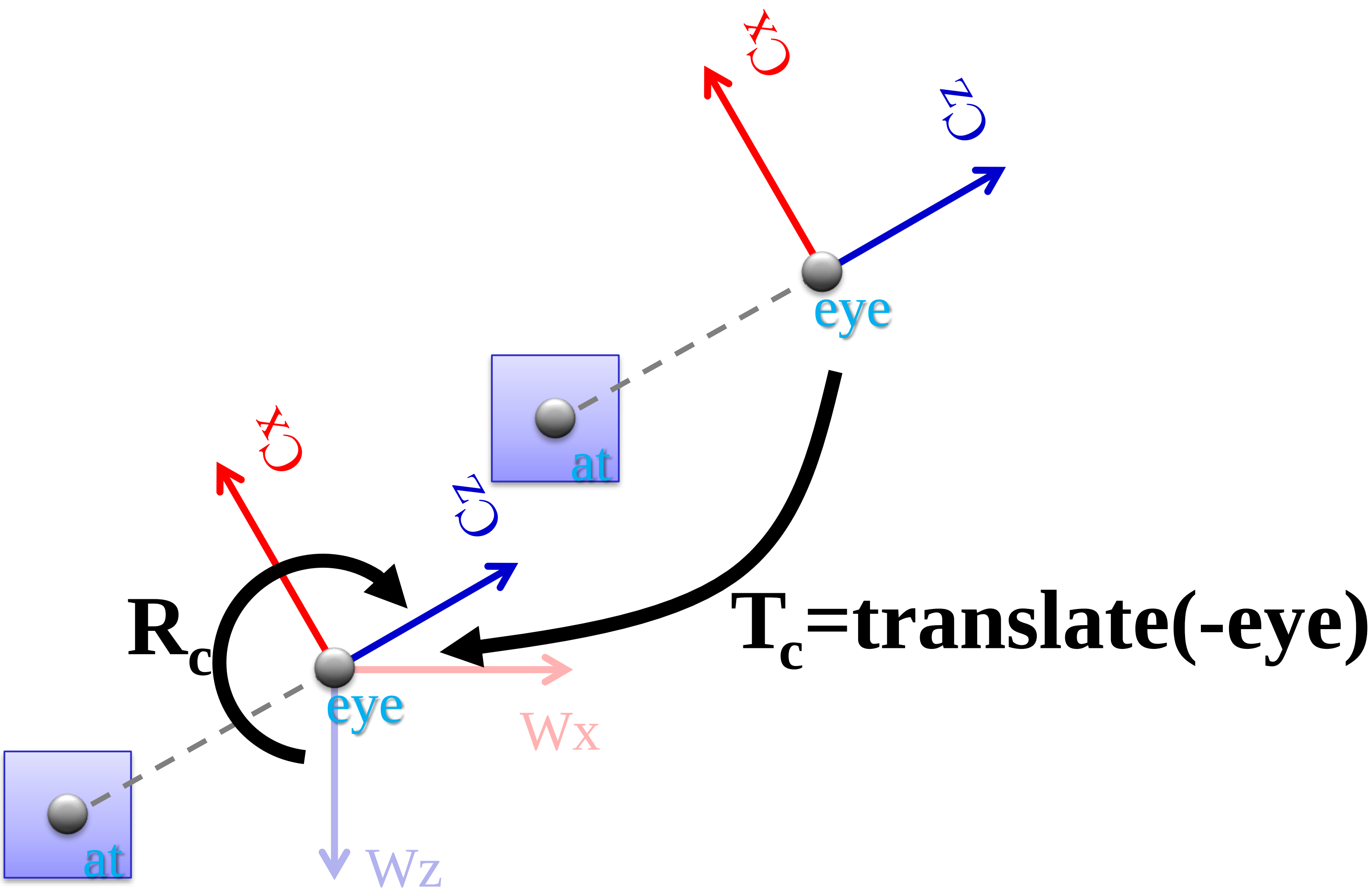
Sang Il Park

Review : Viewing in OpenGL

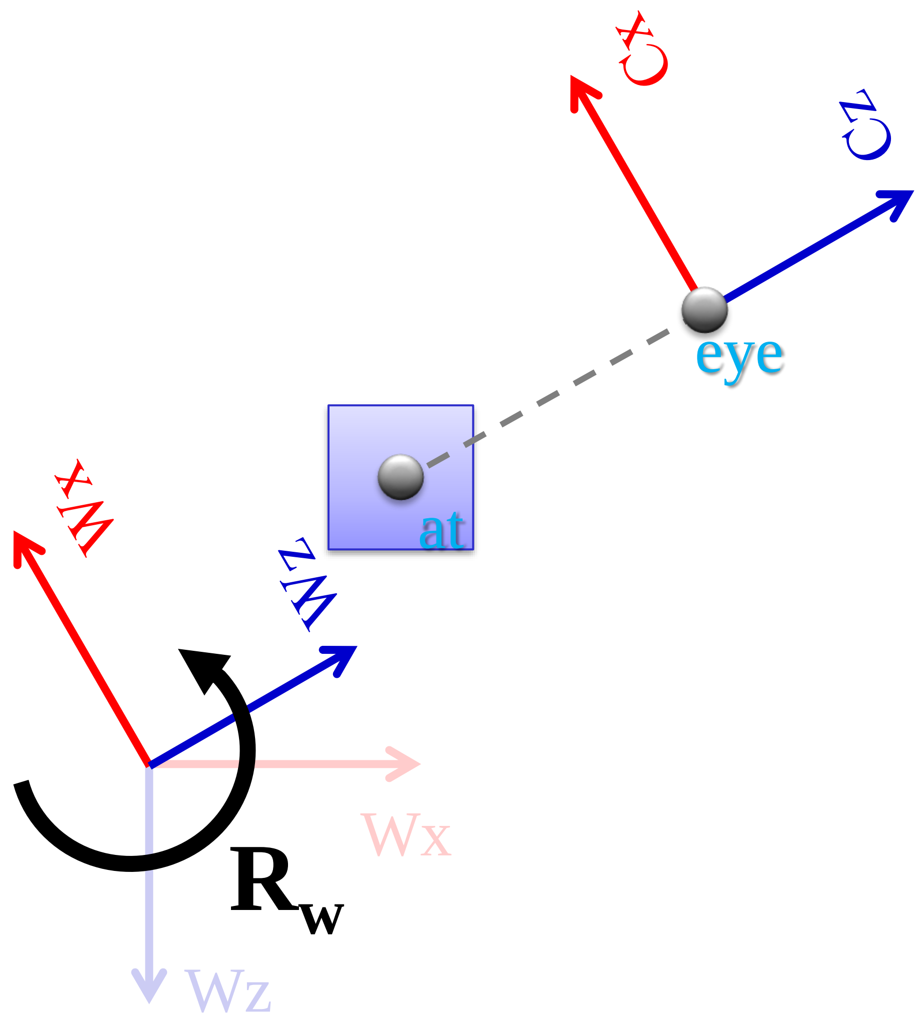
- Remember:
  camera is pointing in the negative z direction



Review : Translate + Rotation



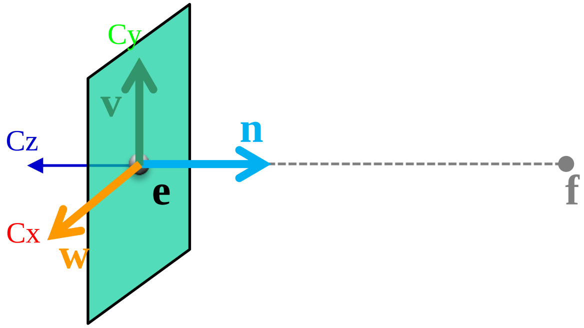
Review : Think about inverse transform



$$R_c = (R_w)^{-1}$$
$$= (R_w)^T$$

Review : Summary of Rotation

- **gluLookAt (ex,ey,ez, fx,fy,fz, ux,uy,uz);**
- n = (f – e) / |f – e|
- v = (u – (u · n) n) / |u – (u · n) n|
- w = n x v



- Rotation must map:
  - (1,0,0) to w
  - (0,1,0) to v
  - (0,0,-1) to n

$$\mathbf{R}_w = \begin{bmatrix} w_x & v_x & -n_x & 0 \\ w_y & v_y & -n_y & 0 \\ w_z & v_z & -n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Review : Putting All Together

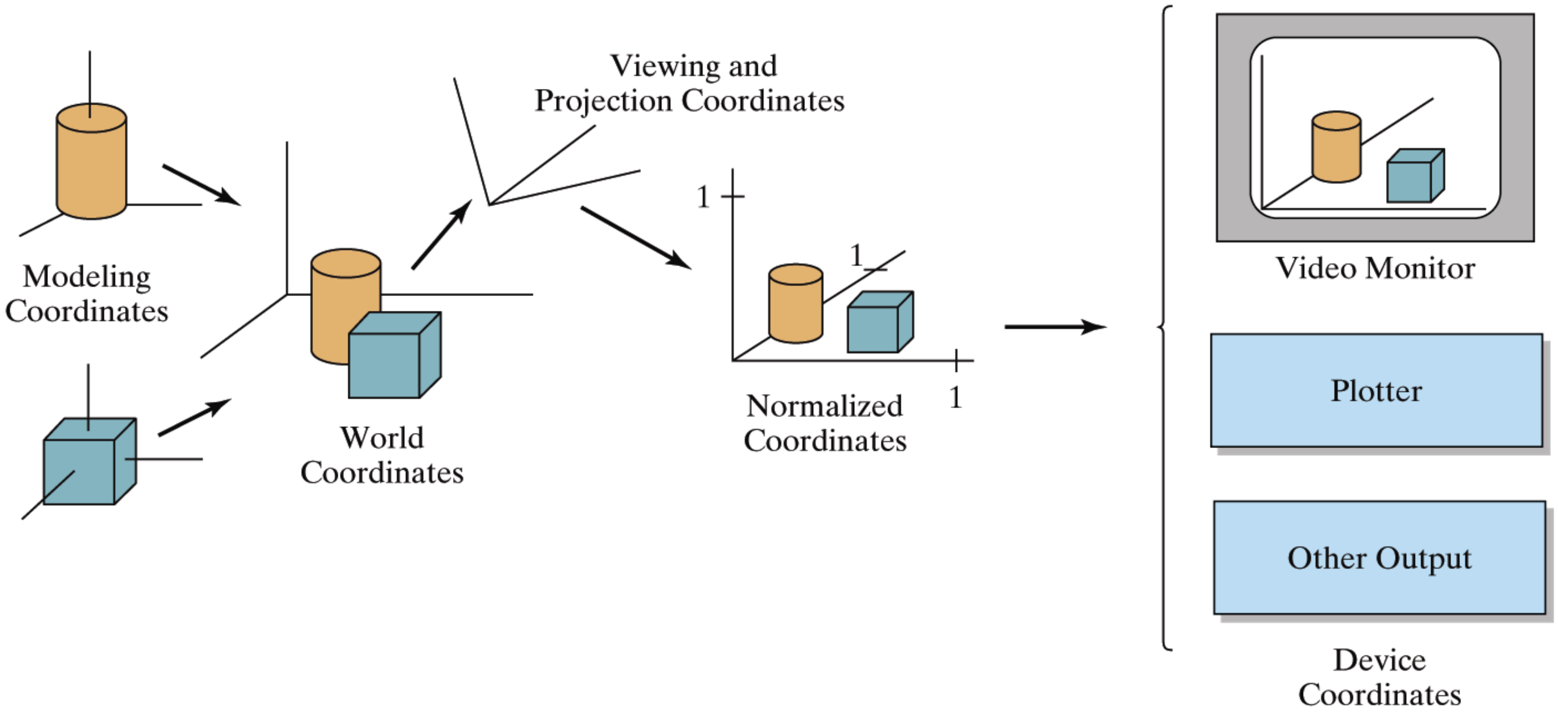
- Calculate V = $R_w^{-1}\ T_w^{-1}$

$$\mathbf{V} = \mathbf{R}^{-1}\mathbf{T}^{-1} = \begin{bmatrix} w_x & w_y & w_z & -w_x e_x - w_y e_y - w_z e_z \\ v_x & v_y & v_z & -v_x e_x - v_y e_y - v_z e_z \\ -n_x & -n_y & -n_z & n_x e_x + n_y e_y + n_z e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- This is different from book [Angel, Ch. 4.3.2]
- There, u, v, n are right-handed (here: u, v, -n)

OpenGL Geometric Transformations (old style)

- `glMatrixMode(GL_MODELVIEW);`



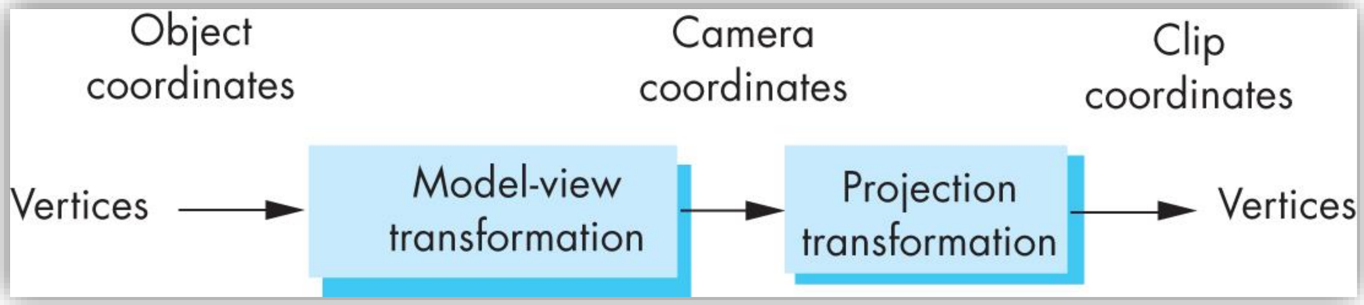
Projection Matrix and its implementation

Topics

- Simple Parallel Projections
- Simple Perspective Projections

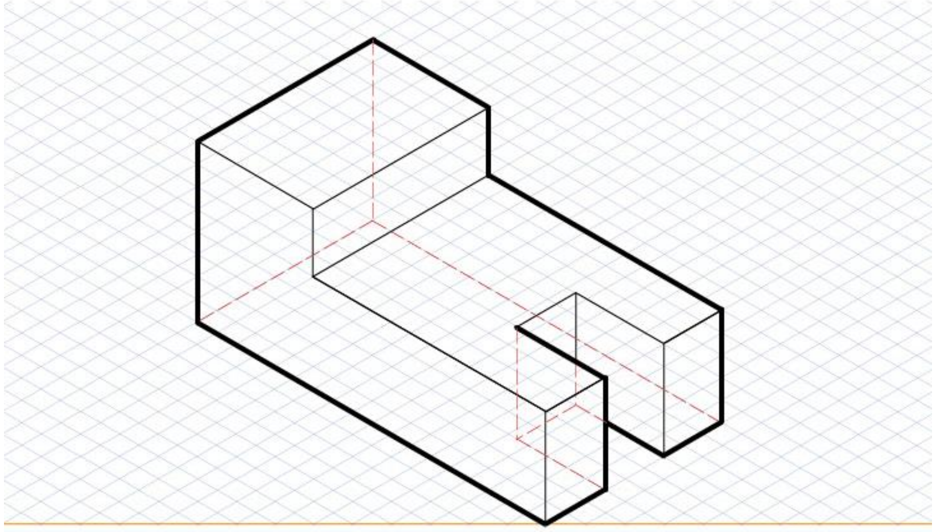
Projection Matrices

- Recall geometric pipeline



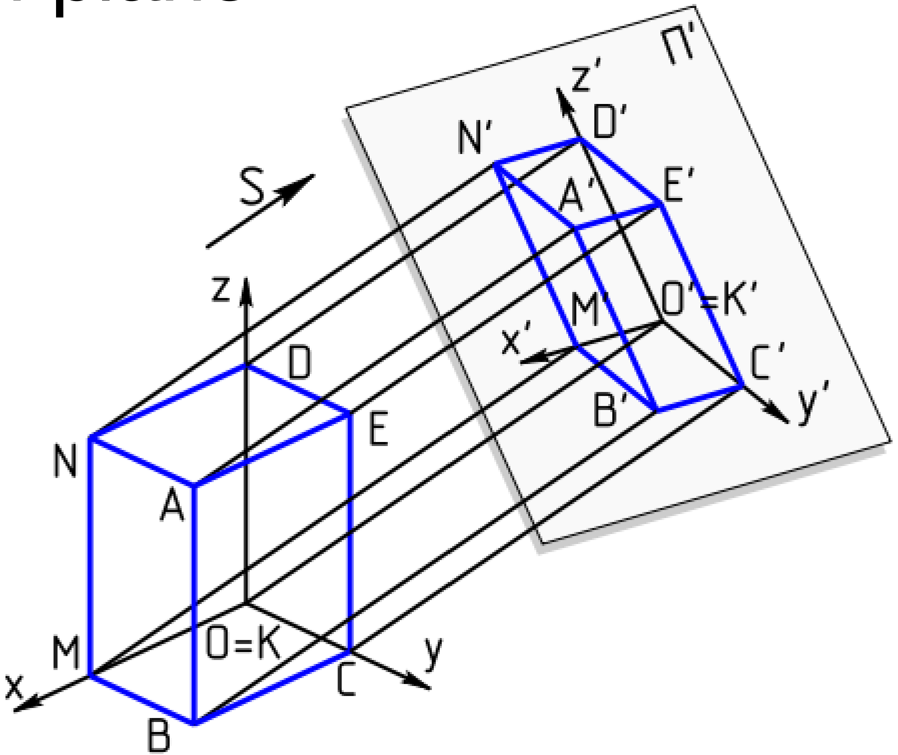
- Projection takes 3D to 2D
- Projections are not invertible
- Projections also described by 4x4 matrix
- Homogenous coordinates crucial
- Parallel and perspective projections

Simple Parallel Projection



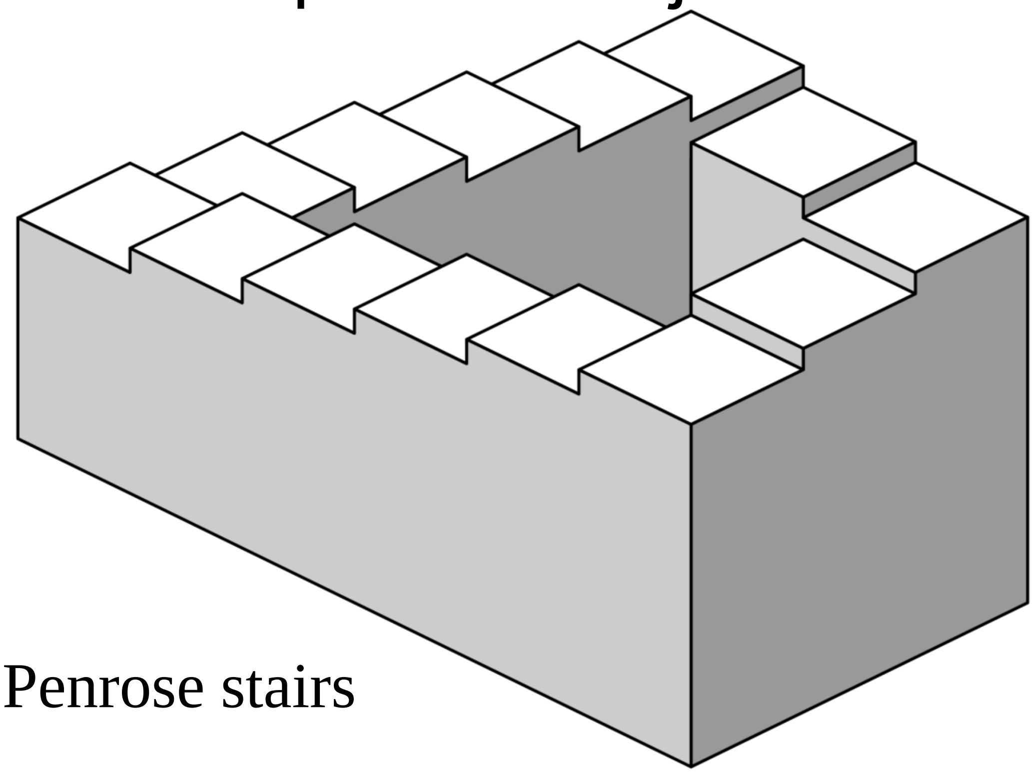
Parallel Projection

- Project 3D object to 2D via parallel lines
- The lines are not necessarily orthogonal to projection plane



Parallel Projection

- Problem: objects far away do not appear smaller
- Can lead to “impossible objects” :

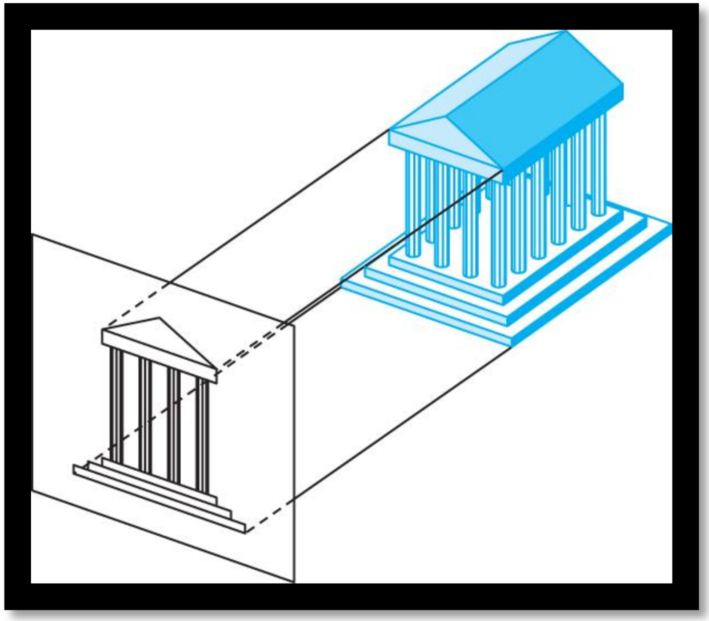


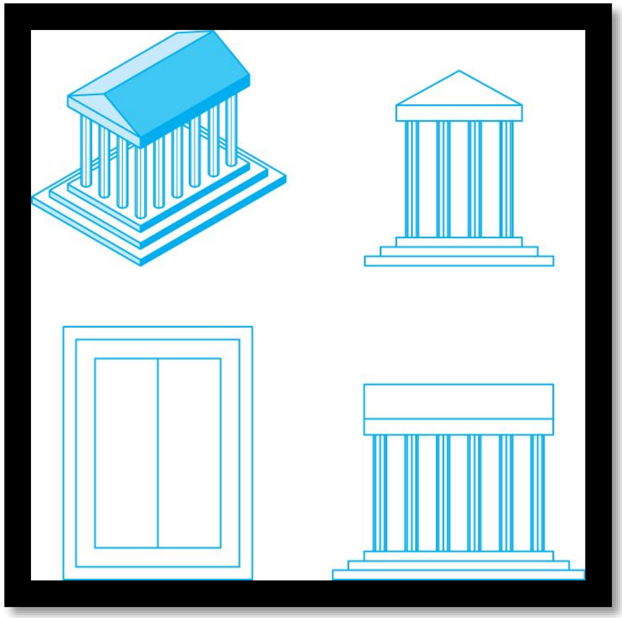
Penrose stairs

- Echochrome: https://www.youtube.com/watch?v=QfICeBtVy8U

Orthographic Projection

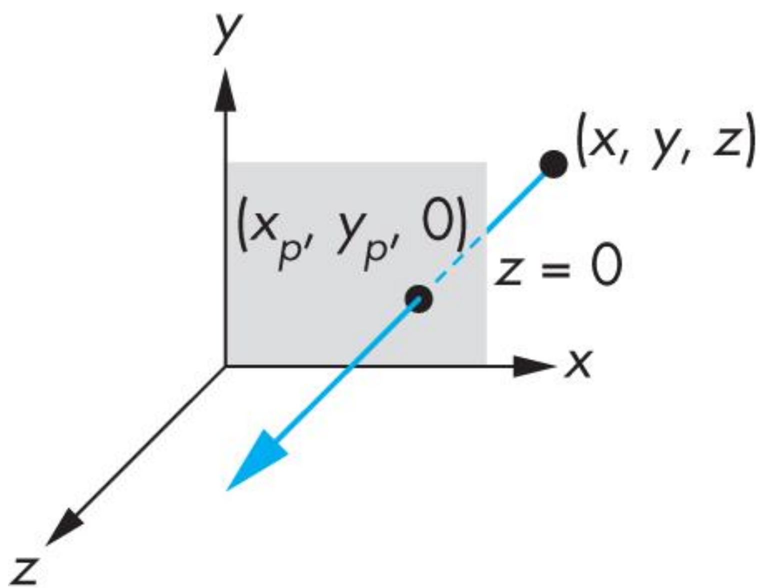
- A special kind of parallel projection: projectors perpendicular to projection plane
- Simple, but not realistic
- Used in blueprints (multiview projections)





Simple Orthographic Projection Matrix

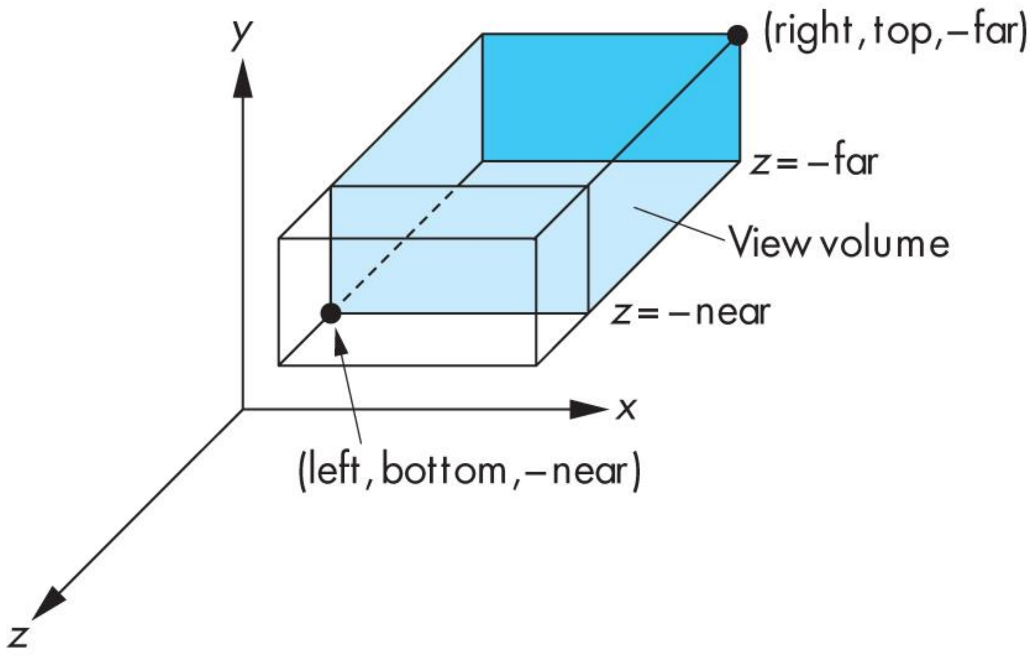
- Project onto $z = 0$
- $x_p = x$, $y_p = y$, $z_p = 0$
- In homogenous coordinates



$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

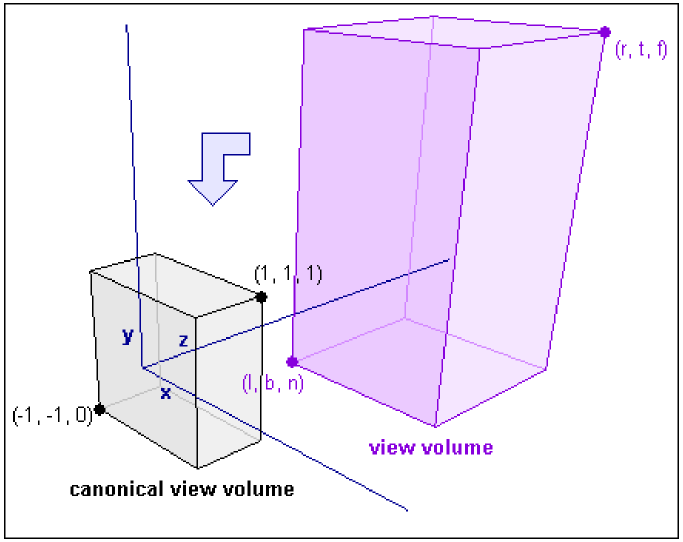
Orthographic Viewing in Old OpenGL

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



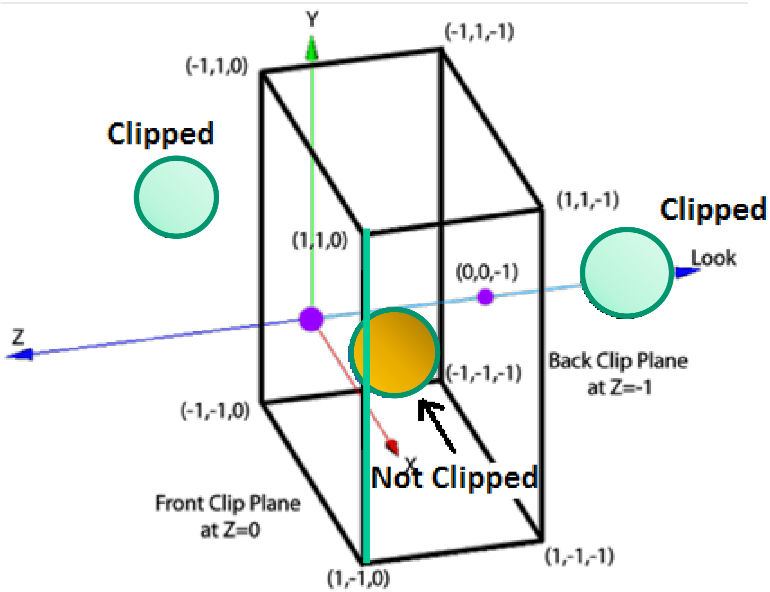
The Normalized view volume

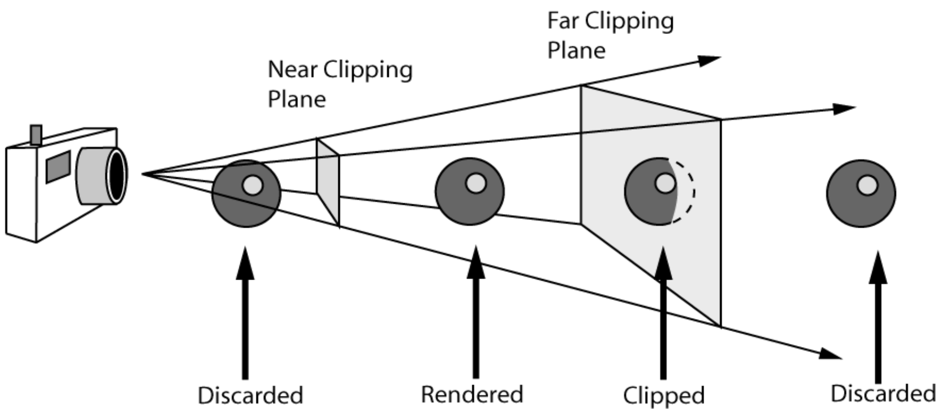
- How exactly do we take contents of an arbitrary view volume and project them to a 2D surface?
- Arbitrary view volume is too complex…
- Reduce it to a simpler problem! The **Normalized view volume!**
- Can also be called the *standard* or *unit* or *canonical* view volume



Clipping against the normalized view volume

- After applying normalizing transformation to all vertices in scene, anything that falls outside the bounds of the planes x = (-1,1), y = (-1, 1) and z = (0, -1), is clipped. Primitives that intersect the view volume must be partially clipped
- Most graphics packages such as OpenGL will do this step for you

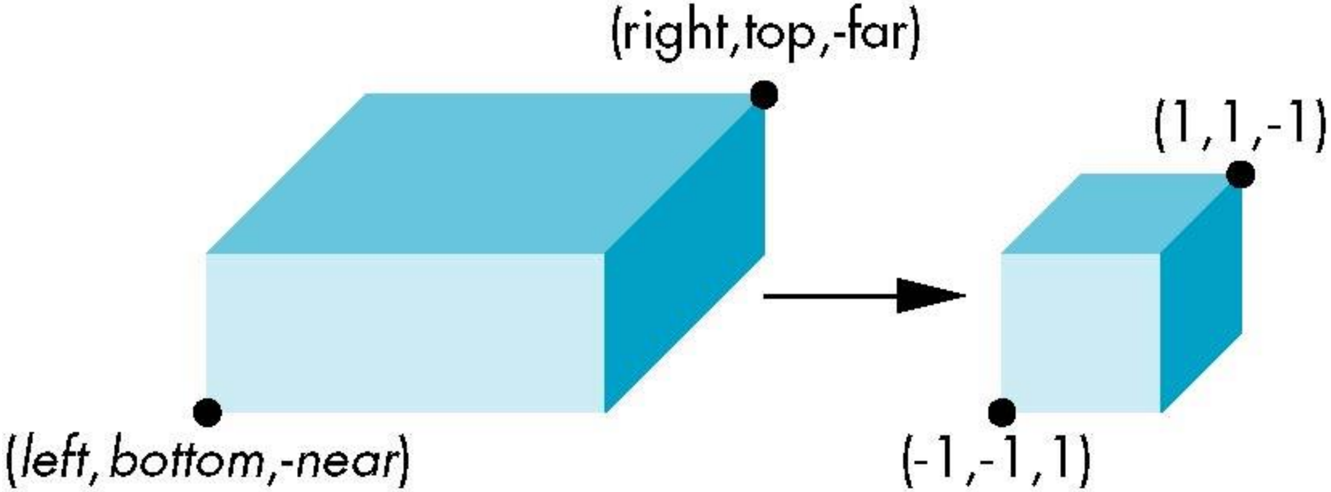




Orthogonal Normalization I

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

normalization ⇒ find transformation to convert specified clipping volume to default



Orthogonal Matrix I

- Two steps
  - Move center to origin

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

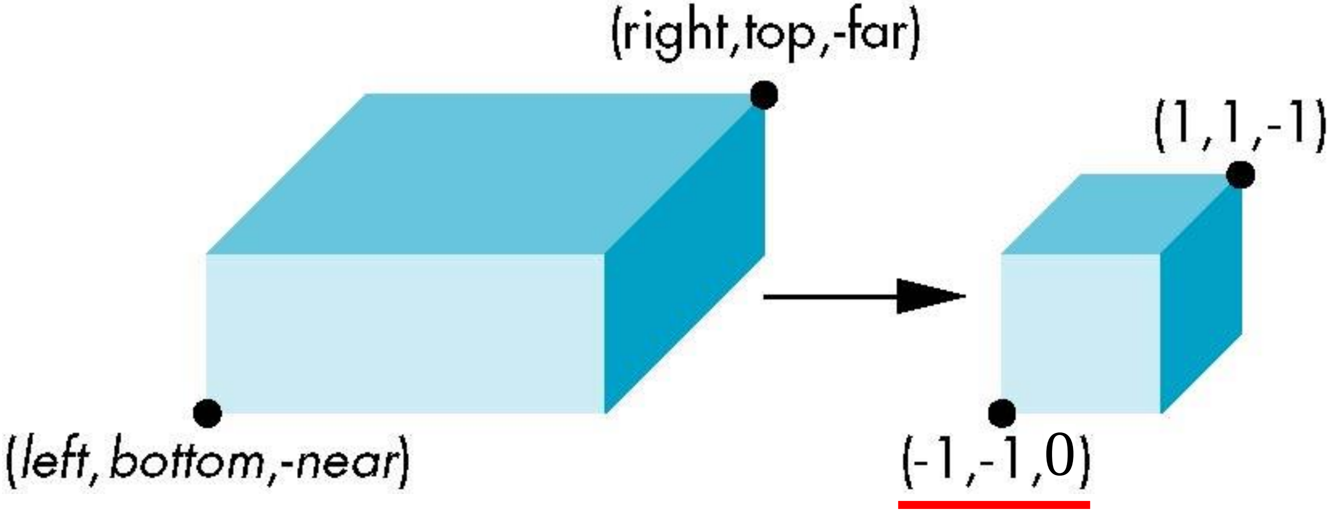
    S(2/(left-right),2/(top-bottom),2/(far-near))

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

Orthogonal Normalization II

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

normalization ⇒ find transformation to convert specified clipping volume to default



Orthogonal Matrix II

- Two steps
  - Move center to origin

    Translate(-(left+right)/2, -(bottom+top)/2, near)
  - Scale to have sides of length 2 for x, y, and length 1 for z

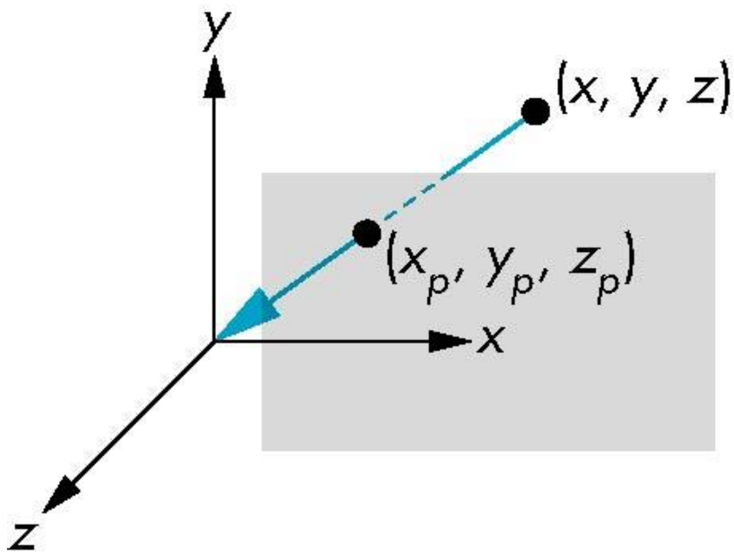
    Scale(2/(left-right),2/(top-bottom),1/(far-near))

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

Perspective Projection

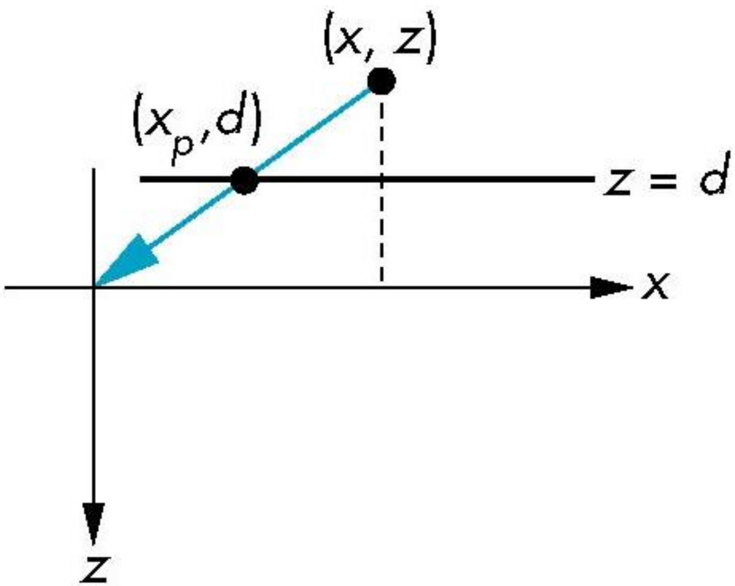
Simple Perspective

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

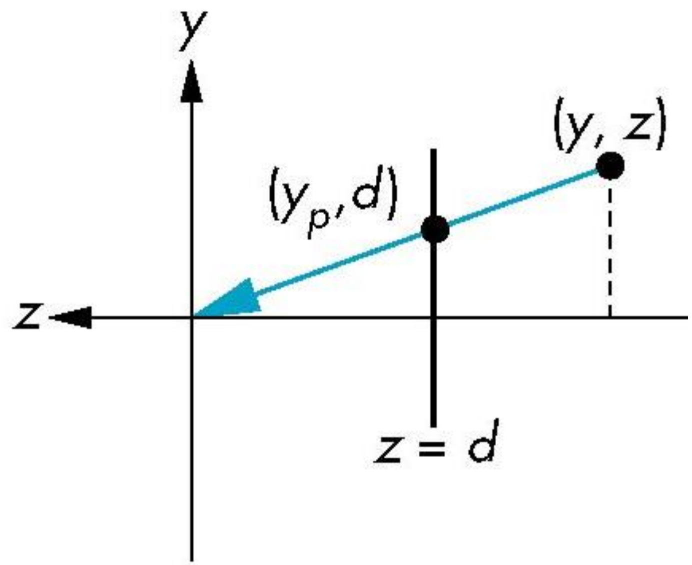


Perspective Equations

Consider top and side views



$$\frac{x}{z/d}$$



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

Homogeneous Coordinate Form

consider $\mathbf{q} = \mathbf{Mp}$ where $\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$

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

Perspective Division

- However $w \neq 1$, so we must divide by w to return from homogeneous coordinates

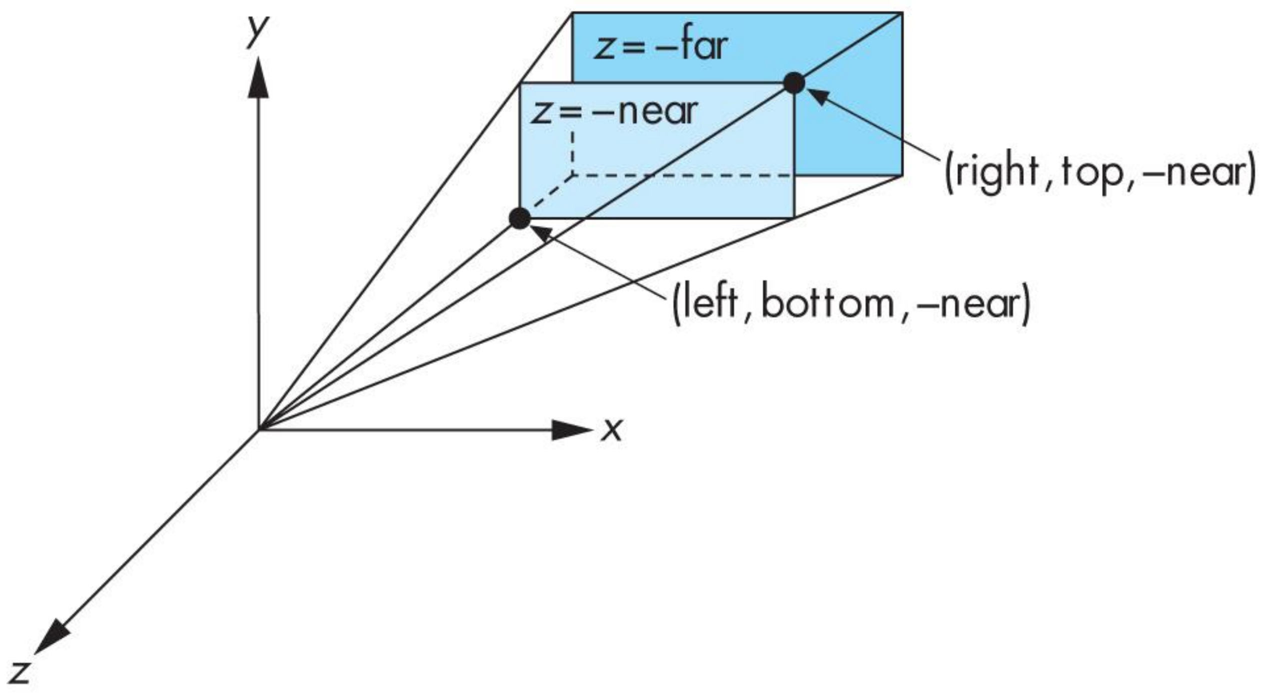
- This *perspective division* yields

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

the desired perspective equations

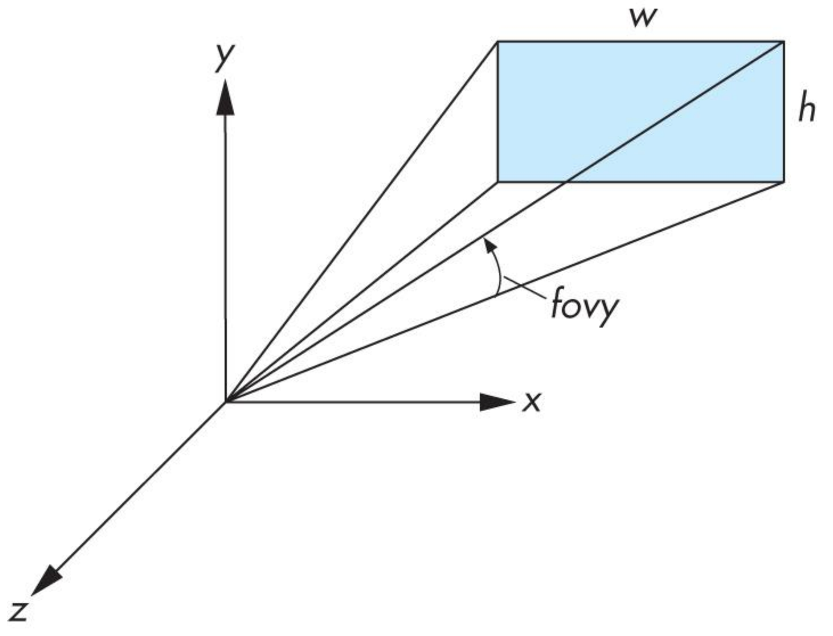
Perspective Viewing in Old OpenGL

- Two interfaces: glFrustum and gluPerspective
- glFrustum(xmin, xmax, ymin, ymax, near, far);



Field of View Interface in Old OpenGL

- gluPerspective(fovy, aspectRatio, near, far);
- aspectRatio = w / h
- fovy specifies field of view as height (y) angle

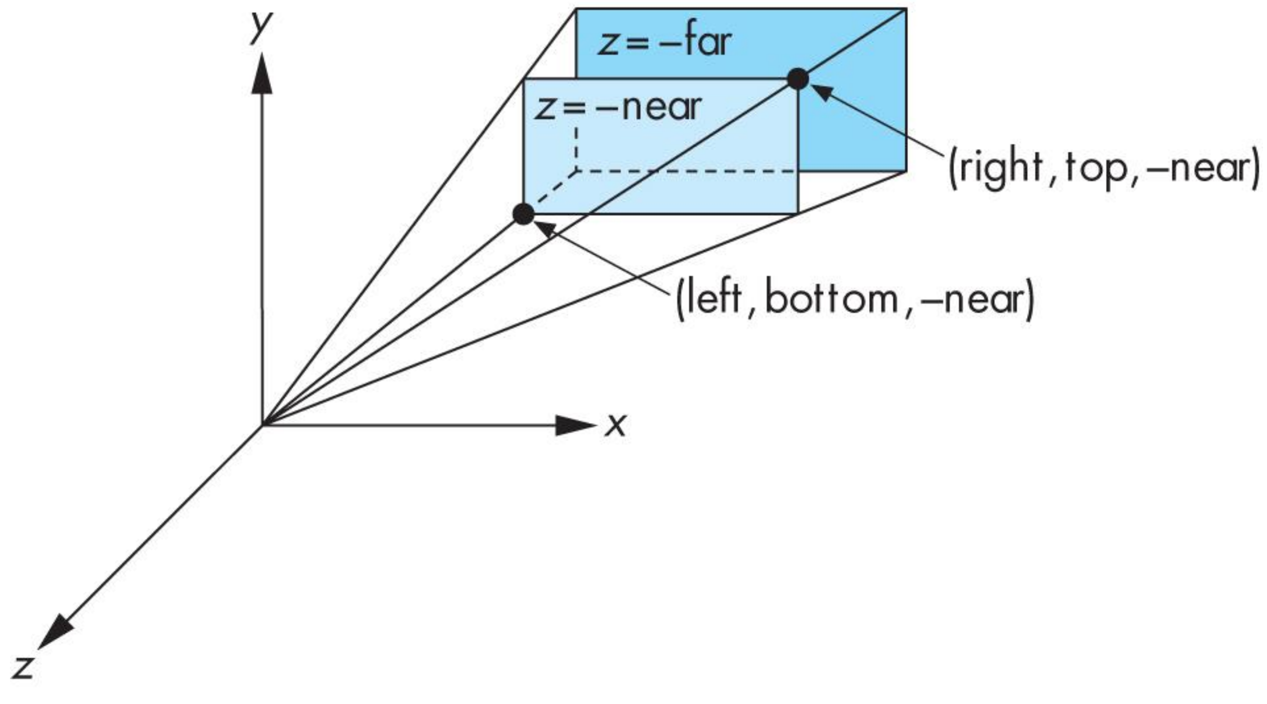


Old OpenGL code

```
void reshape(int x, int y)
{
      glViewport(0, 0, x, y);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluPerspective(60.0, x/float(y), 0.01, 10.0);
}
```

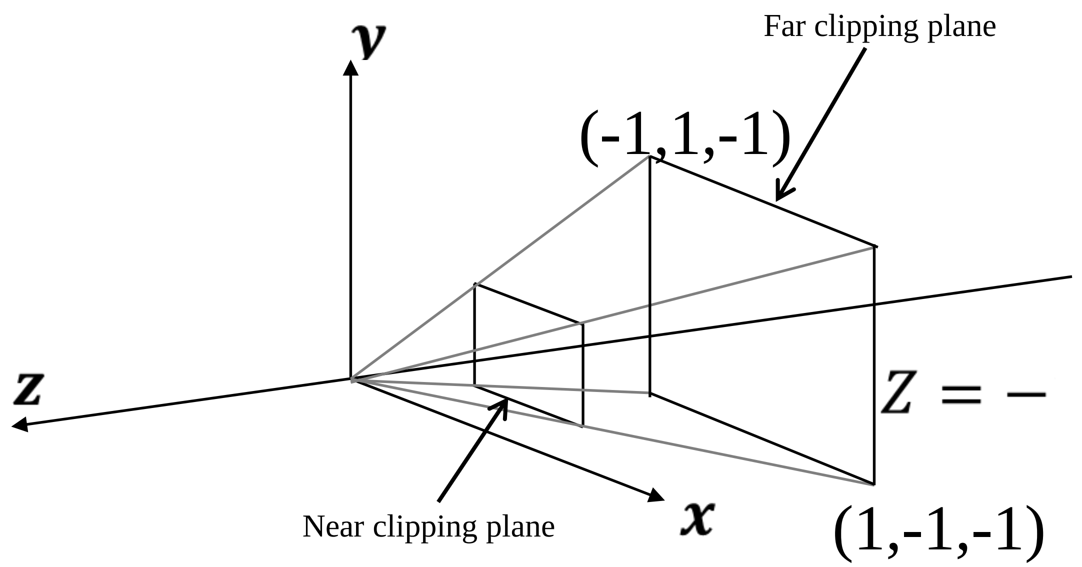
Implementing your own Frustum Function

- `glFrustum(xmin,xmax, ymin,ymax, near,far);`
- `gluPerspective(fovy,aspectRatio, near,far);`

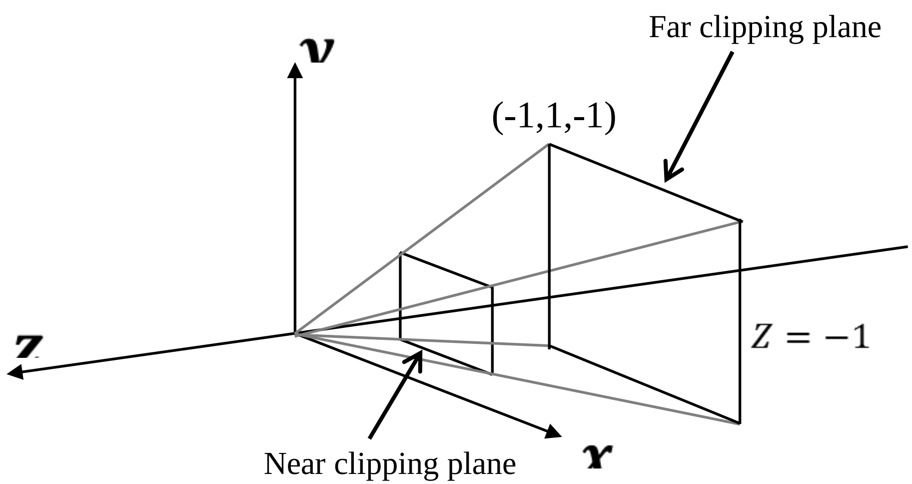


The Perspective View Volume

- Canonical view volume (frustum):



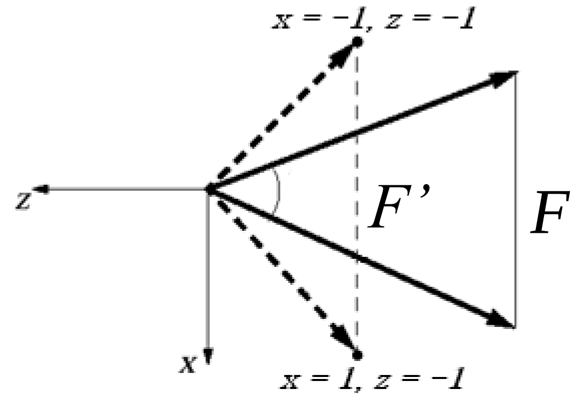
Properties of the canonical view volume



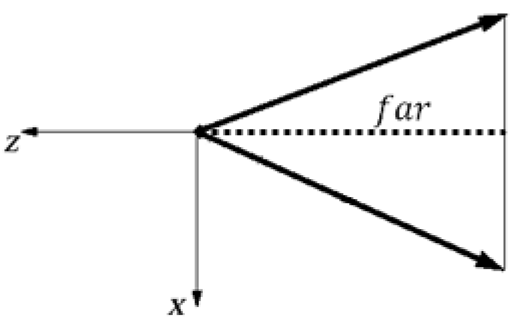
- Sits at origin:
  - Camera position = (0,0,0)
- Looks along negative *z*-axis:
  - Look Vector = (0,0,-1)
- Oriented upright:
  - Up Vector = (0,1,0)
- Near and far clipping planes:
  - Near plane at *z* =$c = -\frac{near}{far}$ (will prove this)
  - Far plane at *z* = -1
- Far clipping plane bounds:
  - (*x*, *y*) from -1 to 1

- Note: *The perspective canonical view volume is just like the parallel one except that the "film"/viewing window is more ambiguous here, so we bound just the far clipping plane for now*

Scaling the perspective view volume (1/4)



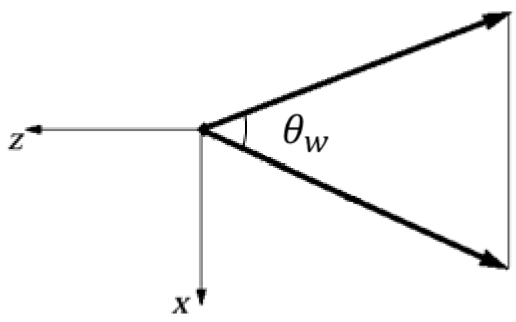
- Top-down view of the perspective view volume:
- Goal: scale the original volume so the solid arrows are transformed to the dotted arrows
  - Equivalently: Scale the original (solid) far plane cross-section F so it lines up with the canonical (dotted) far plane cross-section F'
- First, scale along Z direction
  - Want to scale so far plane lies at $z = -1$
  - Far plane originally lies at $z = -far$
  - Divide by far, since $\frac{-far}{far} = -1$
  - So $Scale_Z = \frac{1}{far}$

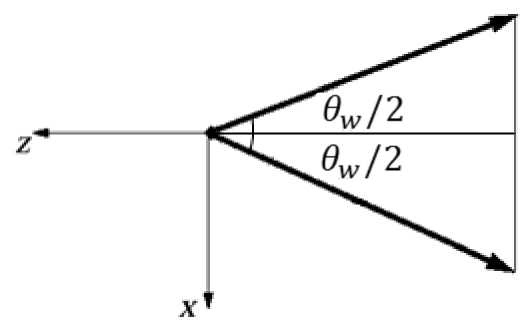


Scaling the perspective view volume (2/4)

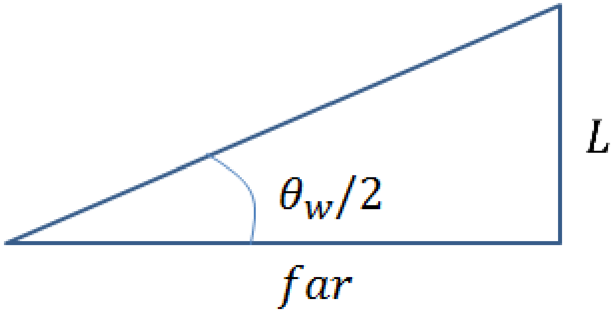
- Next, scale along X direction
  - Use the same trick: divide by size of volume along the X axis

- How long is the side of the volume along X? Find out using trig…
  - Start with the original volume
  - Cut in half along the Z axis





Scaling the perspective view volume (3/4)



- Consider just the top triangle

- Note that L equals the X coordinate of a corner of the perspective view volume's cross-section. Ultimately want to scale by $\frac{1}{L}$ to make $L \rightarrow 1$

- $\frac{L}{far} = \tan(\frac{\theta_w}{2}) \quad \rightarrow \quad L = far \tan\left(\frac{\theta_w}{2}\right)$

- Conclude that $Scale_X = \frac{1}{far \tan(\frac{\theta_w}{2})}$

Scaling the perspective view volume (4/4)

- Finally, scale along Y direction
  - Use the same trig as X direction, but use the height angle θ_h instead of θ_w
  - Result: $Scale_Y = \frac{1}{far \tan(\frac{\theta_h}{2})}$

- The final result is this scale matrix:

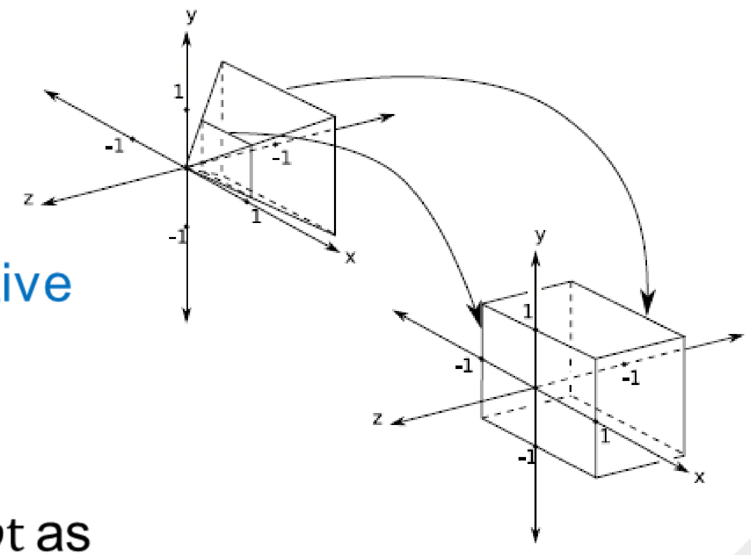
$$\boldsymbol{S_{xyz}} = \begin{bmatrix} \frac{1}{\tan\left(\frac{\theta_w}{2}\right) far} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{\theta_h}{2}\right) far} & 0 & 0 \\ 0 & 0 & \frac{1}{far} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Perspective and Projection

- Now we have our canonical perspective view volume

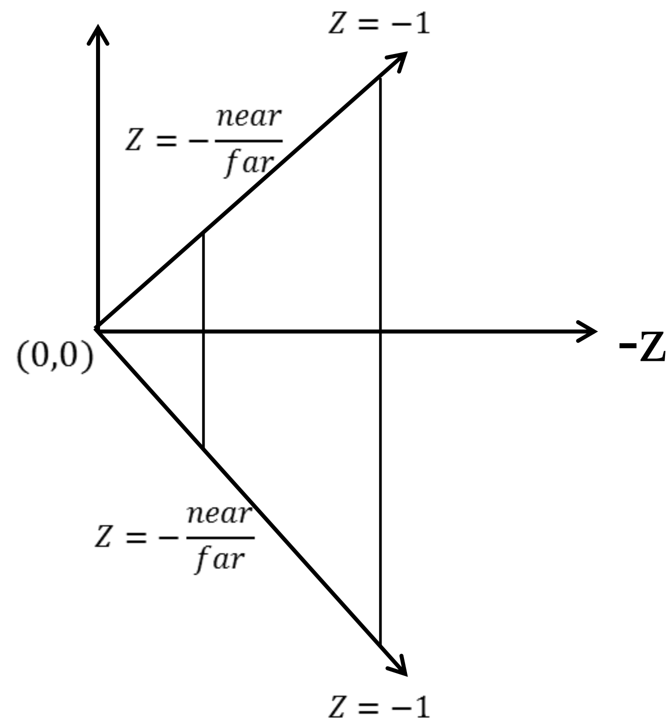
- The final step of our normalizing transformation, transforming the perspective view volume into a parallel one!

- Think of this perspective transformation $\boldsymbol{pt}$ as the ***unhinging transformation***, represented by matrix $\boldsymbol{M_{pt}}$



Unhinging View Volume to Become a Parallel View Volume (1/4)

- Near clipping plane at $c = -\frac{near}{far}$ should transform to $z = 0$



Unhinging View Volume to Become a Parallel View Volume(2/4)

- The derivation of our unhinging transformation matrix is complex.
- Instead, we will give you the matrix and show that it works by example
- Our unhinging transformation matrix, $\boldsymbol{M_{pt}}$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{1}{c+1} & \dfrac{-c}{c+1} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Unhinging View Volume to Become a Parallel View Volume(3/4)

- Our perspective transformation does the following:
  - Sends all points on the z = -1 far clipping plane to themselves
    - We'll check top-left (-1, 1, -1, 1) and bottom-right (1, -1, -1, 1) corners
  - Sends all points on the z = c near clipping plane onto the $z = 0$ plane
    - Note that the corners of the cross section of the near clipping plane in the frustum are $(-c, c, c, 1)$, $(c, -c, c, 1), (c, c, c, 1)$ and $(-c, -c, c, 1)$
    - We'll check to see that $(-c, c, c, 1)$ gets sent to (-1, 1, 0, 1) and that $(c, -c, c, 1)$ gets sent to (1, -1, 0, 1)
  - Let's try $c = -\frac{1}{2}$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{c+1} & \frac{-c}{c+1} \\ 0 & 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Unhinging View Volume to Become a Parallel View Volume(4/4)

$$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&2&1\\0&0&-1&0\end{bmatrix}\begin{bmatrix}-1\\1\\-1\\1\end{bmatrix}\longrightarrow\begin{bmatrix}-1\\1\\-1\\1\end{bmatrix}$$

$$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&2&1\\0&0&-1&0\end{bmatrix}\begin{bmatrix}1\\-1\\-1\\1\end{bmatrix}\longrightarrow\begin{bmatrix}1\\-1\\-1\\1\end{bmatrix}$$

Don't forget to homogenize!

$$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&2&1\\0&0&-1&0\end{bmatrix}\begin{bmatrix}-1/2\\1/2\\-1/2\\1\end{bmatrix}\longrightarrow\begin{bmatrix}-1/2\\1/2\\0\\1/2\end{bmatrix}\longrightarrow\begin{bmatrix}-1\\1\\0\\1\end{bmatrix}$$

$$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&2&1\\0&0&-1&0\end{bmatrix}\begin{bmatrix}1/2\\-1/2\\-1/2\\1\end{bmatrix}\longrightarrow\begin{bmatrix}1/2\\-1/2\\0\\1/2\end{bmatrix}\longrightarrow\begin{bmatrix}1\\-1\\0\\1\end{bmatrix}$$

The normalizing transformation (perspective)

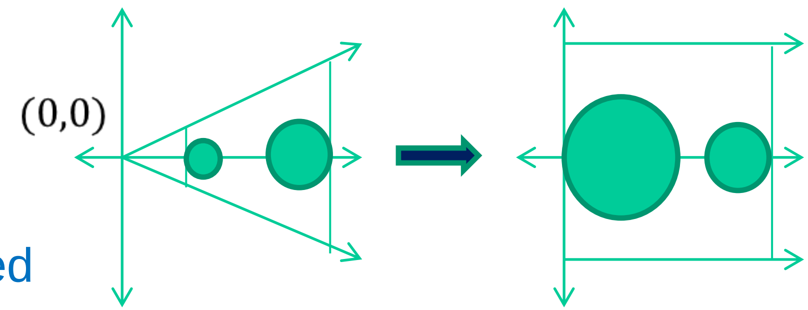
- $\boldsymbol{N_{perspective} = M_{pt} S_{xyz}}$

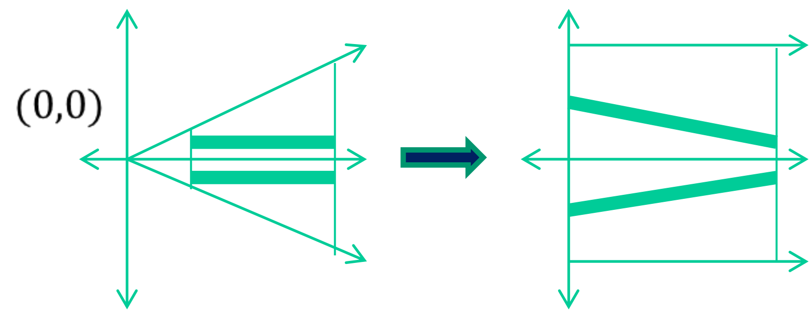
$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{c+1} & \frac{-c}{c+1} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\tan\left(\frac{\theta_w}{2}\right) far} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{\theta_h}{2}\right) far} & 0 & 0 \\ 0 & 0 & 1/far & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Remember to homogenize your points after you apply this transformation

Why it works (1/2)

- The key is in the **unhinging step**
- We can take an intuitive approach to see this
  - The closer the object is to the near clipping plane, the more it is enlarged during the unhinging step
  - Thus, closer objects are larger and farther away objects are smaller, as is to be expected
- Another way to see it is to use the parallel lines
  - Draw parallel lines in a perspective volume
  - When we unhinge the volume, the lines fan out at the near clipping
  - The result is converging lines, the railroad track





Why it works (2/2)

- Yet another way to demonstrate how this works is to use occlusion (when elements in the scene are blocked by other elements)
- Looking at the top view of the frustum, we see a square
- Draw a line from your eye point to the left corner of the square, we can see that points behind this corner are obscured
- Now unhinge the perspective and draw a line again to the left corner, we can see that all points obscured before are still obscured and all points that were visible before are still visible