3.1 Viewing and Projection

Hao Li

http://cs420.hao-li.com

Recall: Affine Transformations

- Given a point $[x \ y \ z]^\top$
- form homogeneous coordinates $[x \ y \ z \ 1]^\top$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- The transformed point is $[x' \ y' \ z']^\top$

Transformation Matrices in OpenGL

- Transformation matrices in OpenGL are vectors of 16 values (column-major matrices)
- In glLoadMatrixf(GLfloat *m);

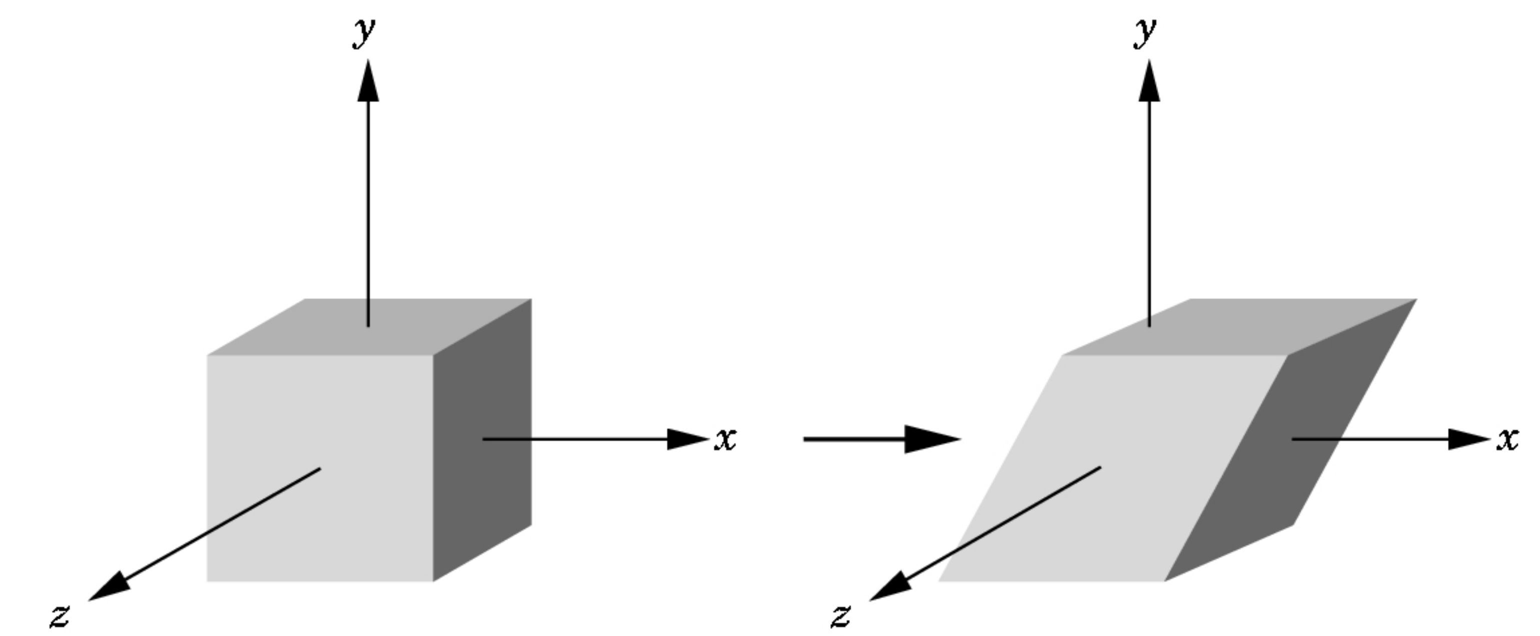
$$\mathbf{m}^\top = [m_1, m_2, \ldots, m_{16}]^\top \text{ represents}$$

$$\begin{bmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{bmatrix}$$

- Some books transpose all matrices!

Shear Transformations

- x-shear scales x proportional to y
- Leaves y and z values fixed

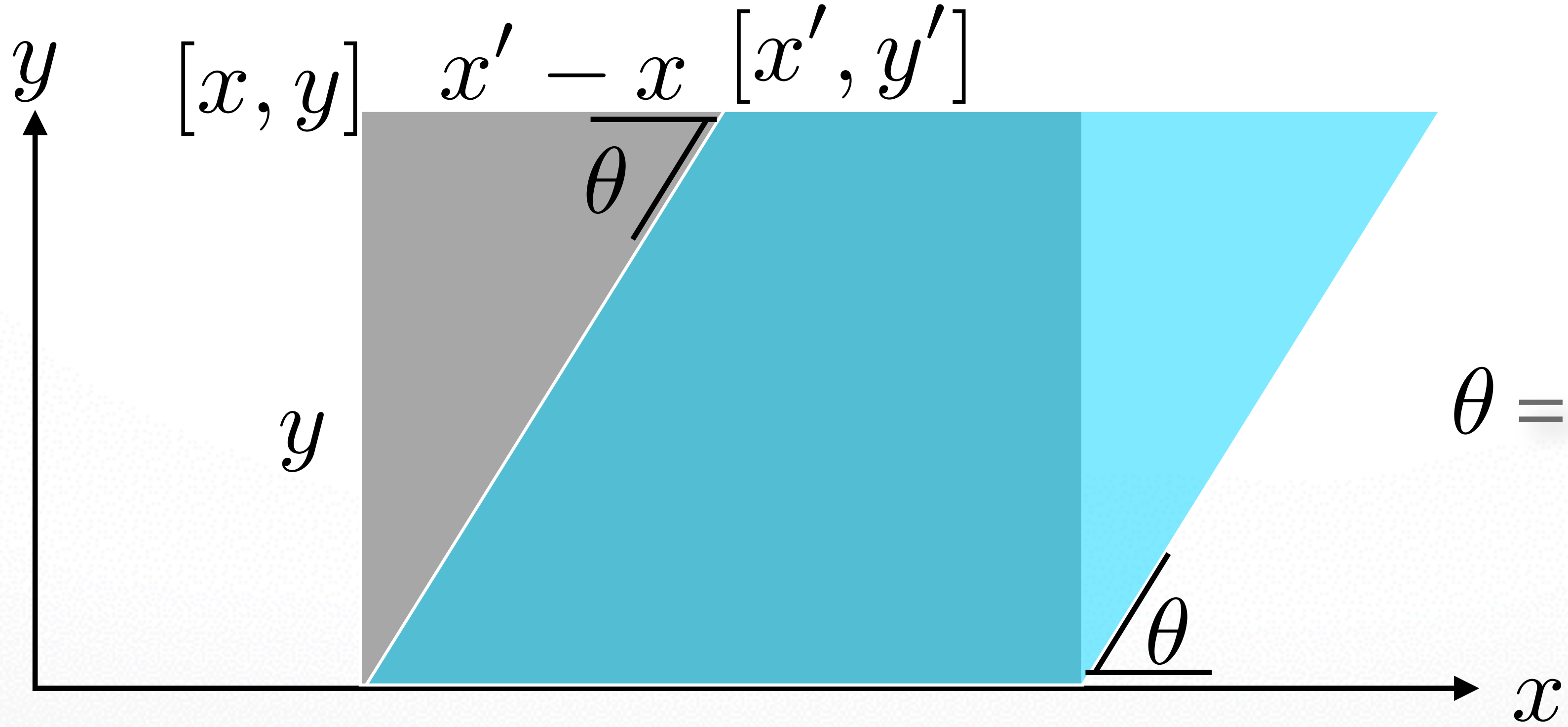


Specification via Shear Angle

$$\cot(\theta) = (x' - x)/y$$

$$x' = x + y\cot(\theta)$$
$$y' = y$$
$$z' = z$$

$$H_x(\theta) = \begin{bmatrix} 1 & \cot(\theta) & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



θ = shear angle

Specification via Ratios

- For example, shear in both x and z direction
- Leave y fixed
- Slope α for x-shear, γ for z-shear
- Solve

$$H_{x,z}(\alpha,\gamma)\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x+\alpha y \\ y \\ z+\gamma y \\ 1 \end{bmatrix}$$

- Yields

$$H_{x,z}(\alpha,\gamma) = \begin{bmatrix} 1 & \alpha & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \gamma & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Composing Transformations

- Let $\mathbf{p} = \mathbf{A}\mathbf{q}$, and $\mathbf{q} = \mathbf{B}\mathbf{s}$
- Then $\mathbf{p} = (\mathbf{A}\mathbf{B})\mathbf{s}$

$$
\begin{tikzcd}
\mathbf{s} \arrow[r, bend right, "\mathbf{B}"'] \arrow[rr, bend right=60, "\mathbf{AB}"'] & \mathbf{q} \arrow[r, bend right, "\mathbf{A}"'] & \mathbf{p}
\end{tikzcd}
$$

matrix multiplication

Composing Transformations

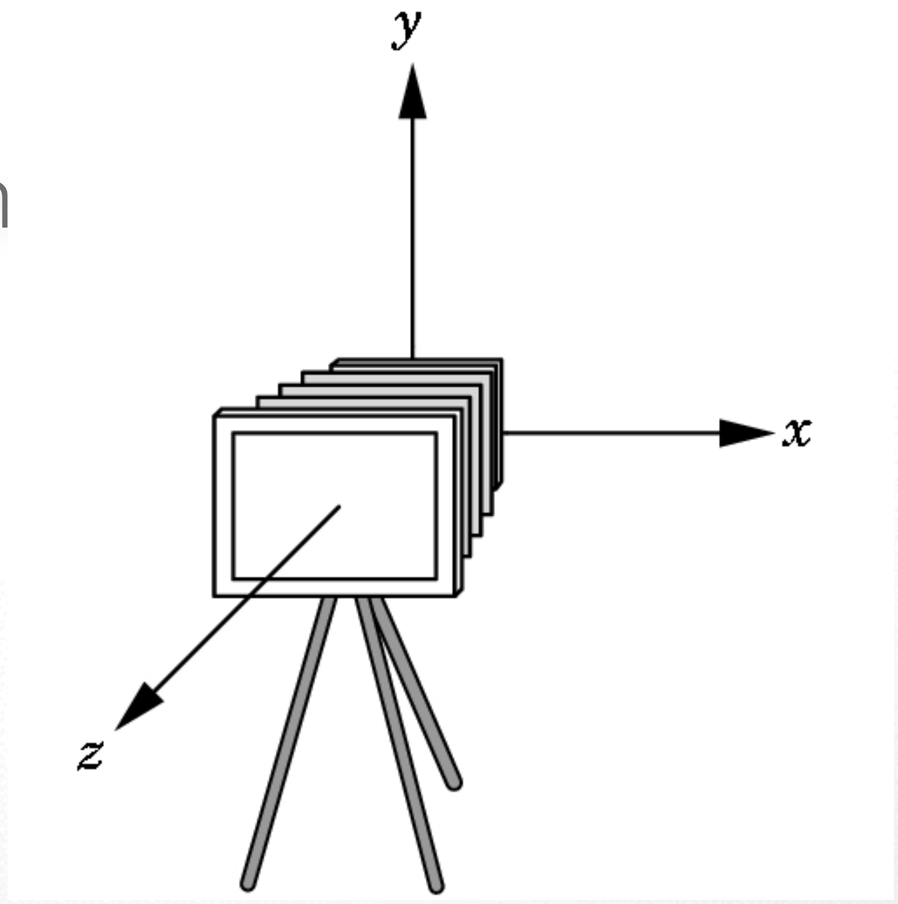
- **Every affine transformation is a composition of rotations, scalings, and translations**
- So, how do we compose these to form an x-shear?
- Exercise!

Outline

- Shear Transformation
- Camera Positioning
- Simple Parallel Projections
- Simple Perspective Projections

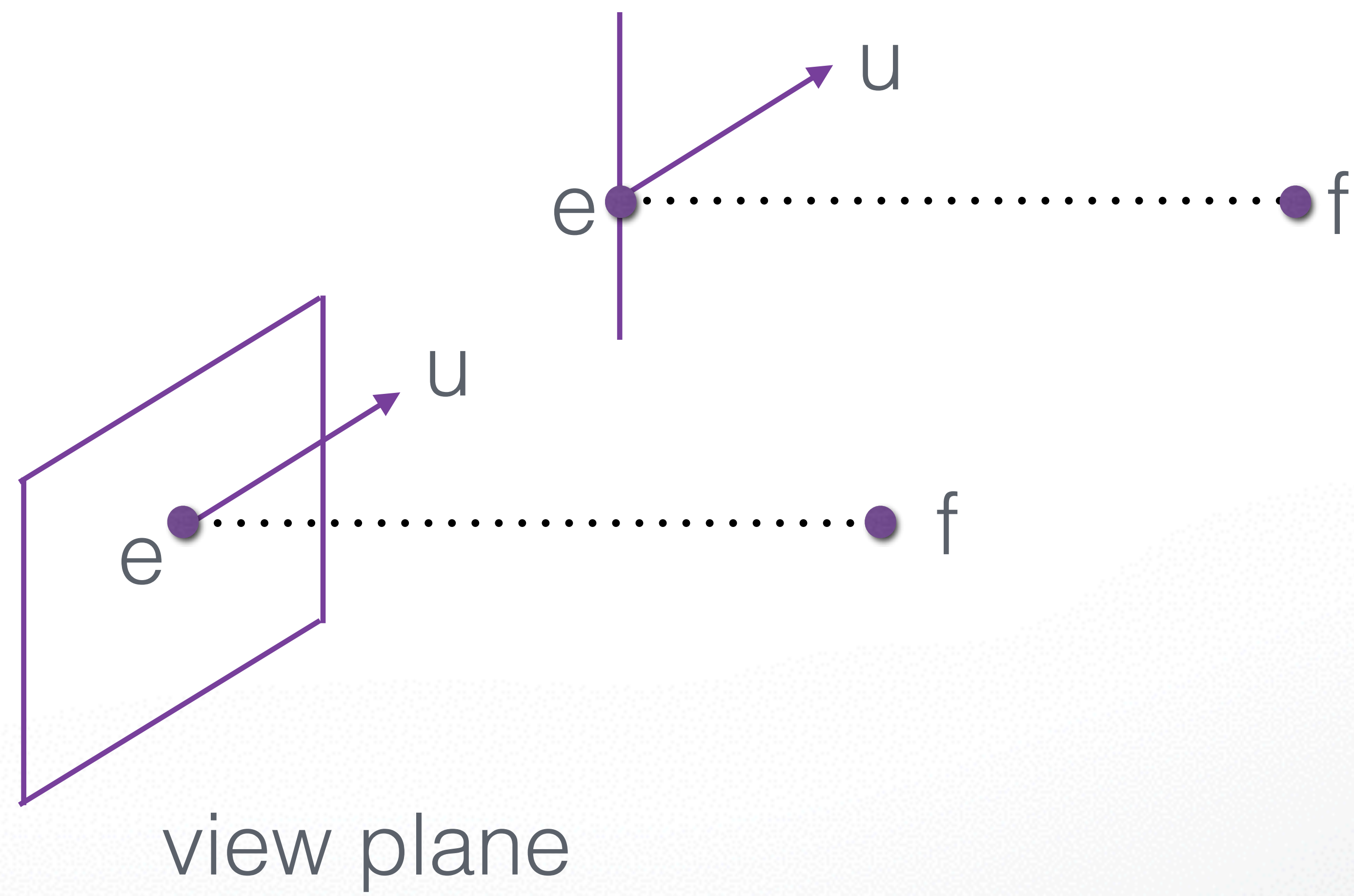
Transform Camera = Transform Scene

- Camera position is identified with a frame
- Either move and rotate the objects
- Or move and rotate the camera
- Initially, camera at origin, pointing in negative z-direction



The Look-At Function

- Convenient way to position camera
- gluLookAt(ex, ey, ez, fx, fy, fz, ux, uy, uz);
- e = eye point
- f = focus point
- u = up vector



OpenGL code

```
void display()
{
  glClear (GL_COLOR_BUFFER_BIT |
        GL_DEPTH_BUFFER_BIT);
  glMatrixMode (GL_MODELVIEW);
  glLoadIdentity();

  gluLookAt (ex, ey, ez,  fx, fy, fz,  ux, uy, uz);

  glTranslatef(x, y, z);
  ...
  renderBunny();

  glutSwapBuffers();
}
```

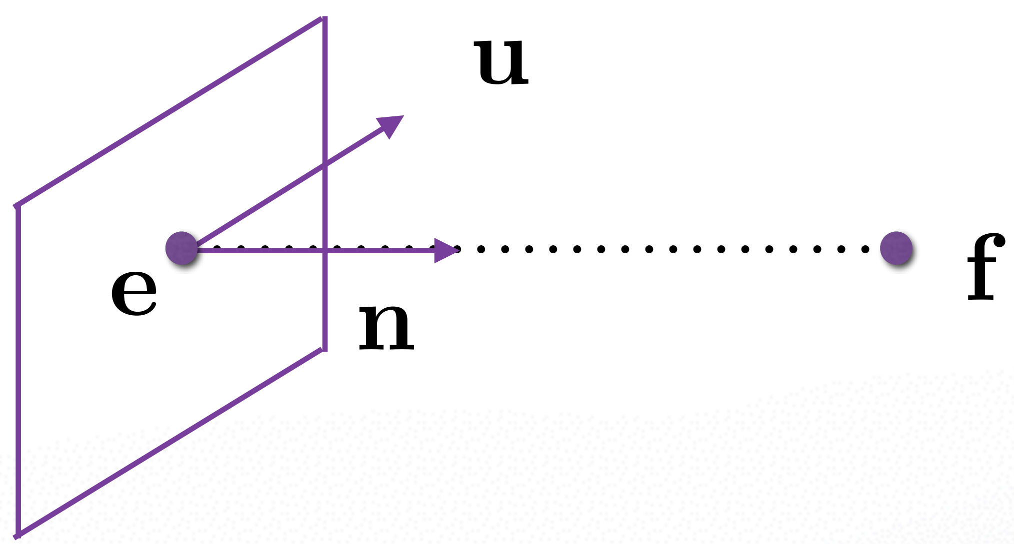
Implementing the Look-At Function

1. Transform world frame to camera frame
   - Compose a rotation $\mathbf{R}$ with translation $\mathbf{T}$
   - $\mathbf{W} = \mathbf{T}\mathbf{R}$

2. Invert $\mathbf{W}$ to obtain viewing transformation $\mathbf{V}$
   - $\mathbf{V} = \mathbf{W}^{-1} = (\mathbf{T}\mathbf{R})^{-1} = \mathbf{R}^{-1}\mathbf{T}^{-1}$
   - Derive $\mathbf{R}$, then $\mathbf{T}$, then $\mathbf{R}^{-1}\mathbf{T}^{-1}$

World Frame to Camera Frame I

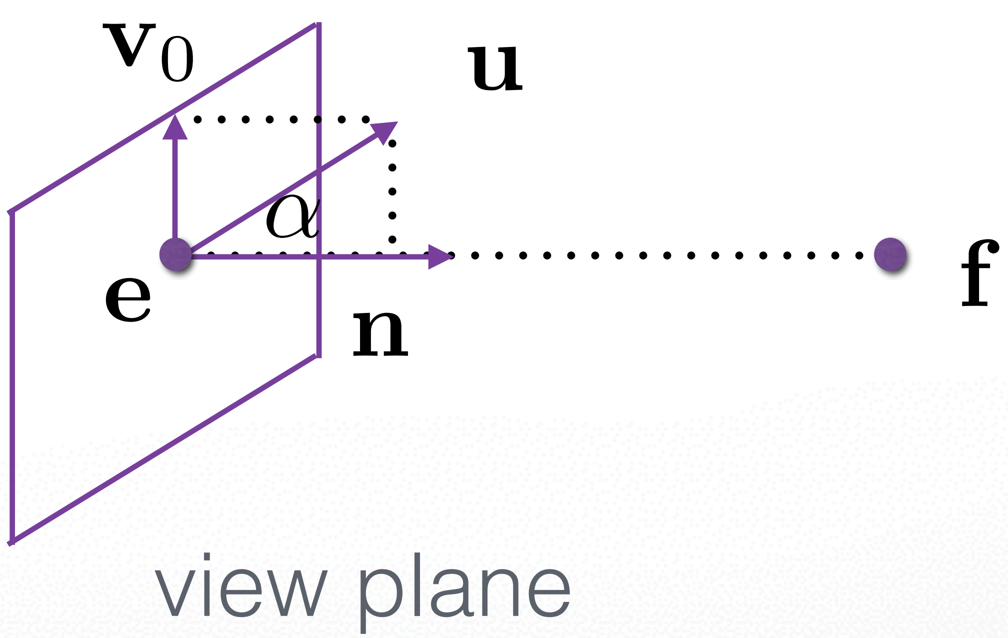
- Camera points in negative z direction
- $\mathbf{n} = (\mathbf{f} - \mathbf{e})/\|\mathbf{f} - \mathbf{e}\|$ is unit normal to view plane
- Therefore, $\mathbf{R}$ maps $[0\,0\,-1]^\top$ to $[n_x\,n_y\,n_z]^\top$



view plane

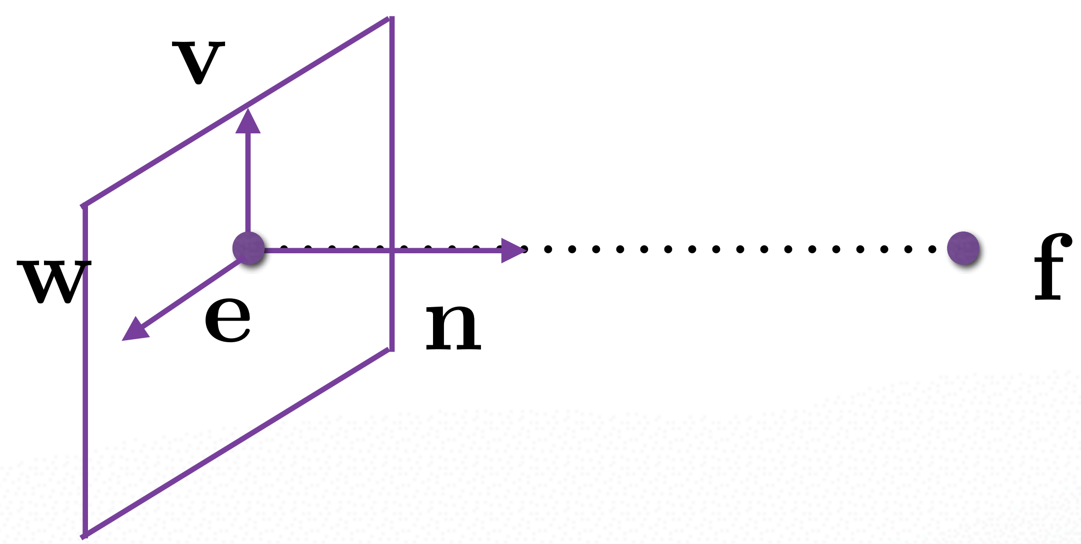
World Frame to Camera Frame II

- $\mathbf{R}$ maps $[0\ 1\ 0]^\top$ to projection of u onto view plane
- This projection $\mathbf{v}$ equals:
  - $\alpha = \mathbf{u}^\top \mathbf{n} / \|\mathbf{n}\| = \mathbf{u}^\top \mathbf{n}$
  - $\mathbf{v}_0 = \mathbf{u} - \alpha \mathbf{n}$
  - $\mathbf{v} = \mathbf{v}_0 / \|\mathbf{v}_0\|$



World Frame to Camera Frame III

- Set $\mathbf{w}$ to be orthogonal to $\mathbf{n}$ and $\mathbf{v}$,
- $\mathbf{w} = \mathbf{n} \times \mathbf{v}$,
- $[\mathbf{w}\ \mathbf{v}\ \ -\mathbf{n}]^\top$ is right-handed



view plane

Summary of Rotation

- gluLookAt(e_x, e_y, e_z, f_x, f_y, f_z, u_x, u_y, u_z);
- $\mathbf{n} = (\mathbf{f} - \mathbf{e}) / \|\mathbf{f} - \mathbf{e}\|$,
- $\mathbf{v} = (\mathbf{u} - (\mathbf{u}^\top \mathbf{n})\mathbf{n}) / \|\mathbf{u} - (\mathbf{u}^\top \mathbf{n})\mathbf{n}\|$,
- $\mathbf{w} = \mathbf{n} \times \mathbf{v}$.
- Rotation must map:
  - $[1\ 0\ 0]$ to $\mathbf{w}$
  - $[0\ 1\ 0]$ to $\mathbf{v}$
  - $[0\ 0\ -1]$ to $\mathbf{n}$

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

World Frame to Camera Frame IV

- Translation of origin to $\mathbf{e}^\top = [e_x \; e_y \; e_z \; 1]^\top$

$$T = \begin{bmatrix} 1 & 0 & 0 & e_x \\ 0 & 1 & 0 & e_y \\ 0 & 0 & 1 & e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Camera Frame to Rendering Frame

- $\mathbf{V} = \mathbf{W}^{-1} = (\mathbf{TR})^{-1} = \mathbf{R}^{-1}\mathbf{T}^{-1}$,
- $\mathbf{R}$ is rotation, so $\mathbf{R}^{-1} = \mathbf{R}^{\top}$

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

- $\mathbf{T}$ is translation, so $\mathbf{T}^{-1}$ negates displacement

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Putting it Together

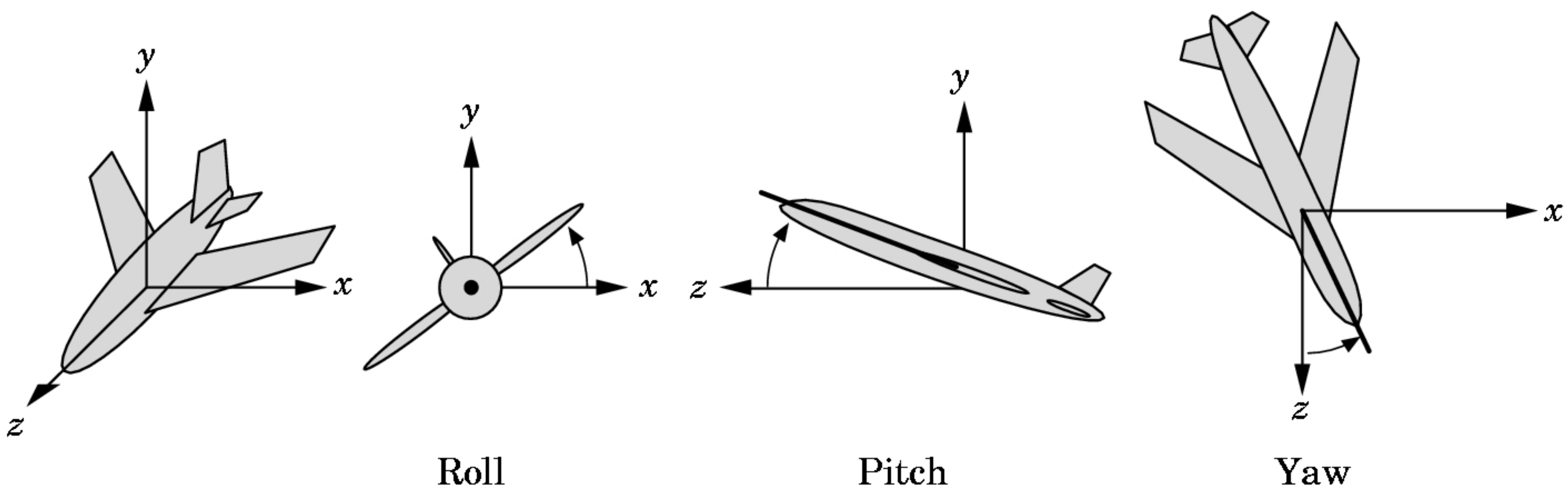
- Calculate $\mathbf{V} = \mathbf{R}^{-1}\mathbf{T}^{-1}$

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

- This is different from book [Angel, Ch. 5.3.2]
- There, $\mathbf{u}, \mathbf{v}, \mathbf{n}$ are right-handed (here: $\mathbf{u}, \mathbf{v}, -\mathbf{n}$)

Other Viewing Functions

- Roll (about z), pitch (about x), yaw (about y)



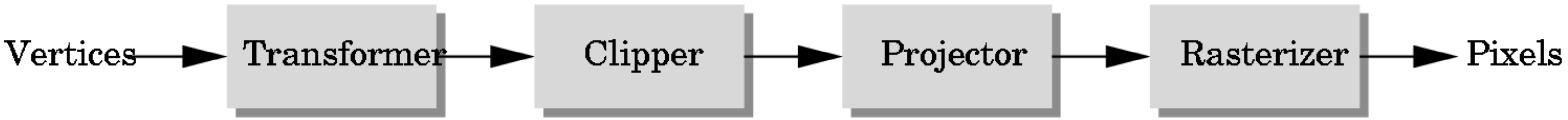
- Assignment 2 poses a related problem

Outline

- Shear Transformation
- Camera Positioning
- Simple Parallel Projections
- Simple Perspective Projections

Projection Matrices

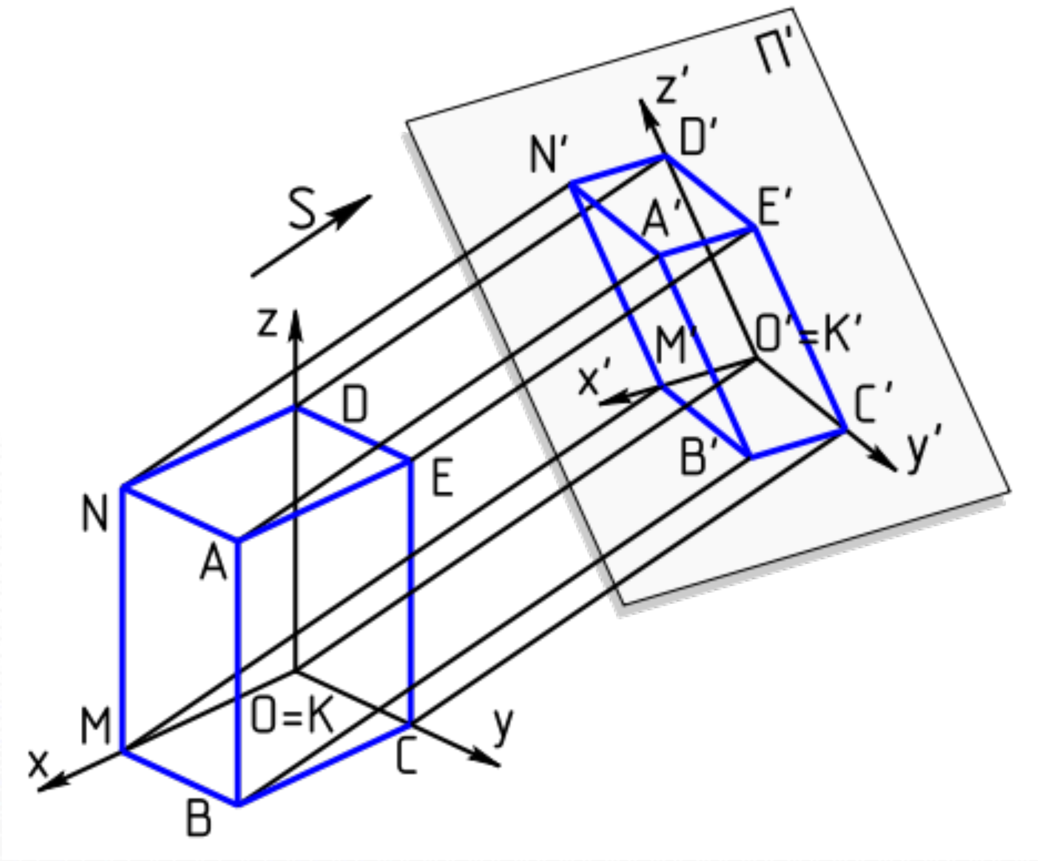
- Recall geometric pipeline



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

Parallel Projection

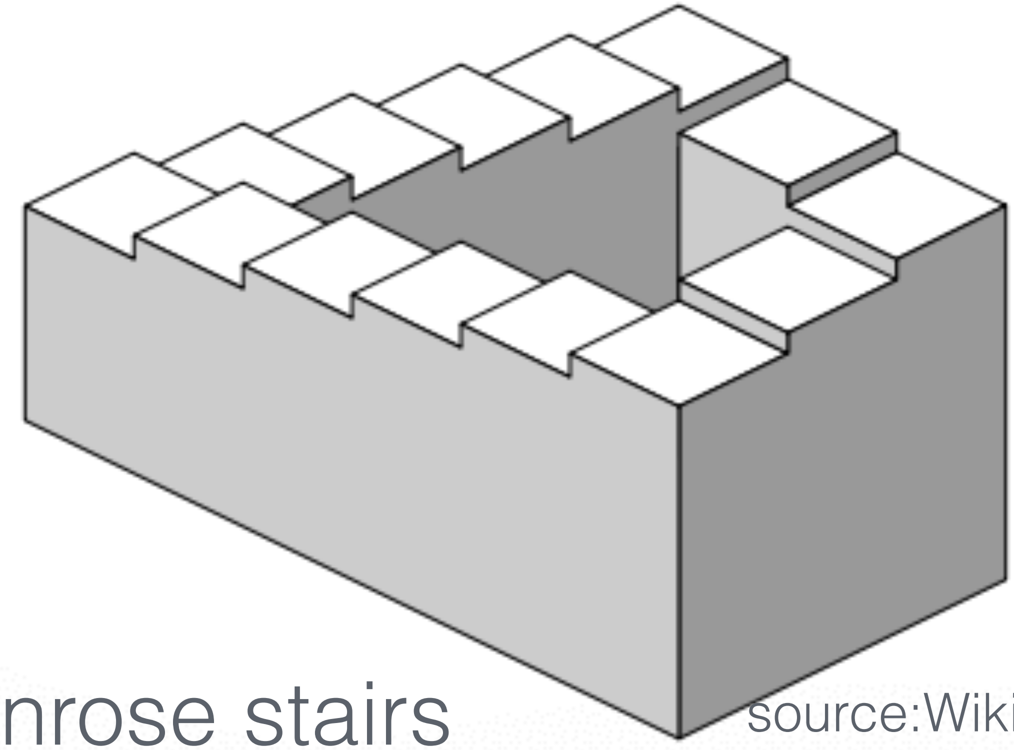
- Project 3D object to 2D via parallel lines
- The lines are not necessarily orthogonal to projection plane



source:Wikipedia

Parallel Projection

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

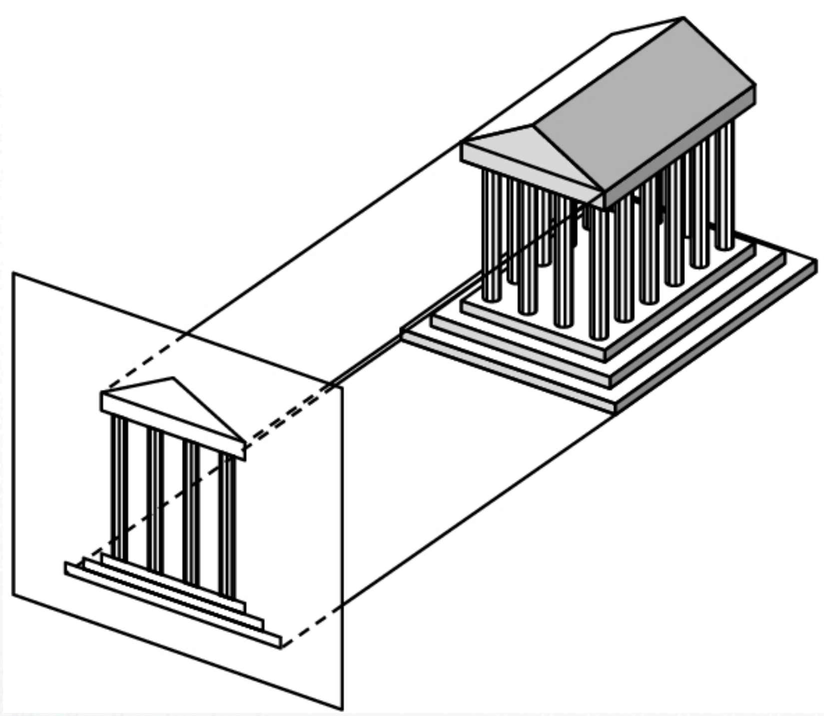


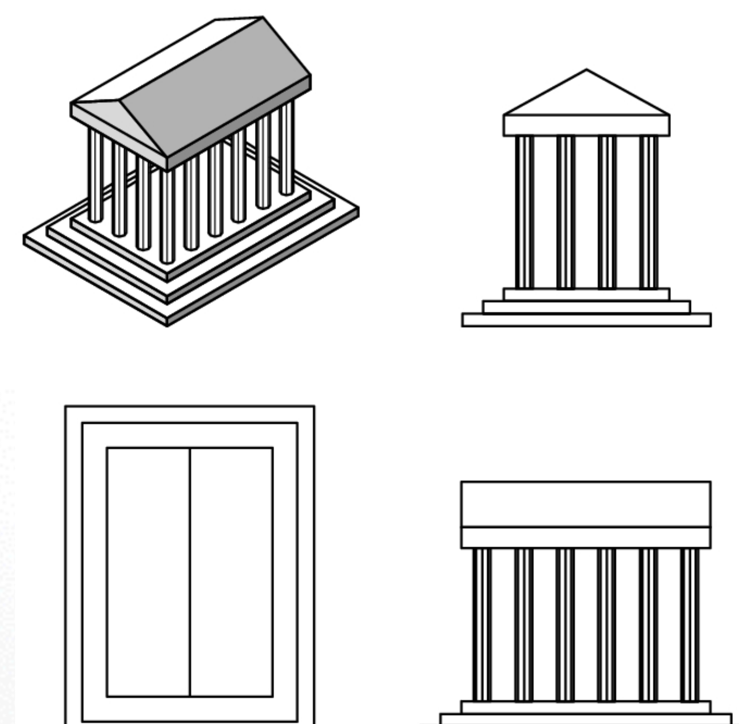
Penrose stairs

source:Wikipedia

Orthographic Projection

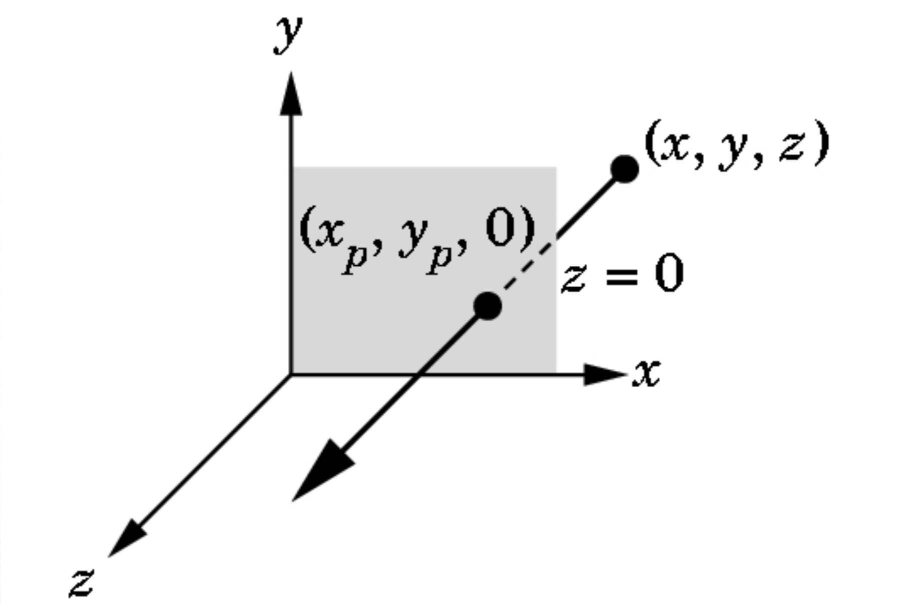
- A special kind of parallel projection:
  projectors perpendicular to projection plane
- Simple, but not realistic
- Used in blueprints (multiview projections)





Orthographic Projection Matrix

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



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

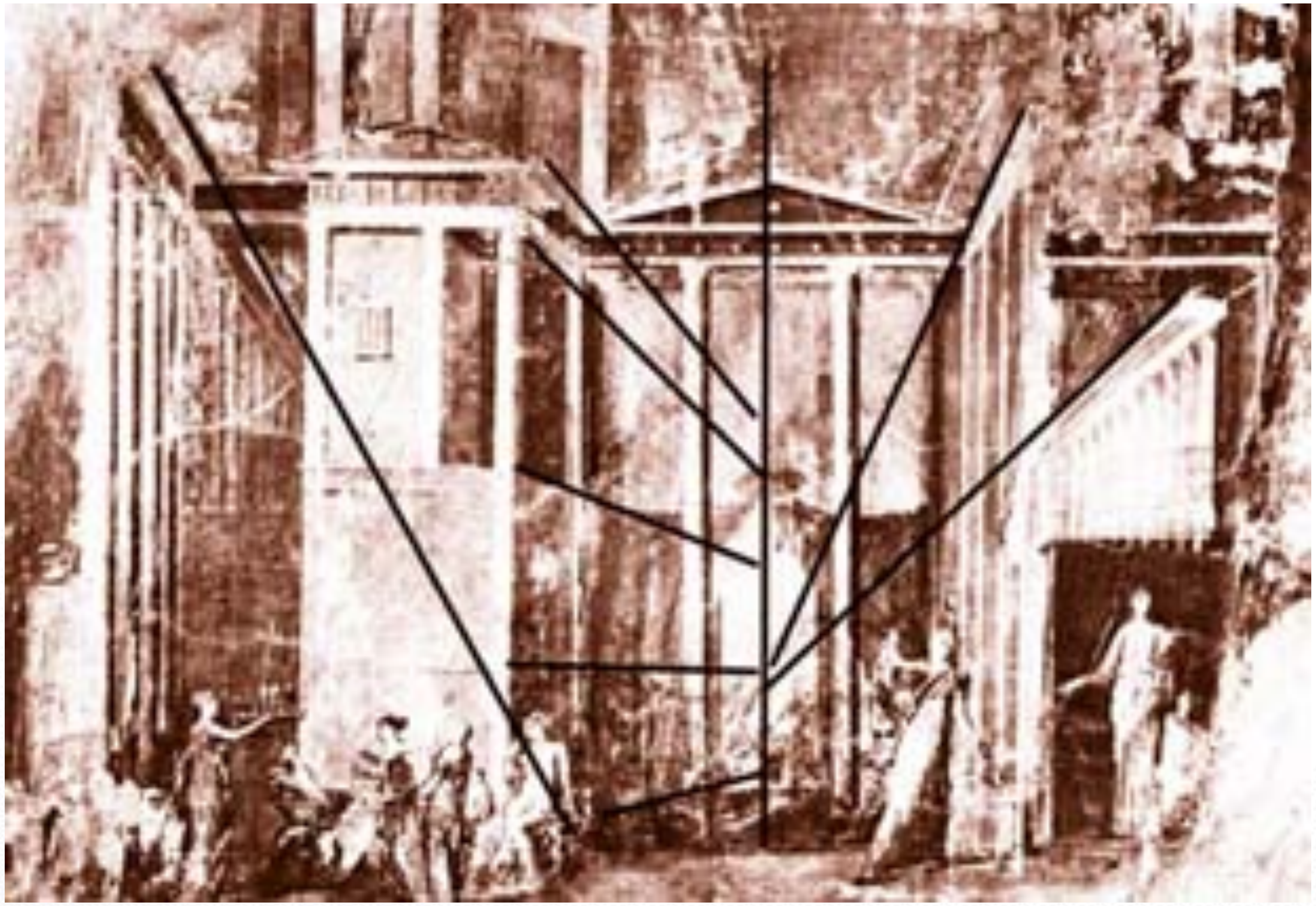
Perspective

- Perspective characterized by foreshortening
- More distant objects appear smaller
- Parallel lines appear to converge
- Rudimentary perspective in cave drawings:

Lascaux, France
source: Wikipedia

Discovery of Perspective

- Foundation in geometry (Euclid)



Mural from Pompeii, Italy

Middle Ages

- Art in the service of religion
- Perspective abandoned or forgotten

Ottonian manuscript,
ca. 1000

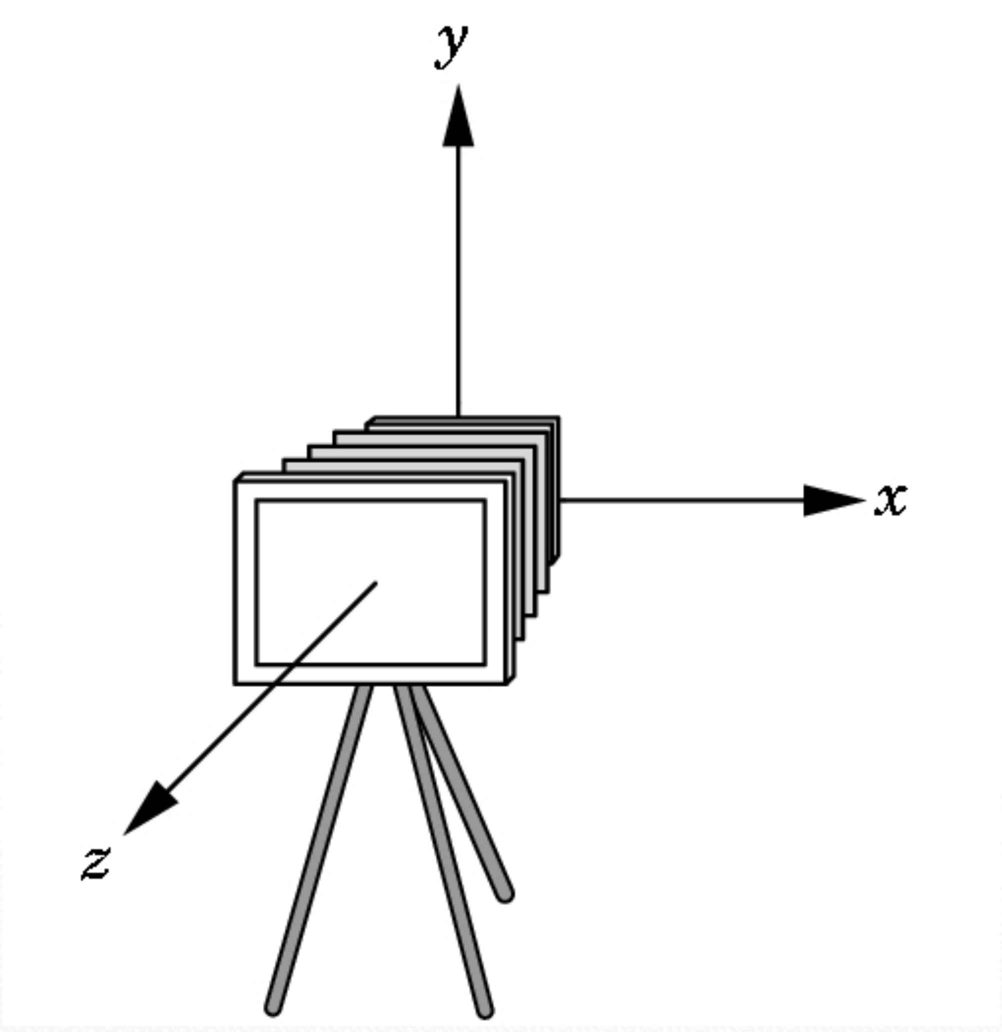
Renaissance

- Rediscovery, systematic study of perspective

Filippo Brunelleschi
Florence, 1415

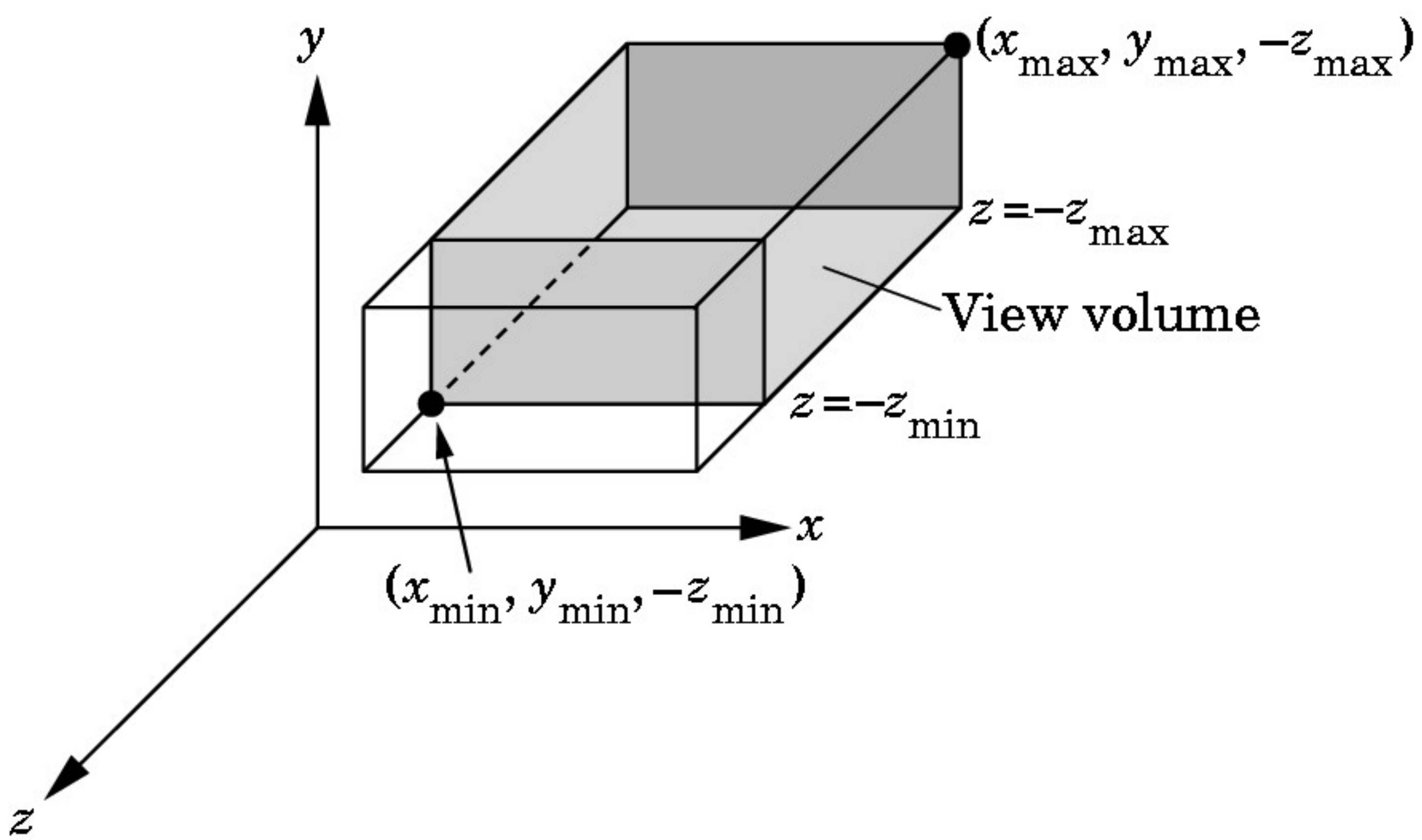
Projection (Viewing) in OpenGL

- Remember: camera is pointing in the negative z direction



Orthographic Viewing in OpenGL

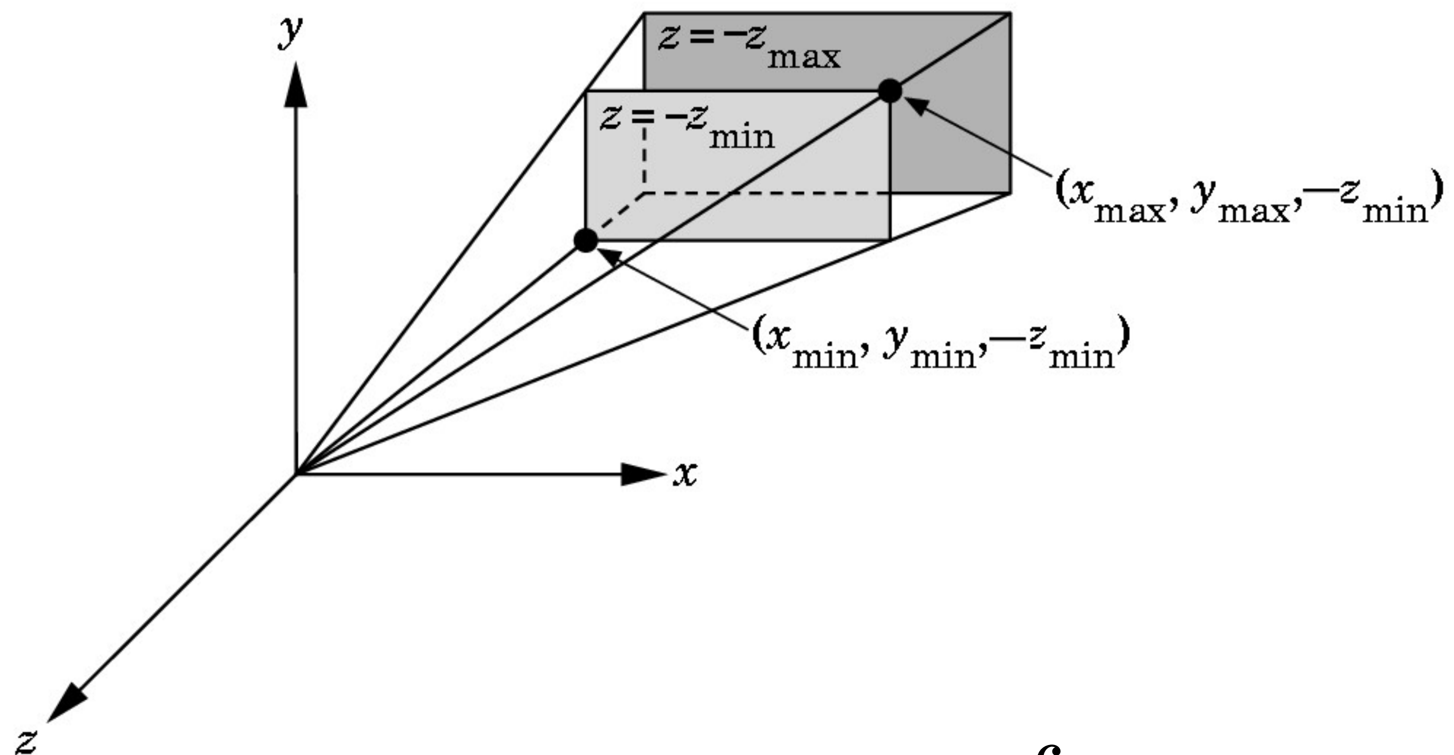
- glOrtho(xmin, xmax, ymin, ymax, near, far)



$$z_{\min} = \text{near}, z_{\max} = \text{far}$$

Perspective Viewing in OpenGL

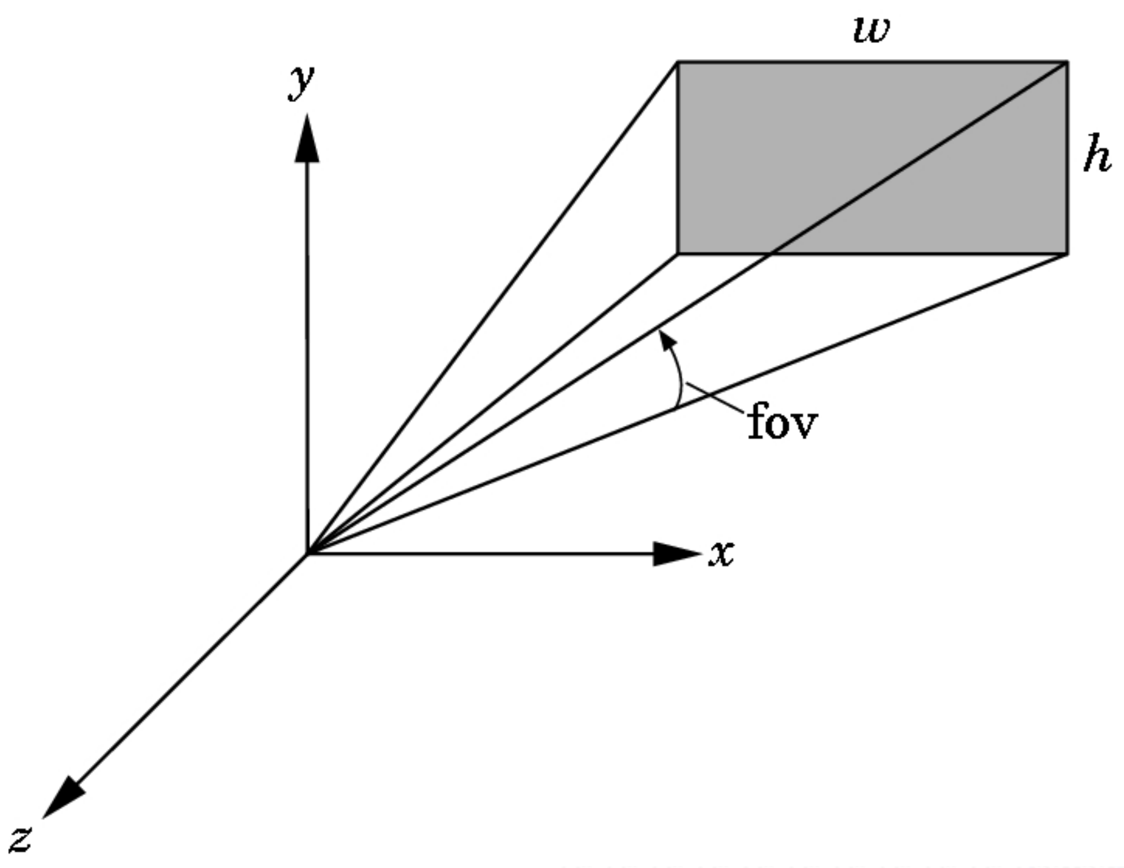
- Two interfaces: glFrustum and gluPerspective
- glFrustum(xmin, xmax, ymin, ymax, near, far);



$$z_{\min} = \text{near}, z_{\max} = \text{far}$$

Field of View Interface

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



OpenGL code

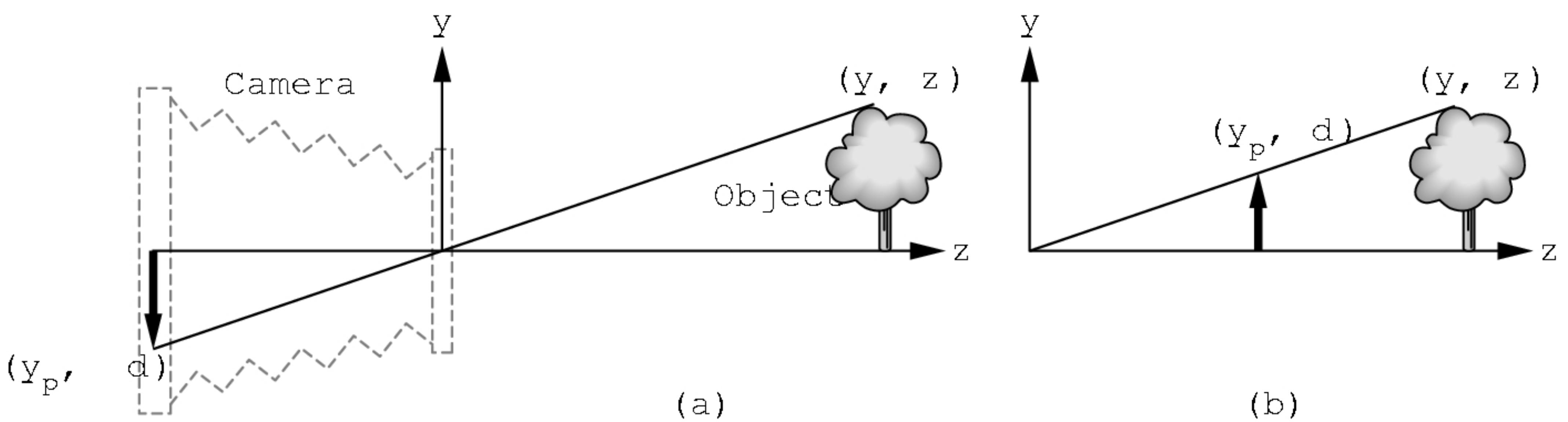
```
void reshape(int x, int y)
{
   glViewport(0, 0, x, y);

   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();

   gluPerspective(60.0, 1.0 * x / y, 0.01, 10.0);

   glMatrixMode(GL_MODELVIEW);
}
```

Perspective Viewing Mathematically



- d = focal length
- $y/z = y_p/d$ so $y_p = y/(z/d) = yd/z$
- Note that y_p is non-linear in the depth z!

Exploiting the 4th Dimension

Perspective projection is not affine:

$$M \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \\ 1 \end{bmatrix}$$

has no solution for M

Idea: exploit homogeneous coordinates

$$p = w \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

for arbitrary $w \neq 0$

Perspective Projection Matrix

- Use multiple of point

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

- Solve

$$M\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ \frac{z}{d} \end{bmatrix}$$ with $$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{bmatrix}$$

Projection Algorithm

- **Input**: 3D point $[x\ y\ z]^\top$ to project
- Form $[x\ y\ z\ 1]^\top$
- Multiply M with $[x\ y\ z\ 1]^\top$; obtaining $[X\ Y\ Z\ W]^\top$
- Perform perspective division:
  X/W , Y/W , Z/W
- **Output**: $[X/W, Y/W, Z/W]^\top$
- (last coordinate will be d)

Perspective Division

- Normalize $[X\ Y\ Z\ W]^\top$ to $[X/W,\ Y/W,\ Z/W,\ 1]^\top$
- Perform perspective division after projection

- Projection in OpenGL is more complex (includes clipping) http://cs420.hao-li.com

Thanks!