Computer Graphics

Projection

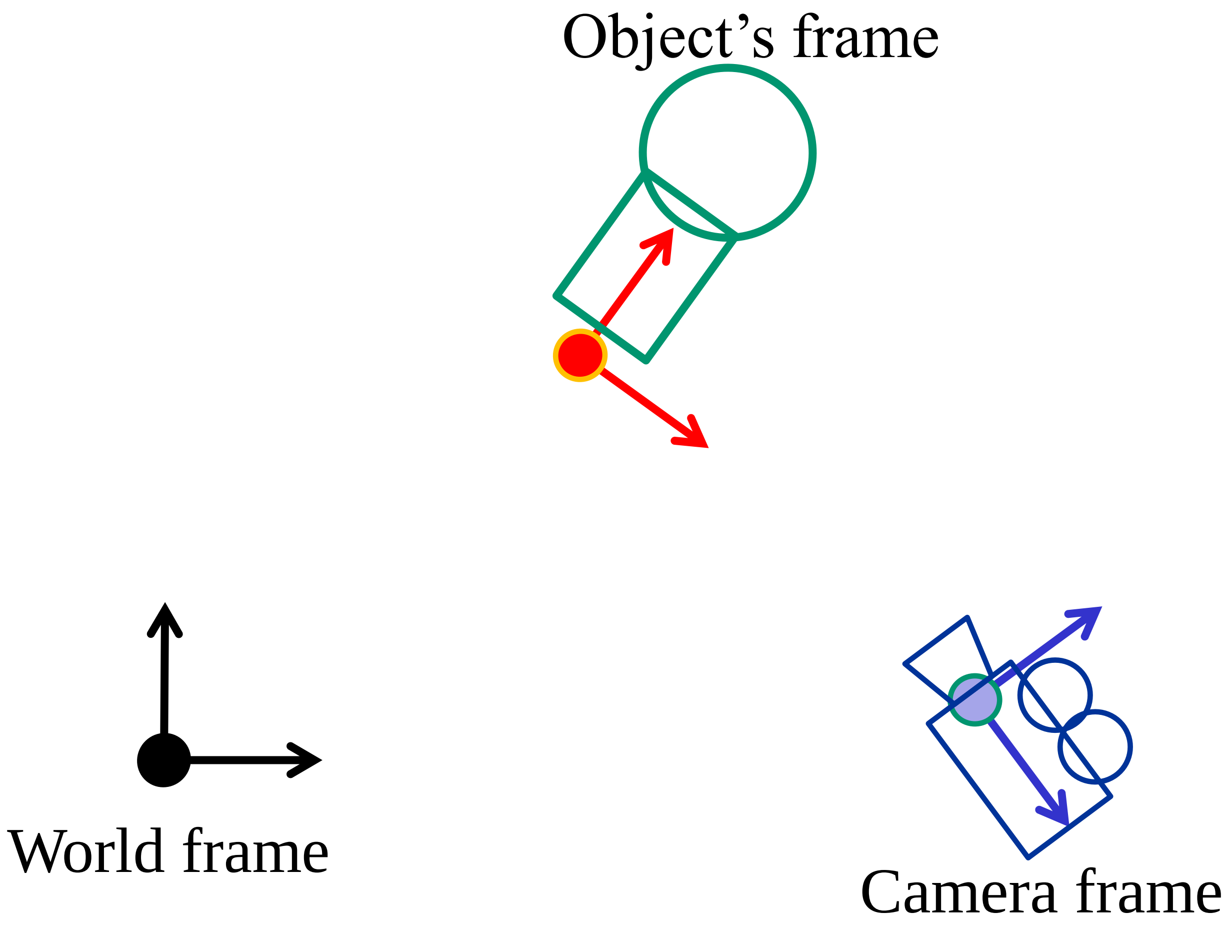
Konstantin Tretyakov
kt@ut.ee

Sep 25, 2013

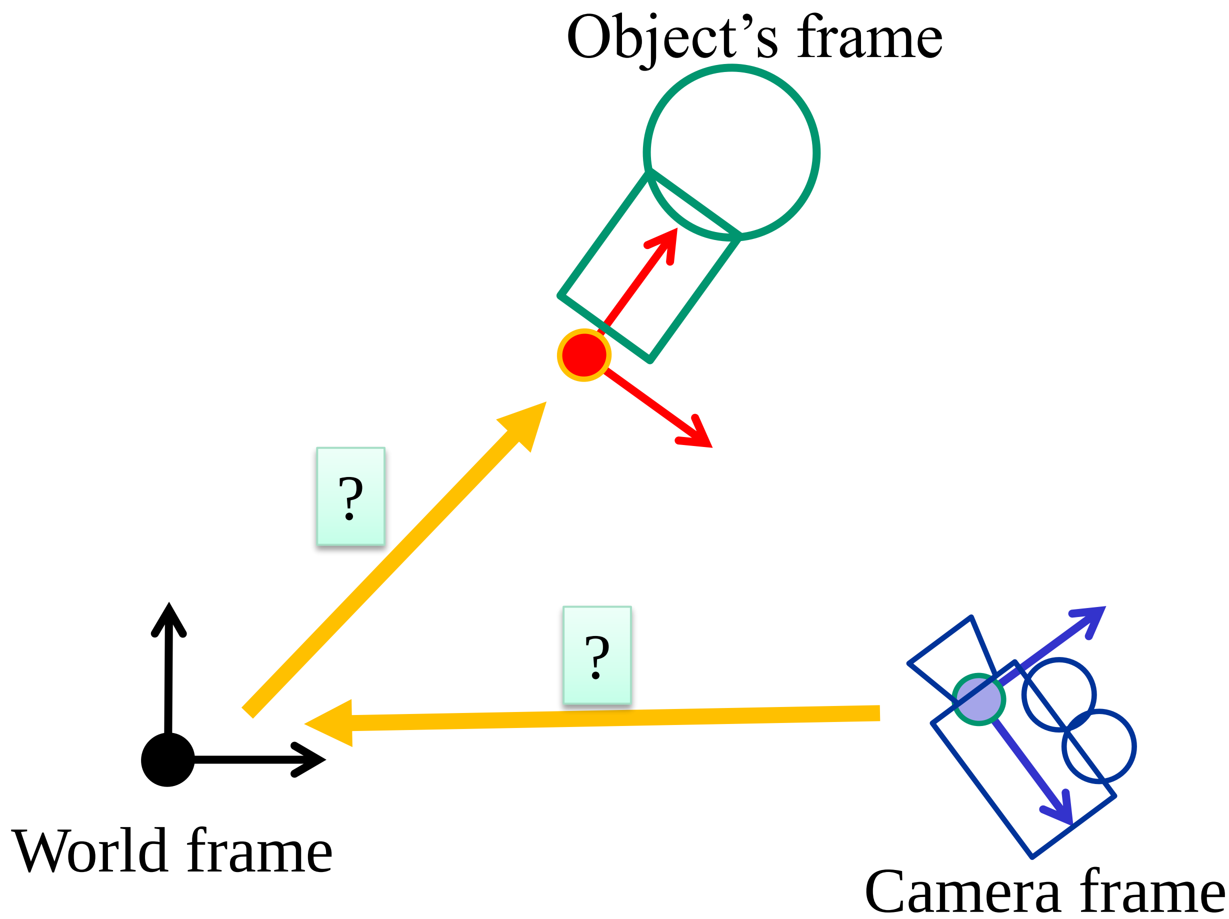
In the previous episodes

- Vectors & Tools

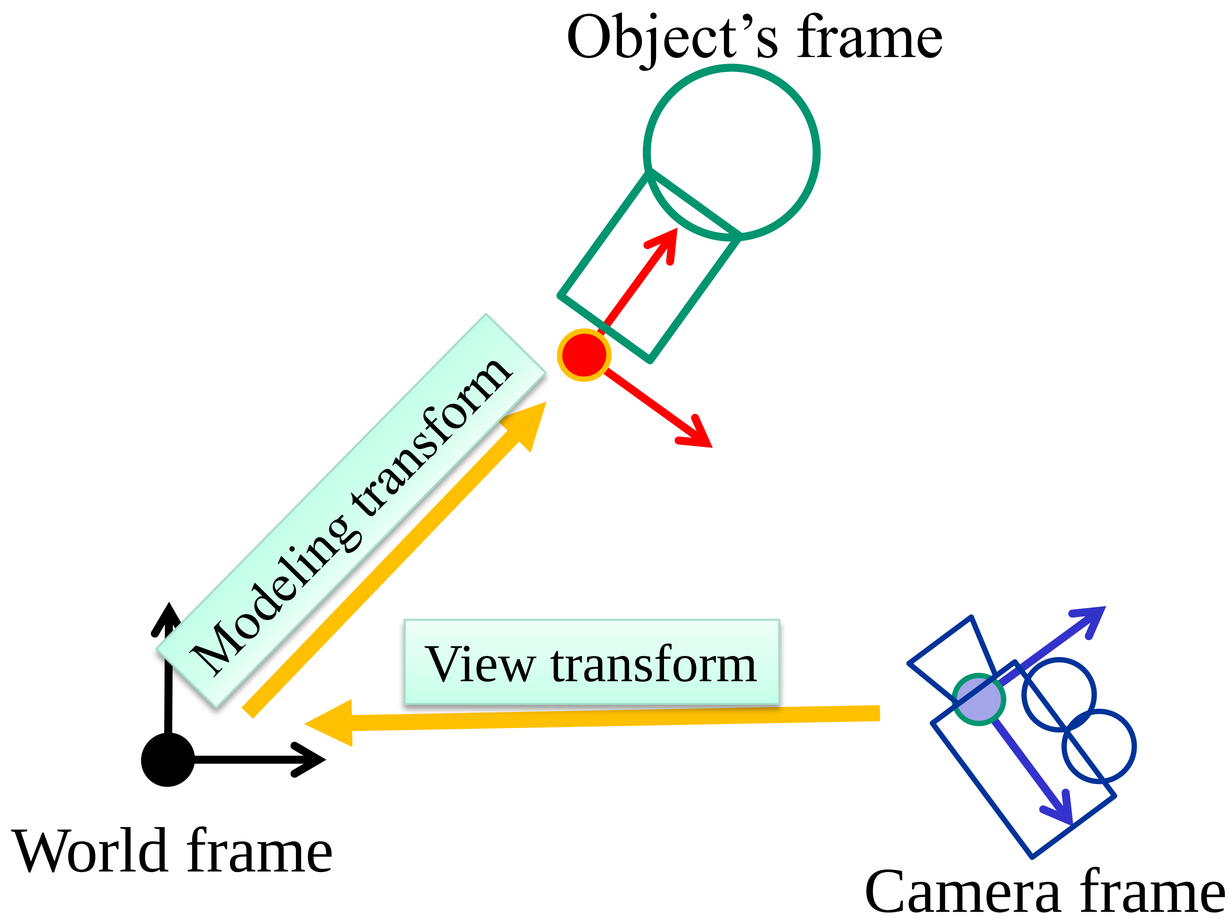
In the previous episodes



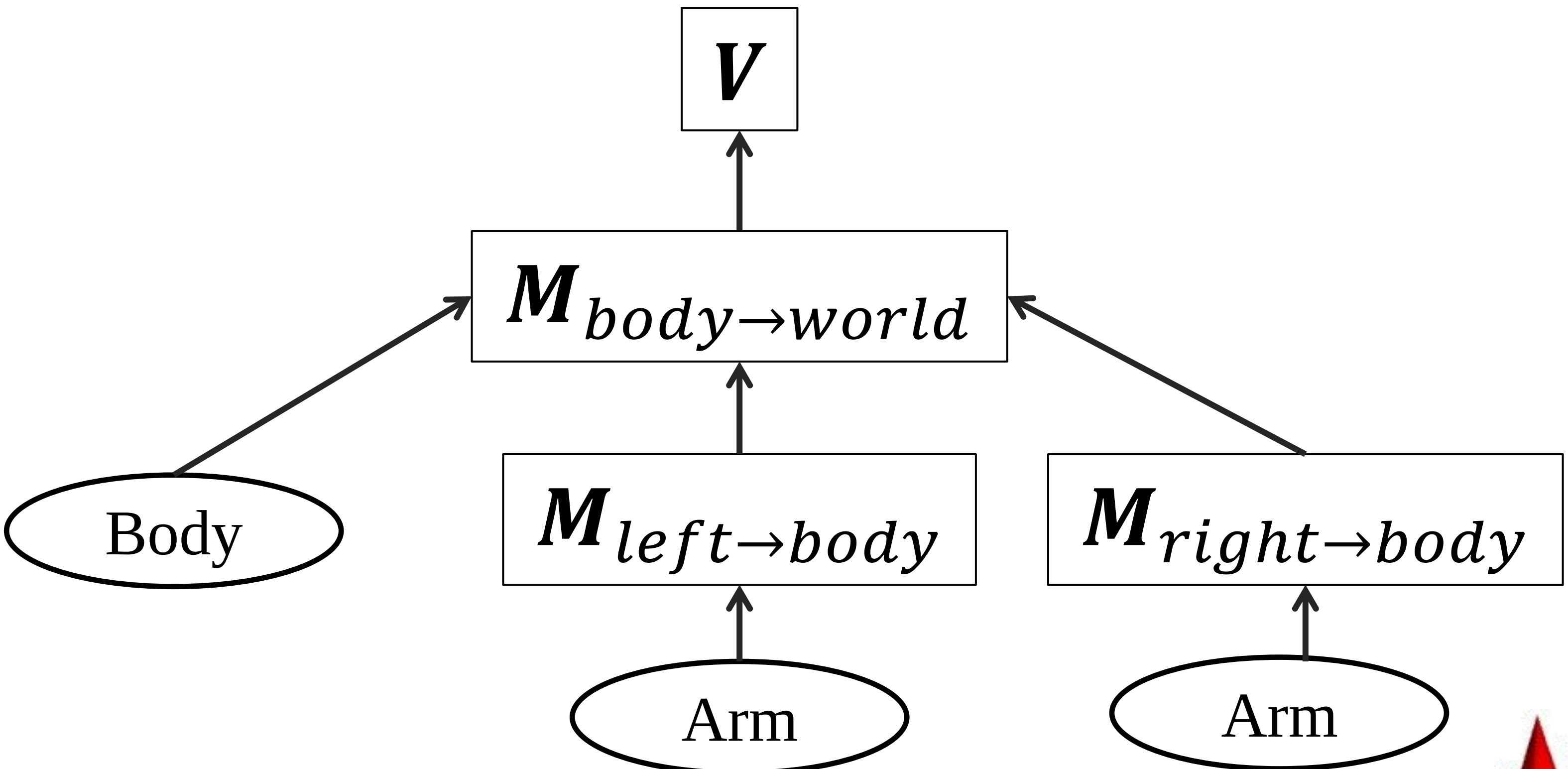
In the previous episodes



Model-View transformations



Scene graph

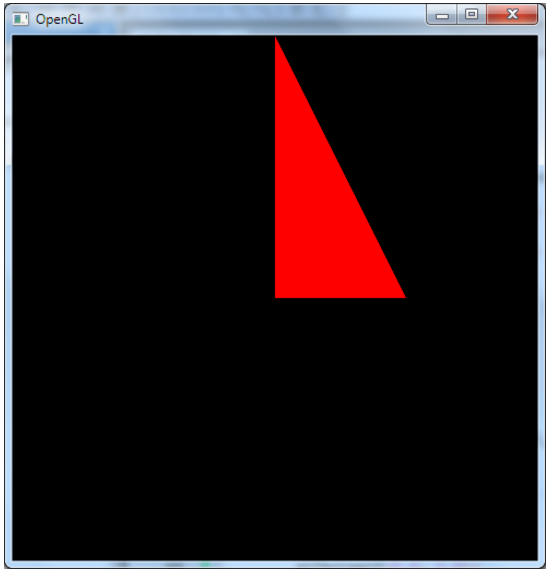


$\boldsymbol{V}$
$\boldsymbol{M}_{body\rightarrow world}$
Body
$\boldsymbol{M}_{left\rightarrow body}$
$\boldsymbol{M}_{right\rightarrow body}$
Arm
Arm

Quiz

- In OpenGL, the code on the left will always result in the picture of a triangle as shown on the right. True or False?

```
glBegin(GL_TRIANGLES);
      glVertex2f(0.0, 0.0);
      glVertex2f(0.5, 0.0);
      glVertex2f(0.0, 1.0);
glEnd();
```



Model-view matrix

- The Model-view matrix can be provided explicitly
  - `glLoadMatrix*(. . . . . . . . .)`
- Or, more commonly, constructed by multiplying with elementary matrices **on the right**.
  - `glLoadIdentity();` $\mathcal{M} = I$
  - `glTranslatef(…);` $\mathcal{M} = IT$
  - `glRotatef(…);` $\mathcal{M} = ITR$

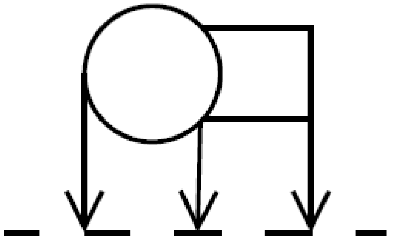
Today

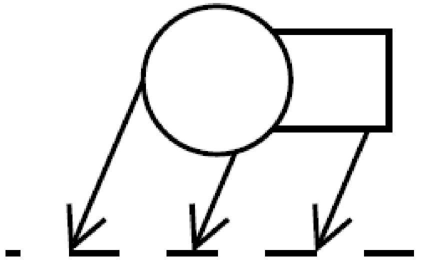
- The Model-View matrix is not the only step that the vertices pass before being displayed.

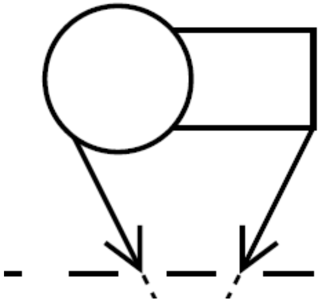
- After the MV transform the vertices are *projected*.

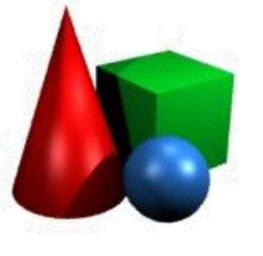
Planar geometric projections

- **Orthographic**
  - Front-top-bottom
  - Axonometric (isometry, dimetry,…)
- **Oblique**
  - Cavalier, cabinet
- **Perspective**



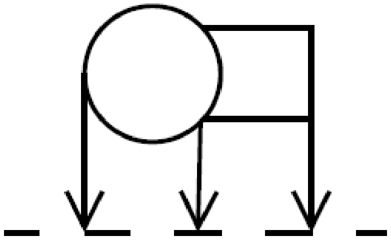


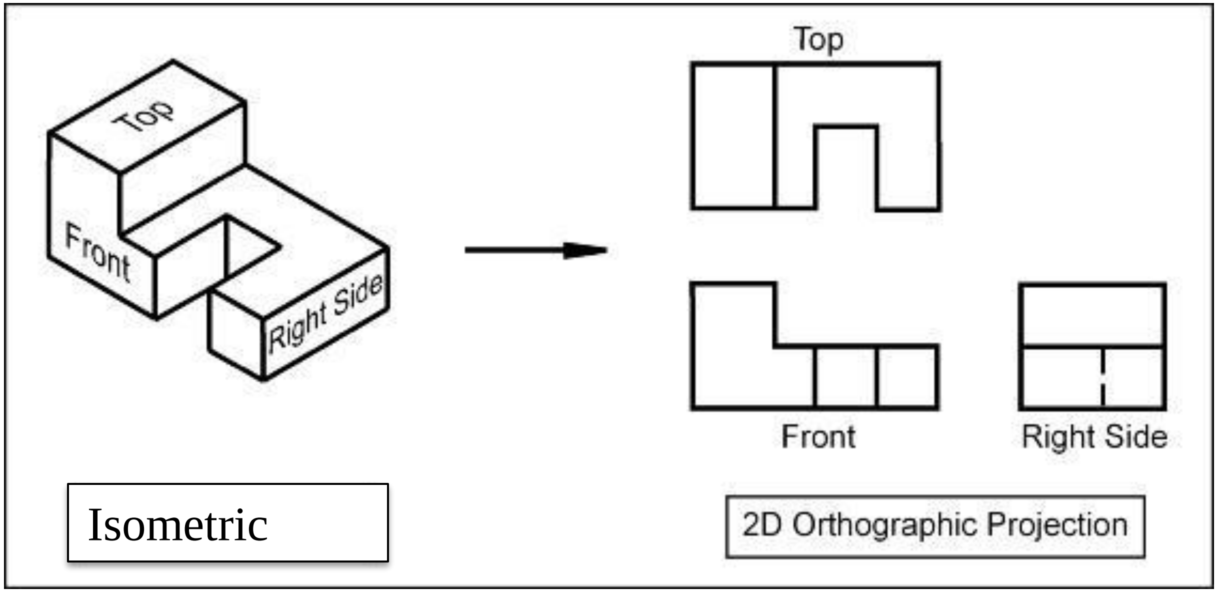




Planar geometric projections

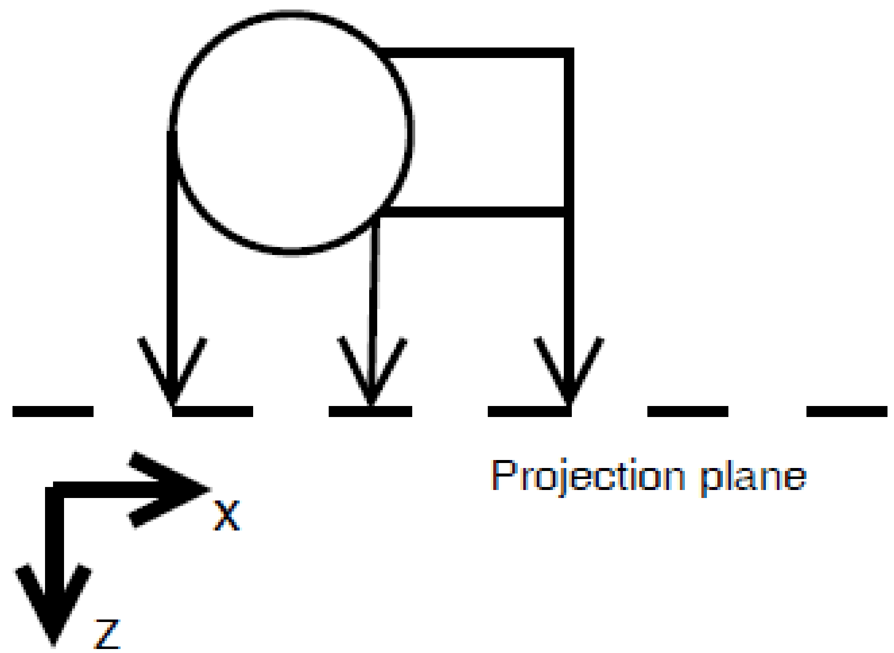
- **Orthographic**
  - Front-top-bottom
  - Axonometric (isometry, dimetry,…)





http://www.engineeringessentials.com/ege/ortho/ortho_page2.htm

Orthographic projection



$$x' = x \qquad y' = y \qquad z' = 0$$

Quiz

- Represent orthographic projection as an affine transformation matrix.

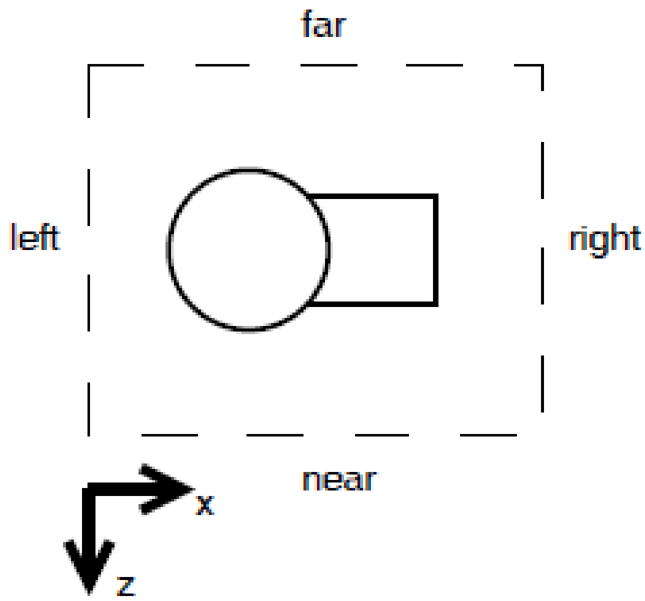
Quiz

- Represent orthographic projection as an affine transformation matrix.

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \left(\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Clipping box

- The screen is not infinite, and we have to specify the actual area that will be displayed: *the clipping box*.



Clipping box

- The clipping box is defined using six *clipping planes*: left, right, top, bottom, near, far.
- It is convenient to transform the space so that the clipping box turns into the cube

$$\{-1 \leq x \leq 1, -1 \leq y \leq 1, -1 \leq z \leq 1\}$$

- The result is often referred to as *normalized device coordinates* or *the clip space*.

Orthographic projection

- Thus, before projecting to (x, y) we normalize to clip space:

$$x' = \frac{2}{x_{\mathrm{r}} - x_{\mathrm{l}}}\left(x - \frac{x_{\mathrm{r}} + x_{\mathrm{l}}}{2}\right)$$

$$y' = \frac{2}{y_{\mathrm{t}} - y_{\mathrm{b}}}\left(y - \frac{y_{\mathrm{t}} + y_{\mathrm{b}}}{2}\right)$$

$$z' = \frac{2}{z_{\mathrm{n}} - z_{\mathrm{f}}}\left(z - \frac{z_{\mathrm{n}} + z_{\mathrm{f}}}{2}\right)$$

Orthographic projection

$$x' = \frac{2}{x_{\mathrm{r}} - x_{\mathrm{l}}} x - \frac{x_{\mathrm{r}} + x_{\mathrm{l}}}{x_{\mathrm{r}} - x_{\mathrm{l}}}$$

$$y' = \frac{2}{y_{\mathrm{t}} - y_{\mathrm{b}}} y - \frac{y_{\mathrm{t}} + y_{\mathrm{b}}}{y_{\mathrm{t}} - y_{\mathrm{b}}}$$

$$z' = \frac{2}{z_{\mathrm{n}} - z_{\mathrm{f}}} z - \frac{z_{\mathrm{n}} + z_{\mathrm{f}}}{z_{\mathrm{n}} - z_{\mathrm{f}}}$$

Orthographic projection

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \left(\begin{array}{ccc|c} \frac{2}{x_\mathrm{r}-x_\mathrm{l}} & 0 & 0 & -\frac{x_\mathrm{r}+x_\mathrm{l}}{x_\mathrm{r}-x_\mathrm{l}} \\ 0 & \frac{2}{y_\mathrm{t}-y_\mathrm{b}} & 0 & -\frac{y_\mathrm{t}+y_\mathrm{b}}{y_\mathrm{t}-y_\mathrm{b}} \\ 0 & 0 & \frac{2}{z_\mathrm{n}-z_\mathrm{f}} & -\frac{z_\mathrm{n}+z_\mathrm{f}}{z_\mathrm{n}-z_\mathrm{f}} \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Orthographic projection

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \left(\begin{array}{ccc|c} \frac{2}{x_\mathrm{r}-x_\mathrm{l}} & 0 & 0 & -\frac{x_\mathrm{r}+x_\mathrm{l}}{x_\mathrm{r}-x_\mathrm{l}} \\ 0 & \frac{2}{y_\mathrm{t}-y_\mathrm{b}} & 0 & -\frac{y_\mathrm{t}+y_\mathrm{b}}{y_\mathrm{t}-y_\mathrm{b}} \\ 0 & 0 & \frac{2}{z_\mathrm{n}-z_\mathrm{f}} & -\frac{z_\mathrm{n}+z_\mathrm{f}}{z_\mathrm{n}-z_\mathrm{f}} \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

- In fact, we do not want to lose the z coordinate just yet.

Orthographic projection

- Denote the matrix shown before by $\boldsymbol{P}_{\mathrm{ort}}$. Now, given the model & view transformations, the complete mapping for each vertex $\boldsymbol{x}_i$ to clip space is given by:

$$\boldsymbol{P}_{\mathrm{ort}}\boldsymbol{V}\boldsymbol{M}\boldsymbol{x}_{\boldsymbol{i}}$$

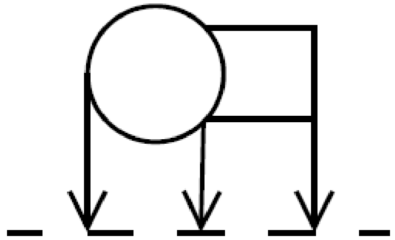
Orthographic projection

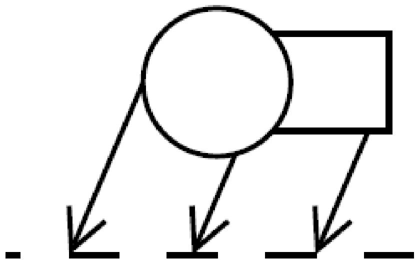
- The projection matrix is also part of OpenGL state:

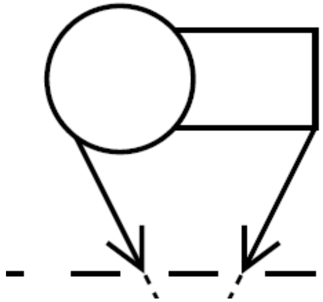
```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(left, right, bottom, top, near, far);
glMatrixMode(GL_MODELVIEW);
```

Planar geometric projections

- **Orthographic**
  - Front-top-bottom
  - Axonometric (isometry, dimetry,…)
- **Oblique**
  - Cavalier, cabinet
- **Perspective**

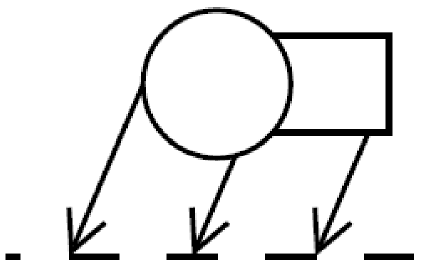


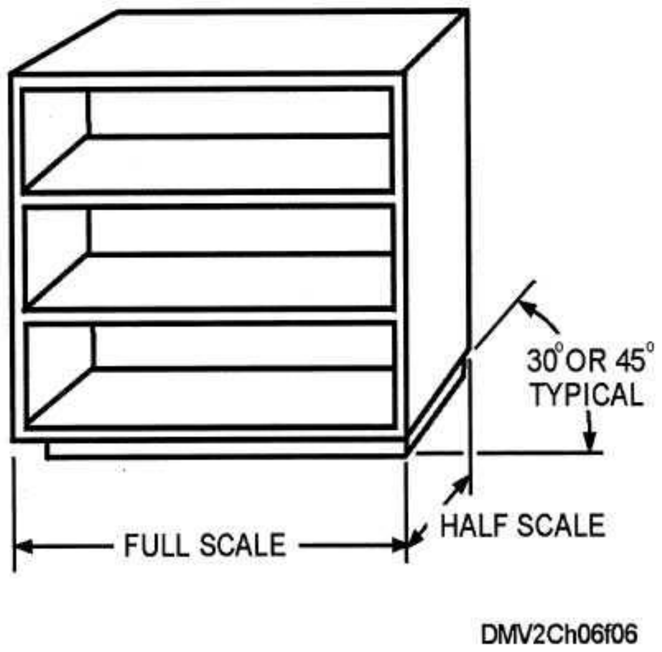




Planar geometric projections

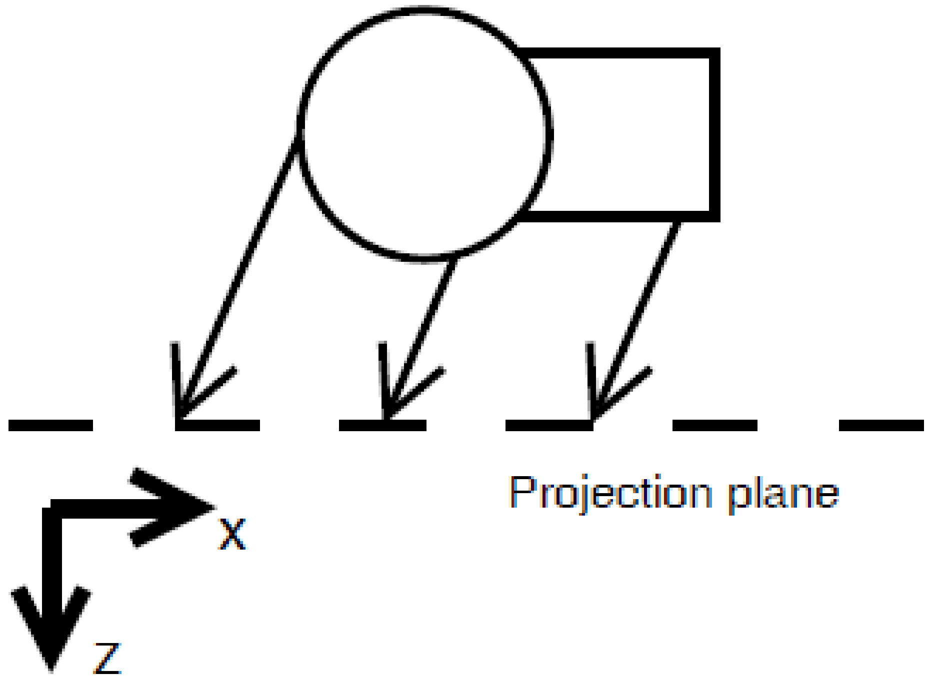
- **Oblique**
  - Cavalier, cabinet



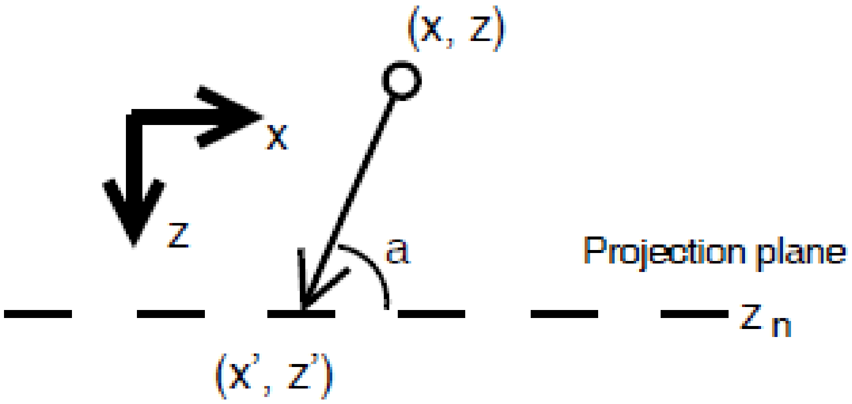


http://draftingmanuals.tpub.com/14276/css/14276_308.htm

Oblique projection



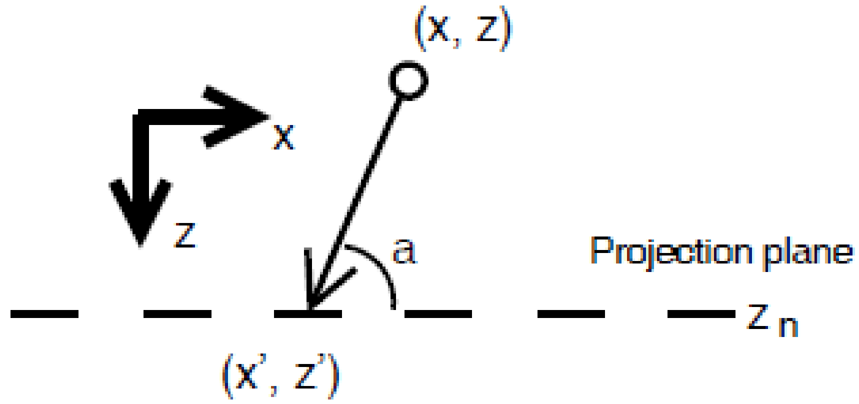
Oblique projection



$$x' =$$

$$y' =$$

Oblique projection



$$x' = x - (z_n - z)\cot\alpha$$

$$y' = y - (z_n - z)\cot\beta$$

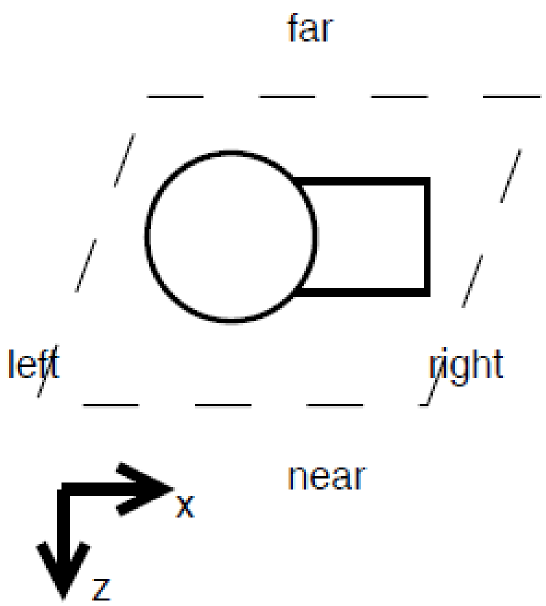
Oblique projection

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \left(\begin{array}{ccc|c} 1 & 0 & \cot\alpha & -z_n \cot\alpha \\ 0 & 1 & \cot\beta & -z_n \cot\beta \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Denote this matrix by ***H***

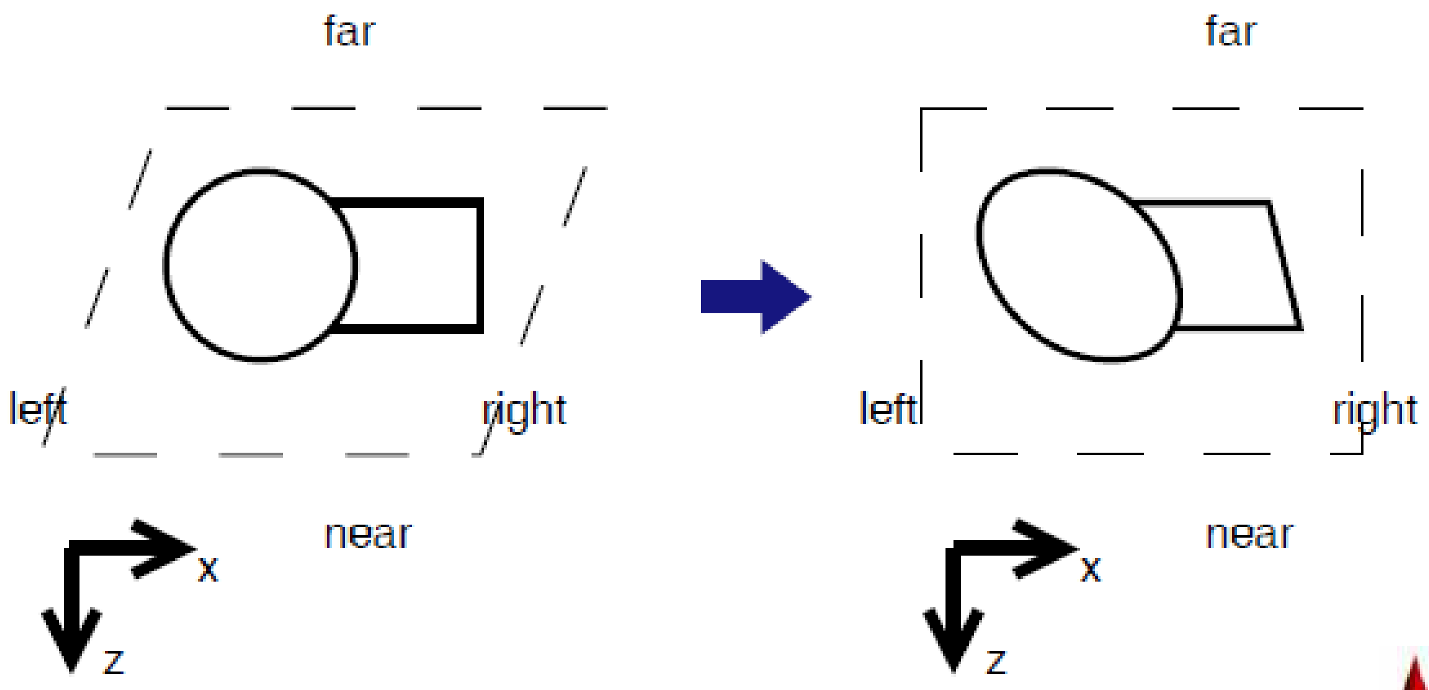
Oblique projection

- Similarly, we can define a *clipping parallelepiped* for the oblique projection.



Oblique projection

- Matrix ***H*** would make it rectangular



Oblique projection

- … we can then further map the resulting box to (-1, +1) clip space using the standard orthogonal projection:

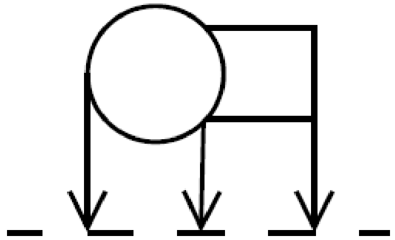
$$\boldsymbol{P}_{\mathrm{obl}} = \boldsymbol{P}_{\mathrm{ort}}\boldsymbol{H}$$

Oblique projection

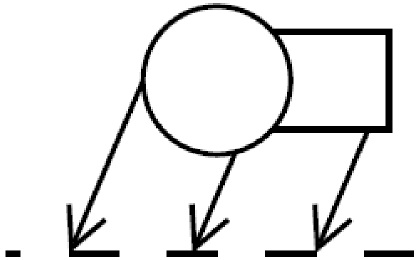
```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(left, right, bottom, top, near, far);

float H[] = {      1,       0,   0,   0,
                   0,       1,   0,   0,
              cot(a), cot(b),   1,   0,
                   0,       0,   0,   1};
glMultMatrixf(H);

glMatrixMode(GL_MODELVIEW);
```

Planar geometric projections

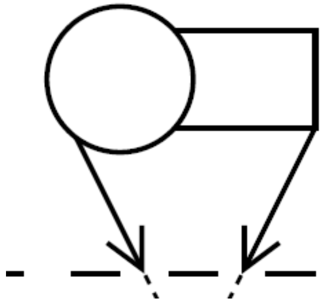
- **Orthographic**
  - Front-top-bottom
  - Axonometric (isometry, dimetry,…)



- **Oblique**
  - Cavalier, cabinet

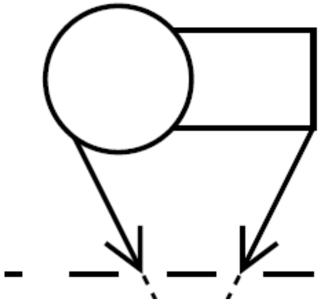


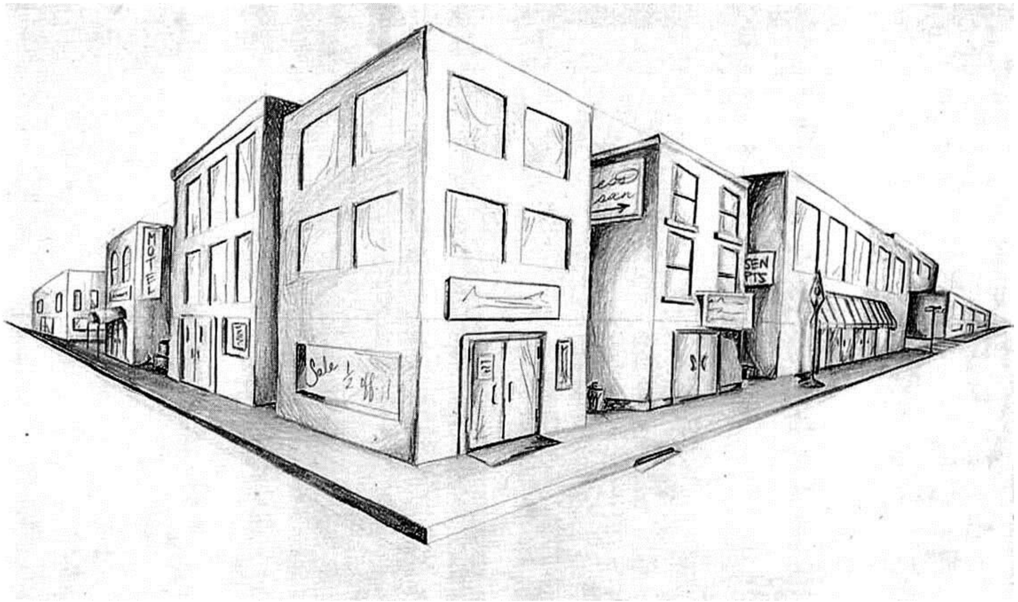
- **Perspective**



Planar geometric projections

- **Perspective**





http://www.perspectiveprojection.com/

Homogeneous coordinates

- Remember we defined a representation for points in homogeneous coordinates:

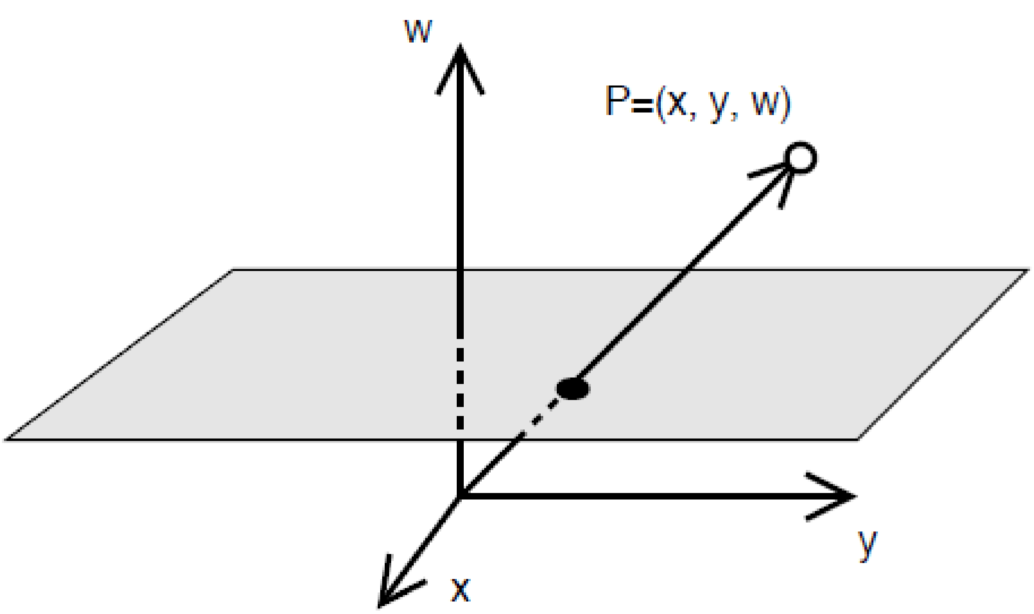
$$(x, y, z) \rightarrow (x, y, z, 1)$$

- Let us now define an inverse correspondence: any vector (x, y, z, w) will denote the point $\left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right) \in \mathbb{R}^3$

Homogeneous coordinates

- Similarly for two dimensions:

$$(x, y, w) \leftrightarrow \left(\frac{x}{w}, \frac{y}{w}\right)$$

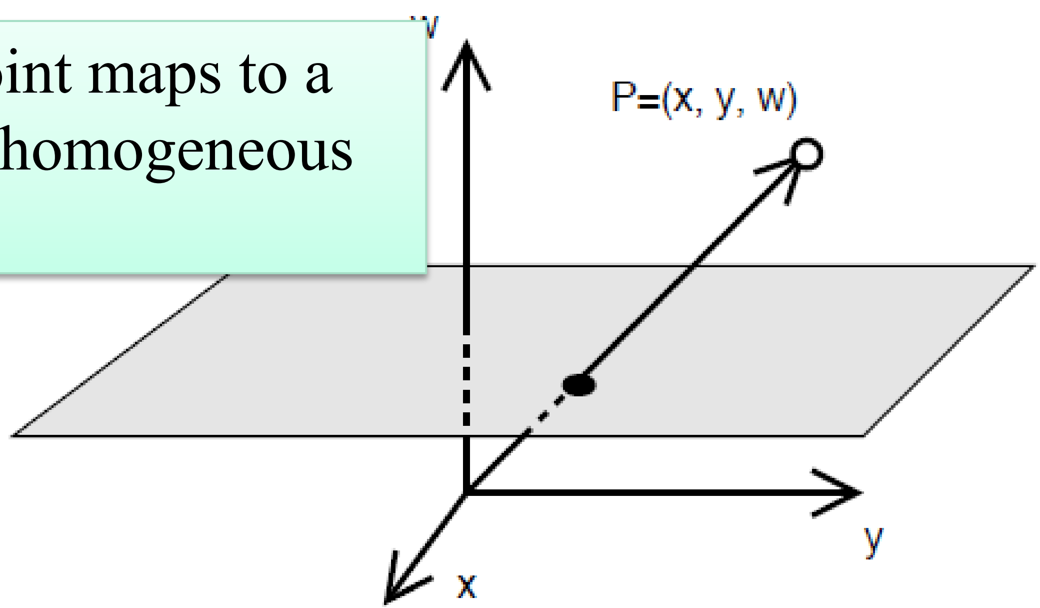


Homogeneous coordinates

- Similarly for two dimensions:

$$(x, y, w) \leftrightarrow \left(\frac{x}{w}, \frac{y}{w}\right)$$

One "actual" point maps to a whole "line" in homogeneous coordinates

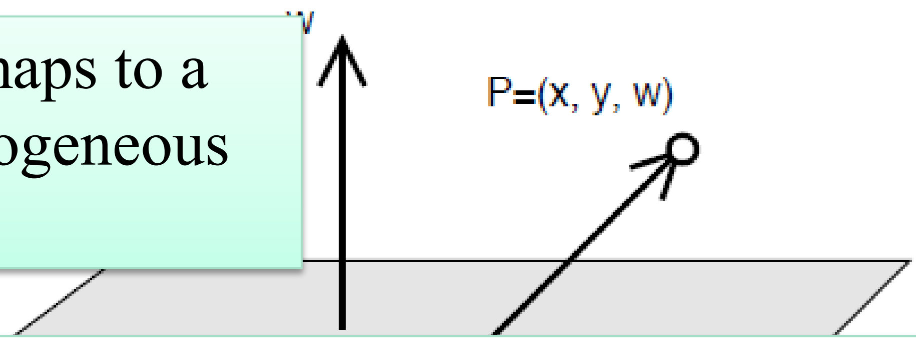


Homogeneous coordinates

- Similarly for two dimensions:

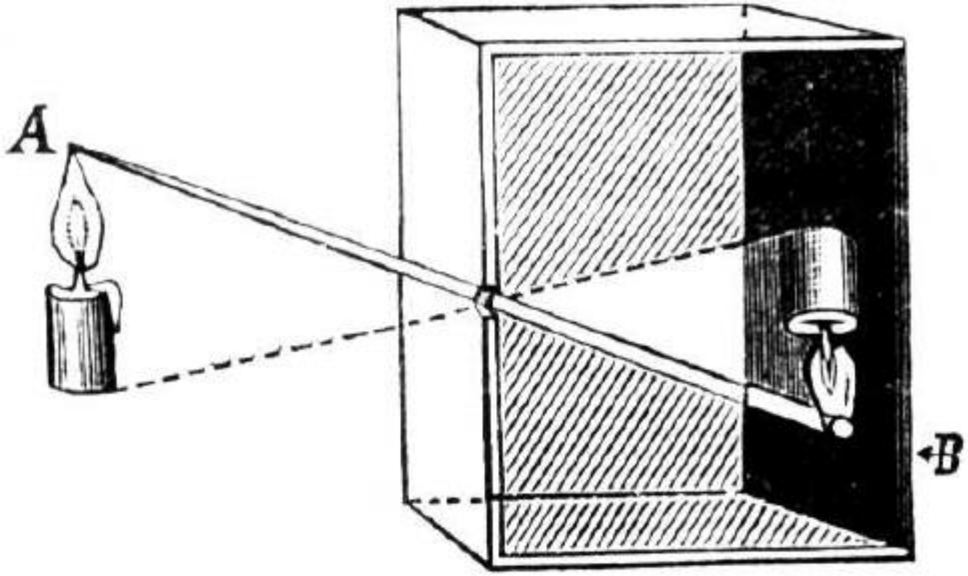
$$(x, y, w) \leftrightarrow \left(\frac{x}{w}, \frac{y}{w}\right)$$

One "actual" point maps to a whole "line" in homogeneous coordinates

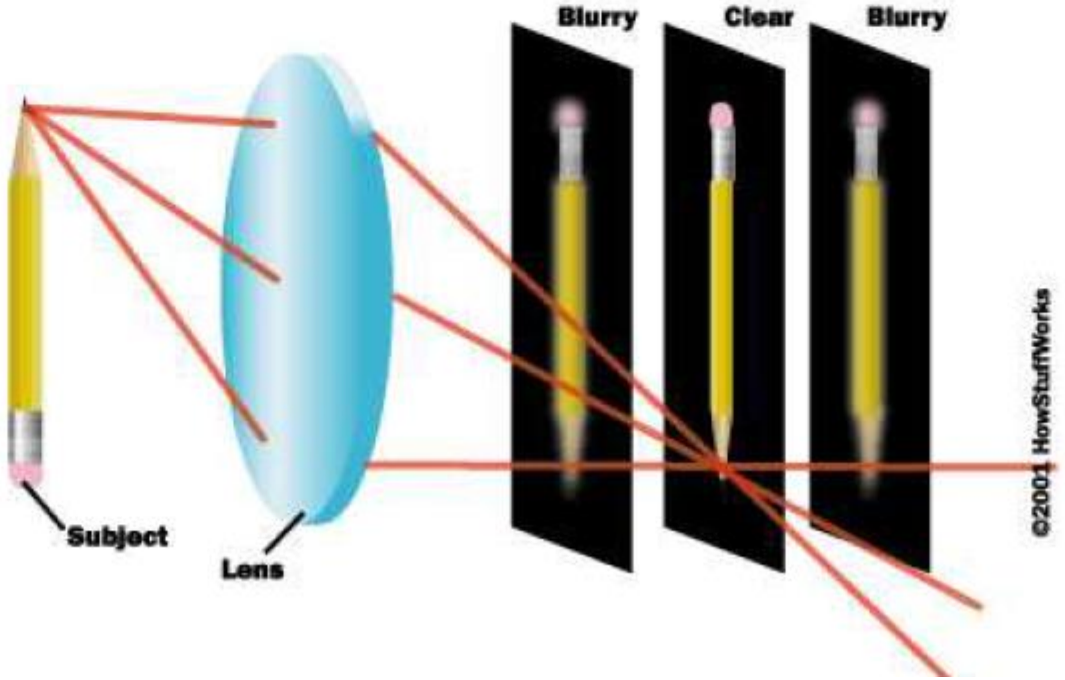


Homogeneous coordinates with $w = 0$, as before, do not correspond to any "actual point". It is a vector (i.e. a direction) or a "point at infinity".

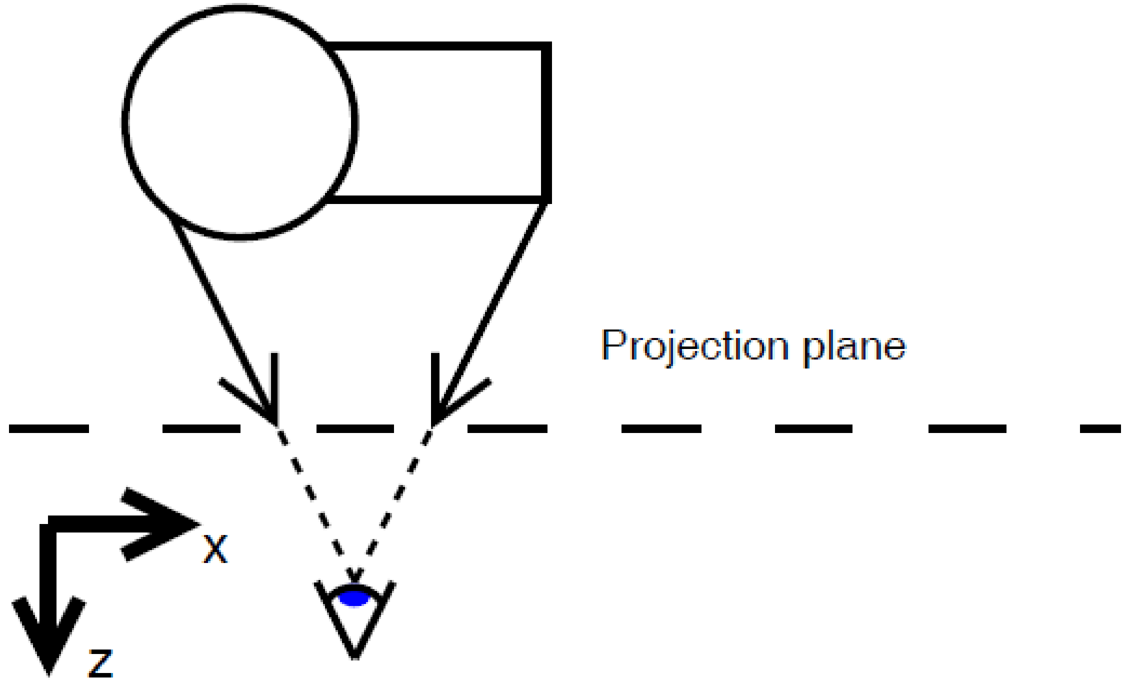
Camera obscura



Actual camera

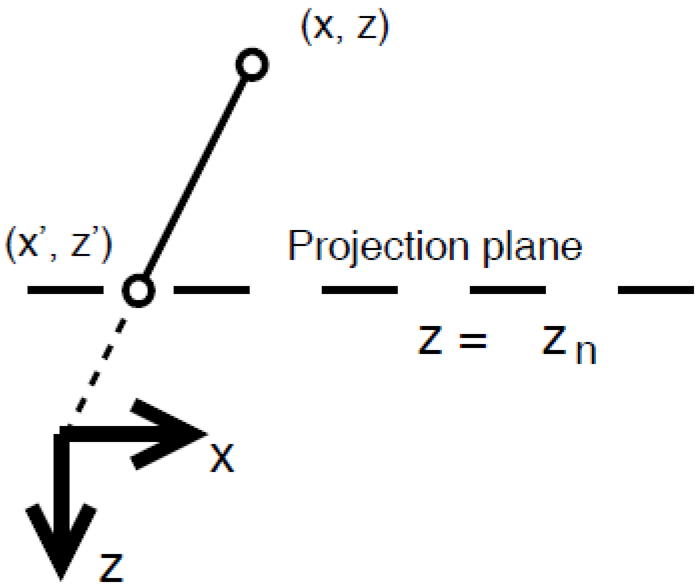


What we shall be doing



Perspective projection

- Suppose the center of projection is $(0, 0, 0)$. Let us project to the plane $z = z_n$



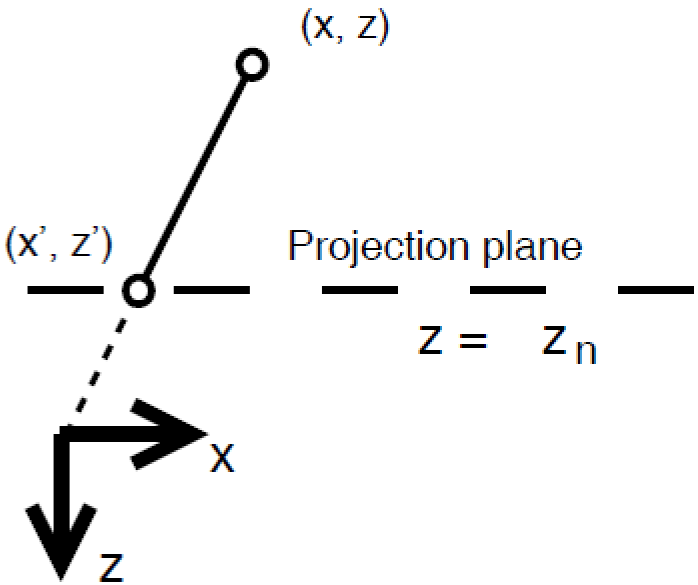
$$x' =$$

$$y' =$$

$$z' =$$

Perspective projection

- Suppose the center of projection is $(0, 0, 0)$. Let us project to the plane $z = z_n$



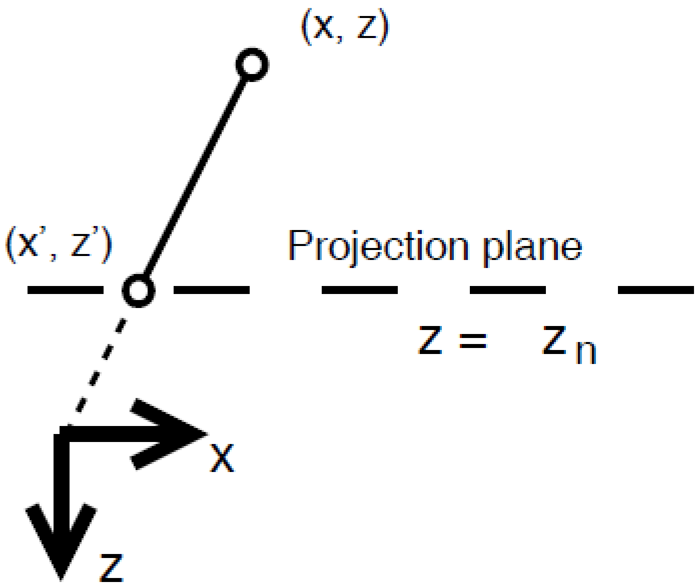
$$x' = x \cdot \frac{z_n}{z}$$

$$y' = y \cdot \frac{z_n}{z}$$

$$z' = z \cdot \frac{z_n}{z}$$

Perspective projection

- Suppose the center of projection is $(0, 0, 0)$. Let us project to the plane $z = z_n$



$$x' = x \cdot \frac{z_n}{z} = \frac{x}{z/z_n}$$

$$y' = y \cdot \frac{z_n}{z} = \frac{y}{z/z_n}$$

$$z' = z \cdot \frac{z_n}{z} = \frac{z}{z/z_n}$$

Perspective projection

- We need to map

$$(x, y, z) \rightarrow \left(\frac{x}{z/z_n}, \frac{y}{z/z_n}, \frac{z}{z/z_n}\right)$$

Perspective projection

- We need to map

$$(x, y, z) \rightarrow \left(\frac{x}{z/z_n}, \frac{y}{z/z_n}, \frac{z}{z/z_n} \right)$$

Is this a linear transformation?

Perspective projection

- We need to map

$$(x, y, z) \to \left(\frac{x}{z/z_n}, \frac{y}{z/z_n}, \frac{z}{z/z_n}\right)$$

In homogeneous coordinates, though, it corresponds to:

$$(x, y, z, 1) \to \left(\frac{x}{z/z_n}, \frac{y}{z/z_n}, \frac{z}{z/z_n}, 1\right)$$

Perspective projection

- We need to map

$$(x, y, z) \to \left(\frac{x}{z/z_n}, \frac{y}{z/z_n}, \frac{z}{z/z_n} \right)$$

In homogeneous coordinates, though, it corresponds to:

$$(x, y, z, 1) \to (x, y, z, z/z_n)$$

Perspective projection

- We need to map

$$(x, y, z) \rightarrow \left(\frac{x}{z/z_n}, \frac{y}{z/z_n}, \frac{z}{z/z_n} \right)$$

In homogeneous coordinates, though, it corresponds to:

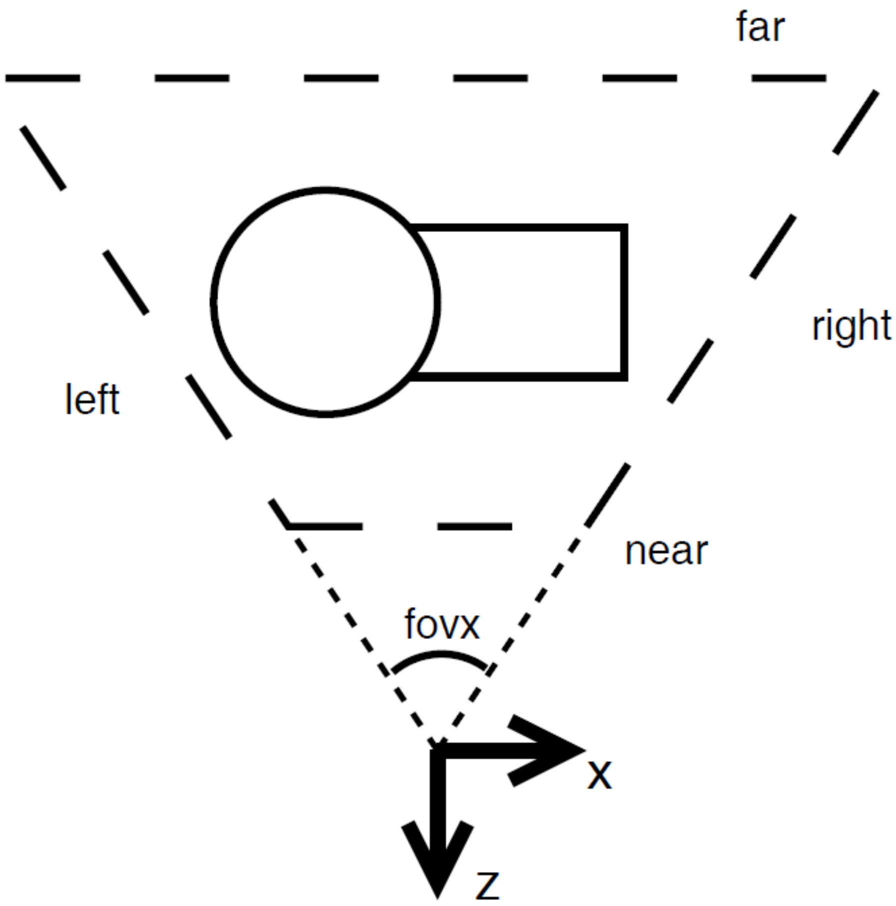
$$(x, y, z, 1) \rightarrow (x, y, z, z/z_n)$$

What matrix performs this?

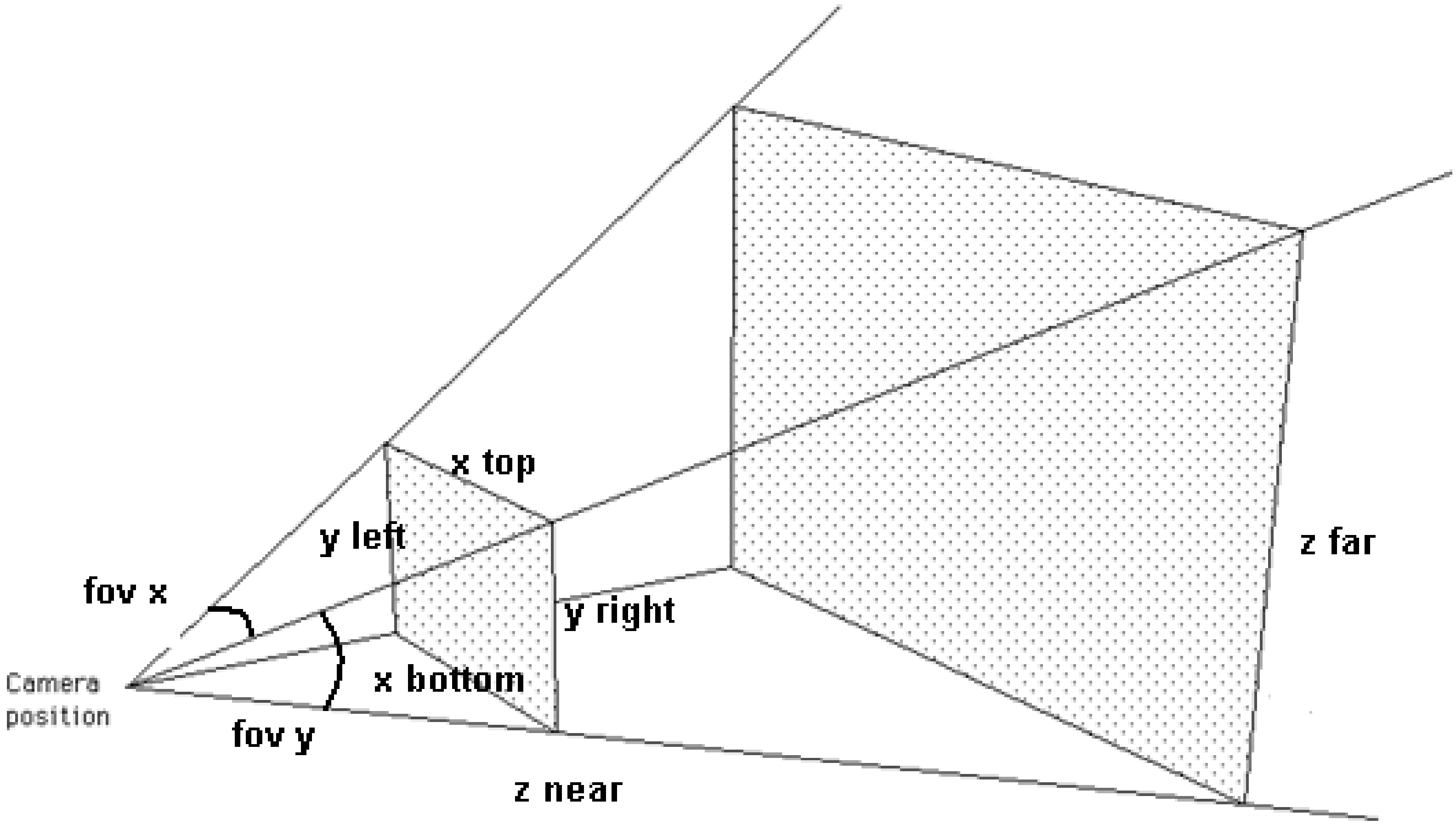
Perspective projection

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \left(\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 1/z_n & 0 \end{array}\right) \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

View frustum

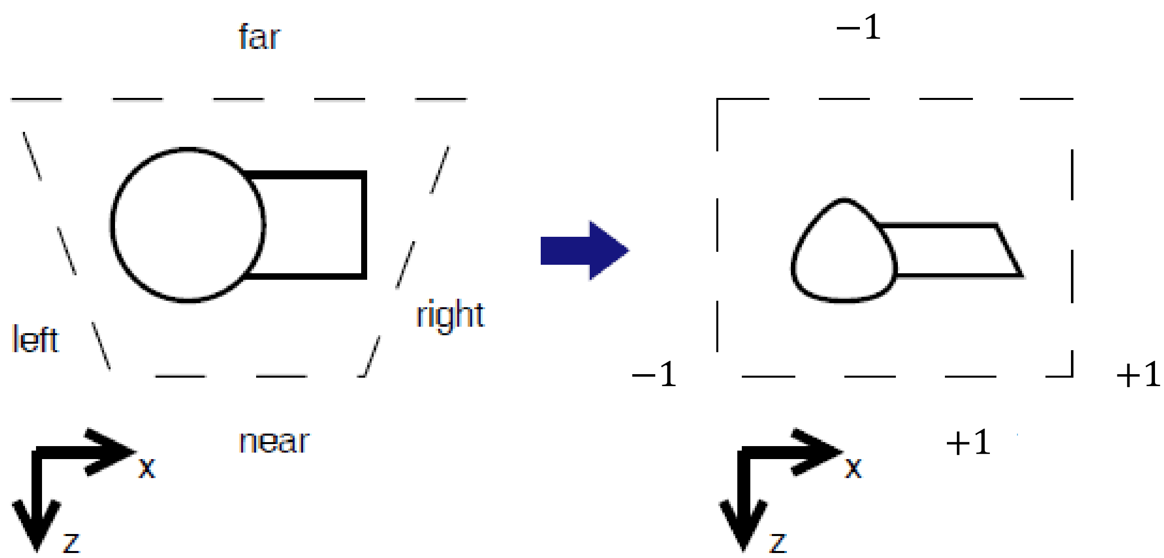


View frustum



View frustum normalization

- We want to do the same trick as before – normalize the view frustum into $2 \times 2 \times 2$ clipping space.



View frustum normalization

- We want to do the same trick as before – normalize the view frustum into $2 \times 2 \times 2$ clipping space.

- Remember, our current mapping is

$$(x, y, z) \rightarrow \left(\frac{x}{z/z_n}, \frac{y}{z/z_n}, z_n\right)$$

View frustum normalization

$$(x, y, z) \to \left(\frac{x}{z/z_n}, \frac{y}{z/z_n}, z_n\right)$$

View frustum normalization

Rescaling this from {left, right} to {-1, +1} is straightforward

$$(x, y, z) \rightarrow \left(\frac{x}{z/z_n}, \frac{y}{z/z_n}, z_n \right)$$

View frustum normalization

Rescaling this from {bottom, top} to {-1, +1} is straightforward

$$(x, y, z) \rightarrow \left(\frac{x}{z/z_n}, \frac{y}{z/z_n}, z_n\right)$$

View frustum normalization

This is a problem!

$$(x, y, z) \rightarrow \left(\frac{x}{z/z_n}, \frac{y}{z/z_n}, z_n \right)$$

View frustum normalization

This is a problem!

We have lost depth information!

$$(x, y, z) \rightarrow \left(\frac{x}{z/z_n}, \frac{y}{z/z_n}, z_n\right)$$

View frustum normalization

This is a problem!

Turns out a better way to transform z is:

$$A + \frac{B}{z}$$

We have lost depth information!

$$(x, y, z) \to \left(\frac{x}{z/z_n}, \frac{y}{z/z_n}, z_n\right)$$

View frustum normalization

Depth information is preserved

$$(x, y, z) \rightarrow \left(\frac{x}{z/z_n}, \frac{y}{z/z_n}, A + \frac{B}{z} \right)$$

View frustum normalization

Depth information is preserved

It is a linear transformation in homogeneous coordinates:

$$(x, y, z, 1) \rightarrow \left(x, y, \frac{Az + B}{z_n}, \frac{z}{z_n}\right)$$

$$(x, y, z) \rightarrow \left(\frac{x}{z/z_n}, \frac{y}{z/z_n}, A + \frac{B}{z}\right)$$

View frustum normalization

Depth information is preserved

It is a linear transformation in homogeneous coordinates:

$$(x, y, z, 1) \to \left(x, y, \frac{Az + B}{}, \frac{z}{}\right)$$

We can choose A and B to ensure that $z_n \to 1$, $z_f \to -1$

$$(x, y, z) \to \left(\frac{x}{z/z_n}, \frac{y}{z/z_n}, A + \frac{B}{z}\right)$$

View frustum normalization

We can choose A and B to ensure that $z_n \to 1$, $z_f \to -1$

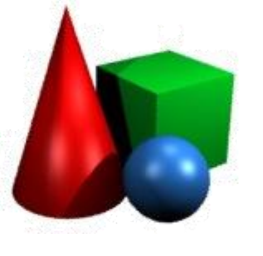
$$(x, y, z) \to \left(\frac{x}{z/z_n}, \frac{y}{z/z_n}, A + \frac{B}{z} \right)$$

View frustum normalization

We can choose A and B to ensure that $z_n \to 1$, $z_f \to -1$

$$A + \frac{B}{z_n} = 1 \qquad\qquad A + \frac{B}{z_f} = -1$$

$$(x, y, z) \to \left(\frac{x}{z/z_n}, \frac{y}{z/z_n}, A + \frac{B}{z} \right)$$



View frustum normalization

We can choose A and B to ensure that $z_n \to 1$, $z_f \to -1$

$$A = -\frac{(z_f + z_n)}{z_f - z_n} \qquad \mathrm{B} = \frac{2z_n z_f}{z_f - z_n}$$

$$(x, y, z) \to \left(\frac{x}{z/z_n}, \frac{y}{z/z_n}, A + \frac{B}{z}\right)$$

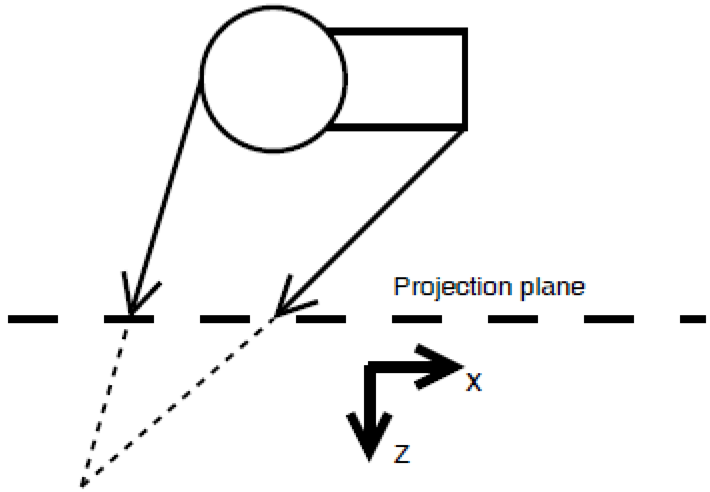
Perspective transformation

- Putting all together:

$$\mathbf{P}_{\text{persp}} = \left(\begin{array}{ccc|c} \frac{2z_n}{x_r - x_l} & 0 & 0 & 0 \\ 0 & \frac{2z_n}{y_t - y_b} & 0 & 0 \\ 0 & 0 & \frac{-(z_f + z_n)}{z_f - z_n} & \frac{2z_n z_f}{z_f - z_n} \\ \hline 0 & 0 & 1 & 0 \end{array}\right)$$

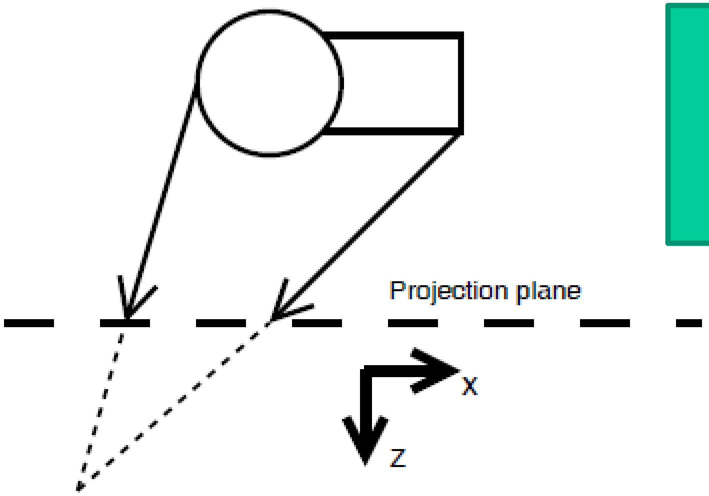
Skewed perspective

- Sometimes you might want to project so that the projection plane is not perpendicular to the viewing direction



Skewed perspective

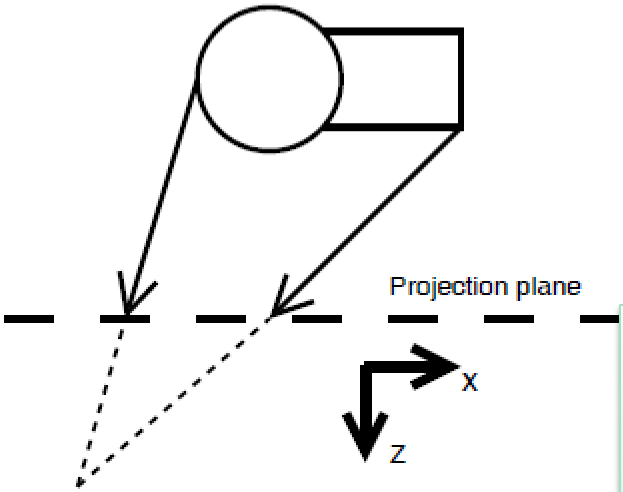
- Sometimes you might want to project so that the projection plane is not perpendicular to the viewing direction



Quiz: How to do this?

Skewed perspective

- Sometimes you might want to project so that the projection plane is not perpendicular to the viewing direction



Quiz: How to do this?

Apply the “unskewing” H matrix first.

Perspective projection

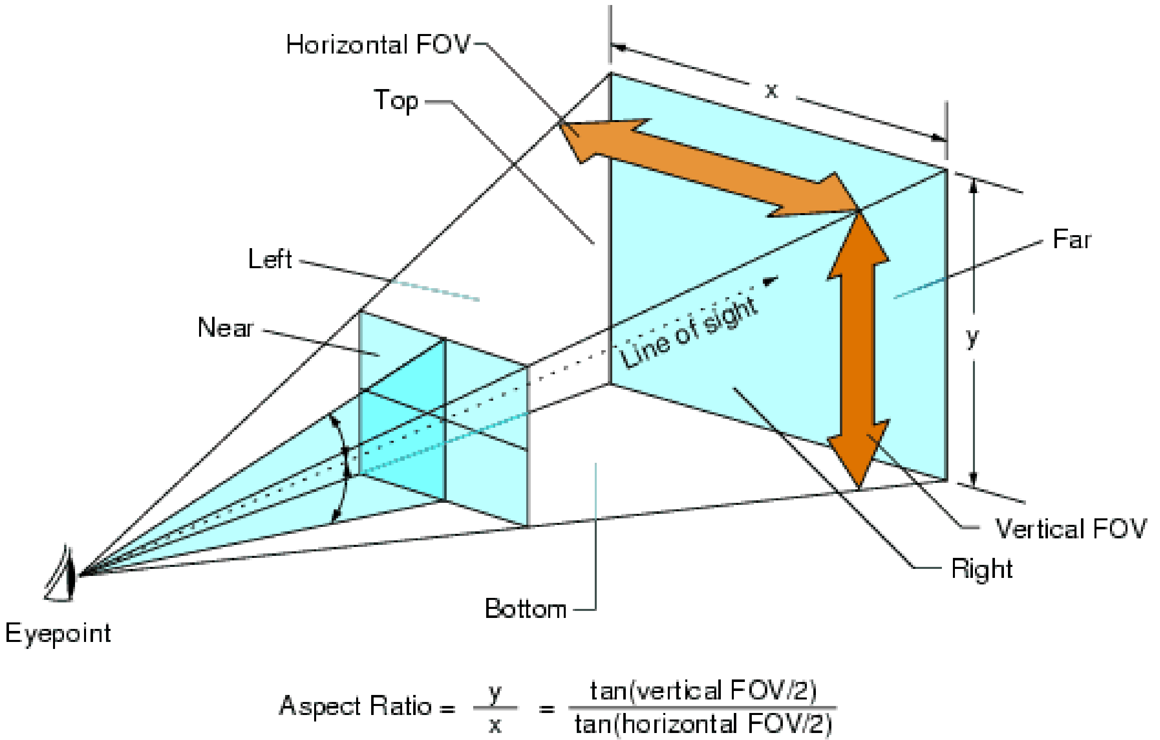
```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(left, right, bottom, top, dNear, dFar);
glMatrixMode(GL_MODELVIEW);
```

Perspective projection

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(fovY, aspect, dNear, dFar);
glMatrixMode(GL_MODELVIEW);
```

Perspective projection

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(fovY, aspect, dNear, dFar);
glMatrixMode(GL_MODELVIEW);
```



http://www.codinghorror.com/blog/2007/08/widescreen-and-fov.html

Field of View

- Quiz: How does the picture look like when FOV is very large?

Field of View

- Quiz: How does the picture look like when FOV is very large?

"Fisheye lens effect"

http://www.photoanswers.co.uk/Advice/Search-Results/Photopedia/Fisheye-lens-/

Field of View

- Quiz: How does the picture look like when FOV is very large?

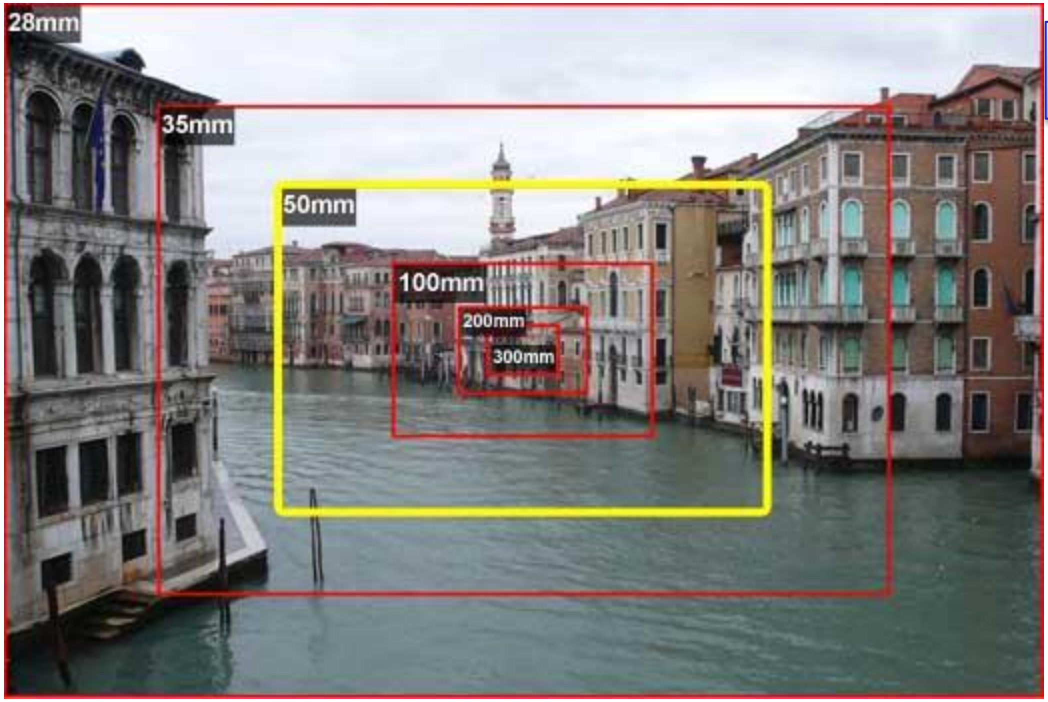
http://strlen.com/gfxengine/panquake/

Field of View

- Quiz: How does the picture look like when FOV is very small?

Field of View

- Quiz: How does the picture look like when FOV is very small?



"Zoom effect"

http://www.wexphotographic.com/?/guides/lens-sense.html

Field of View

- Quiz: How does the picture look like when FOV is very small?

“Zoom effect”

Resembles orthographic projection

http://www.mir.com.my/rb/photography/hardwares/classics/eos/EF-lenses/EF75300mm/

What we know so far

- We need to set up the **Model-View matrix** using
  - `glMatrixMode(GL_MODELVIEW)`
- We need to set up the **Projection matrix** using
  - `glMatrixMode(GL_PROJECTION)`
- When we emit vertices (via `glVertex**`), they are transformed as follows:

$$\boldsymbol{x} \rightarrow \boldsymbol{P} \cdot (\boldsymbol{V}\boldsymbol{M}) \cdot \boldsymbol{x}$$

Last two steps

- Last two steps of the pipeline:
  - Perspective division
    - $(x, y, z, w) \rightarrow \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}, 1\right)$
  - Viewport transform
    - x is mapped from $(-1, +1)$ to $(0, \text{screen width})$
    - y is mapped from $(-1, +1)$ to $(\text{screen height}, 0)$
    - z is mapped from $(-1, +1)$ to $(0, 1)$

Last two steps

- Last two steps of the pipeline:
  - Perspective division
    - $(x, y, z, w) \rightarrow \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}, 1\right)$
  - Viewport transform
    - x is mapped from $(-1, +1)$ to $(0, \text{screen width})$
    - y is mapped from $(-1, +1)$ to $(\text{screen height}, 0)$
    - z is mapped from $(-1, +1)$ to $(0, 1)$

$$x_{\text{win}} = \frac{\text{screen width}}{2}(x_{\text{norm}} + 1)$$

$$y_{\text{win}} = \frac{\text{screen height}}{2}(1 - y_{\text{norm}})$$

Last two steps

- Last two steps of the pipeline:
  - Perspective division
    - $(x, y, z, w) \rightarrow \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}, 1\right)$
  - Viewport transform
    - x is mapped from $(-1, +1)$ to $(0, \text{screen width})$
    - y is mapped from $(-1, +1)$ to $(\text{screen height}, 0)$
    - z is mapped from $(-1, +1)$ to $(0, 1)$

```
glViewport(0, 0, width, height);
glDepthRange(0, 1);
```

Standard Graphics Pipeline

Vertex transform

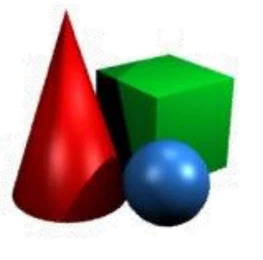
- $x' = PVMx$
- Perspective division
- Viewport transform

Culling and clipping

Rasterization

Fragment shading

Visibility tests & blending



Standard Graphics Pipeline

Vertex transform

- $\boxed{x' = PVMx}$ “Vertex shader”
- **Perspective division**
- **Viewport transform**

Culling and clipping

Rasterization

Fragment shading

Visibility tests & blending

Standard Graphics Pipeline

$$x' = PVMx$$

"Vertex shader"

Vertex

```
void main() {
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

Rasterization

Fragment shading

Visibility tests & blending

Standard Graphics Pipeline

$x' = PVMx$

"Vertex shader"

Vertex

```
void main() {
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
    normal = gl_NormalMatrix * gl_Normal;
}
```

Rasterization

Fragment shading

Visibility tests & blending

Standard Graphics Pipeline

- Vertex transform
  - $\boxed{x' = PVMx}$ “Vertex shader”
  - **Perspective division**
  - **Viewport transform**
- Culling and clipping
- Rasterization
- Fragment shading
- Visibility tests & blending