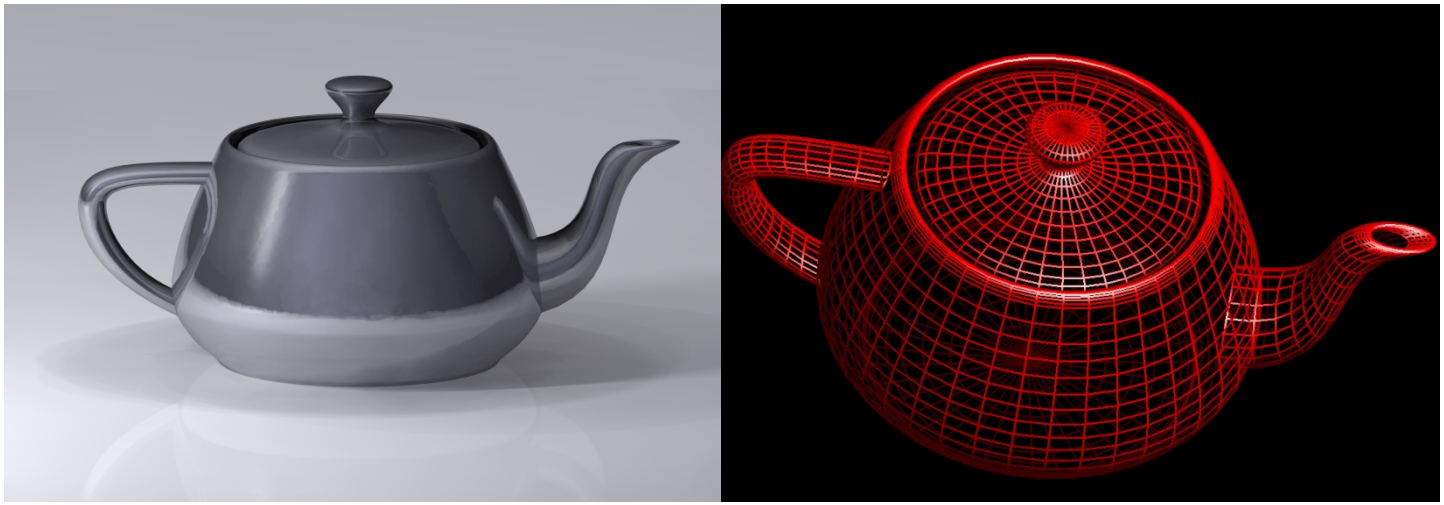
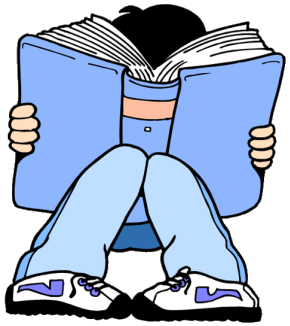


Foundations of Computer Graphics

Saurabh Ray

Reading

Reading for Lecture 8 : Sections 5.1 - 5.7.



We have already talked about hidden surface removal.
This is in Section 5.8.

We will discuss “Culling” later.

Reading for Lecture 9 : Sections 6.1 - 6.8.

Please also read the code to make sure you understand the details.

Code on github: https://github.abudhabi.nyu.edu/sr194/CG-2020

Typical Frames used in Computer Graphics

- Object (or model) coordinates
- World coordinates
- Eye (or camera) coordinates

The transformations from model to world coordinates and from world to eye coordinates are usually combined into a **Model-View matrix**.

- Clip coordinates

4D Homogeneous coordinates

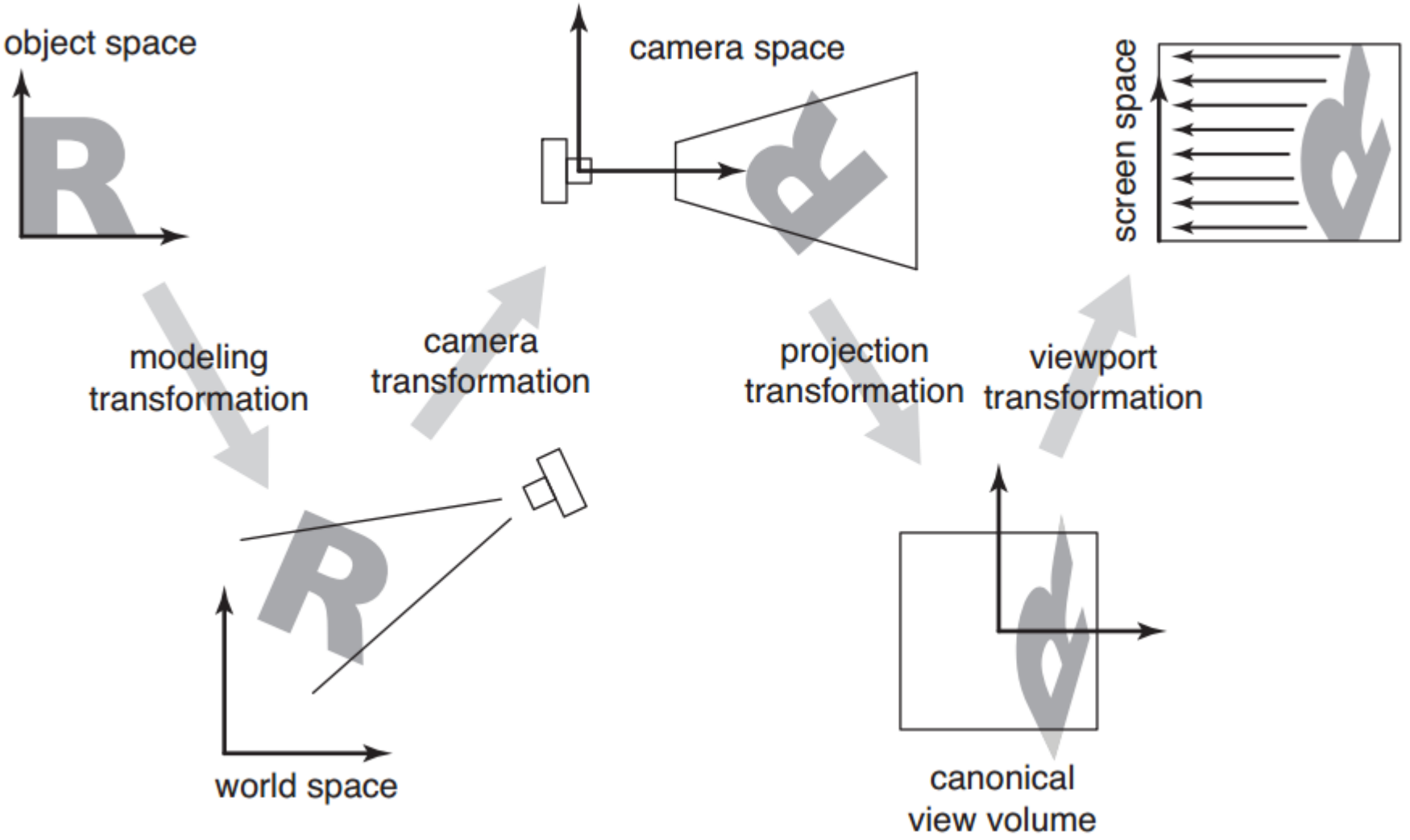
- Normalized device coordinates

3D coordinates

- Window (or screen) coordinates

2D coordinates

Viewing Transformations



Computer Viewing

Three main aspects:

Positioning the camera

- *setting the model-view matrix*

Selecting a lens

- *setting the projection matrix*

Clipping

- *setting the viewing volume*

Camera or Eye Transformation

What do we need to specify our "viewpoint" in 3D?

- Camera or eye position e *location*

- gaze direction $\vec{g}$ *vector*

- view-up vector $\vec{t}$ *vector*

All the above specified in world coordinates.

Camera space coordinate frame:

origin: e

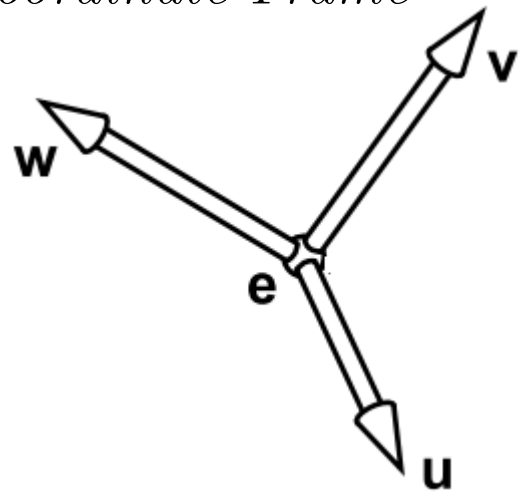
$\vec{w} = -\vec{g}/\|\vec{g}\|$ *unit vector opposite to the gaze direction*

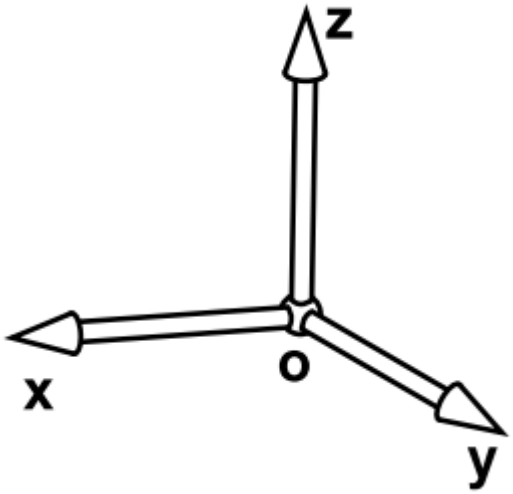
$\vec{v} = \vec{t}/\|\vec{t}\|$ *unit vector in the up direction*

$\vec{u} = \vec{v} \times \vec{w}$ *unit vector to the right of viewer*

Camera or Eye Transformation

Camera Coordinate Frame





World Coordinate Frame

Camera or Eye Transformation

Coordinates in Camera Frame

columns are homogeneous coordinates w.r.t. the world frame

Coordinates in World Frame

$$\boldsymbol{\beta} = \underbrace{\begin{bmatrix} \vec{u} & \vec{v} & \vec{w} & e \end{bmatrix}^{-1}}_{M_{cam}} \boldsymbol{\alpha}$$

Matrix that implements Camera transformation

Rotation Matrices

Consider a 3×3 rotation matrix : $$R = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \\ \vec{a} & \vec{b} & \vec{c} \end{bmatrix}$$

Consider the usual basis vectors:

$$\hat{i} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \qquad \hat{j} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \qquad \hat{k} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

How does R transform $\hat{i}, \hat{j}$ and $\hat{k}$?

$$R\,\hat{i} = \vec{a}, \qquad R\,\hat{j} = \vec{b}, \qquad R\,\hat{k} = \vec{c}$$

Since $\hat{i}, \hat{j}$ and $\hat{k}$ are unit length and orthogonal to each other, $\vec{a}$, $\vec{b}$ and $\vec{c}$ are also unit length and orthogonal to each other.

Rotation Matrices

$$R^T = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix} \begin{matrix} \vec{a} \\ \vec{b} \\ \vec{c} \end{matrix} \qquad R = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} \begin{matrix} \\ \\ \\ \end{matrix} \\ \begin{matrix} \vec{a} & \vec{b} & \vec{c} \end{matrix}$$

$$R^T R = \begin{bmatrix} \vec{a}\cdot\vec{a} & \vec{a}\cdot\vec{b} & \vec{a}\cdot\vec{c} \\ \vec{b}\cdot\vec{a} & \vec{b}\cdot\vec{b} & \vec{b}\cdot\vec{c} \\ \vec{c}\cdot\vec{a} & \vec{c}\cdot\vec{b} & \vec{c}\cdot\vec{c} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

So, $R^{-1} = R^T$

The inverse of a rotation matrix is its transpose.

This is true for any matrix whose columns are unit vectors orthogonal to each other, i.e., the columns form an orthonormal basis.

Inverse of a Homogeneous Transformation matrix

$$\begin{pmatrix} \overset{3\times 3}{\mathbf{L}} & \overset{3\times 1}{\mathbf{t}} \\ 0 \quad 0 \quad 0 & 1 \end{pmatrix} \begin{pmatrix} \overset{3\times 1}{\mathbf{x}} \\ w \end{pmatrix} = \begin{pmatrix} \overset{3\times 1}{\mathbf{y} = \mathbf{Lx} + w\mathbf{t}} \\ w \end{pmatrix}$$

$$y = L\mathbf{x} + w\mathbf{t}$$

$$\implies L^{-1}\mathbf{y} = \mathbf{x} + L^{-1}w\mathbf{t}$$

$$\implies \mathbf{x} = L^{-1}\mathbf{y} + w(-L^{-1}\mathbf{t})$$

So, the inverse transformation matrix is:

$$\begin{pmatrix} \overset{3\times 3}{\mathbf{L}^{-1}} & \overset{3\times 1}{\mathbf{-L}^{-1}\mathbf{t}} \\ 0 \quad 0 \quad 0 & 1 \end{pmatrix}$$

Computing Camera Transformation Matrix

$$M_{cam} = \begin{bmatrix} \vec{u} & \vec{v} & \vec{w} & e \end{bmatrix}^{-1}$$

If we treat $\vec{u}$, $\vec{v}$ and $\vec{w}$ and 3D vectors, and let $\vec{e}$ be the 3D position vector of the eye, then

$$M_{cam} = \begin{pmatrix} \vec{u} & \vec{v} & \vec{w} & \vec{e} \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} \vec{u} & -\vec{u}\cdot\vec{e} \\ \vec{v} & -\vec{v}\cdot\vec{e} \\ \vec{w} & -\vec{w}\cdot\vec{e} \\ 0\ 0\ 0 & 1 \end{pmatrix}$$

LookAt function

A popular API: `LookAt(eye,at,up)`

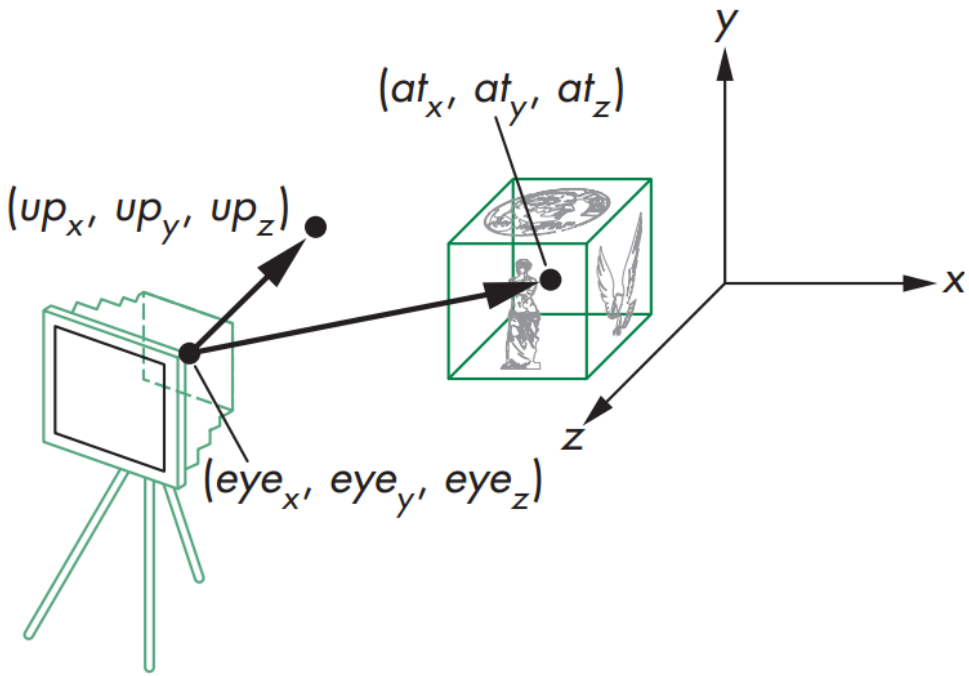
Camera Frame:

$e = eye$

$\vec{w} = -(at - eye)/\|at - eye\|$

$\vec{u} = \vec{up} \times \vec{w}/\|\vec{up} \times \vec{w}\|$

$\vec{v} = \vec{w} \times \vec{u}$



$$M_{cam} = \begin{pmatrix} \vec{u} & -\vec{u}\cdot\vec{e} \\ \vec{v} & -\vec{v}\cdot\vec{e} \\ \vec{w} & -\vec{w}\cdot\vec{e} \\ 0\ \ 0\ \ 0 & 1 \end{pmatrix}$$

LookAt function

Implementation in `MV.js`. Uses different variable names.

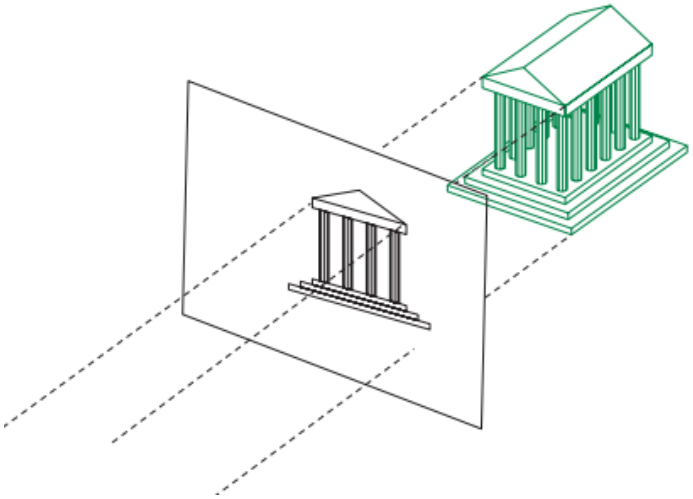
```
function lookAt( eye, at, up )
{
    if ( !Array.isArray(eye) || eye.length != 3) {
        throw "lookAt(): first parameter [eye] must be an a vec3";
    }

    if ( !Array.isArray(at) || at.length != 3) {
        throw "lookAt(): first parameter [at] must be an a vec3";
    }

    if ( !Array.isArray(up) || up.length != 3) {
        throw "lookAt(): first parameter [up] must be an a vec3";
    }

    if ( equal(eye, at) ) {
        return mat4();
    }
```

LookAt function

```
    var v = normalize( subtract(at, eye) );  // view direction vector
    var n = normalize( cross(v, up) );       // perpendicular vector
    var u = normalize( cross(n, v) );        // "new" up vector

    v = negate( v );

    var result = mat4(
        vec4( n, -dot(n, eye) ),
        vec4( u, -dot(u, eye) ),
        vec4( v, -dot(v, eye) ),
        vec4()
    );

    return result;
}
```

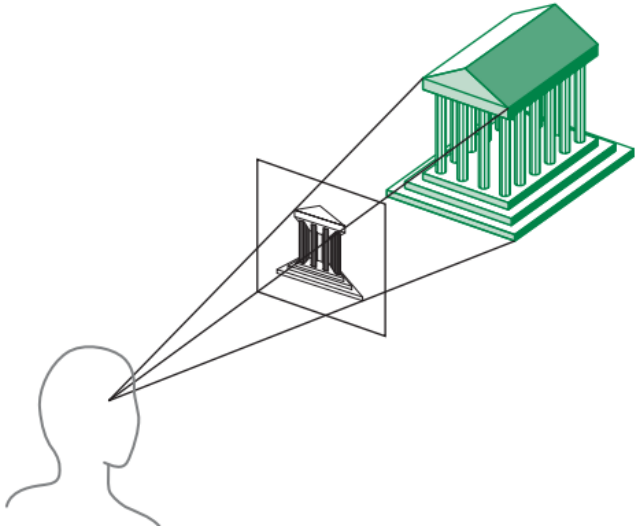
Projection

Camera Space $\rightarrow$ *2D*



Orthographic Projection

projectors are parallel

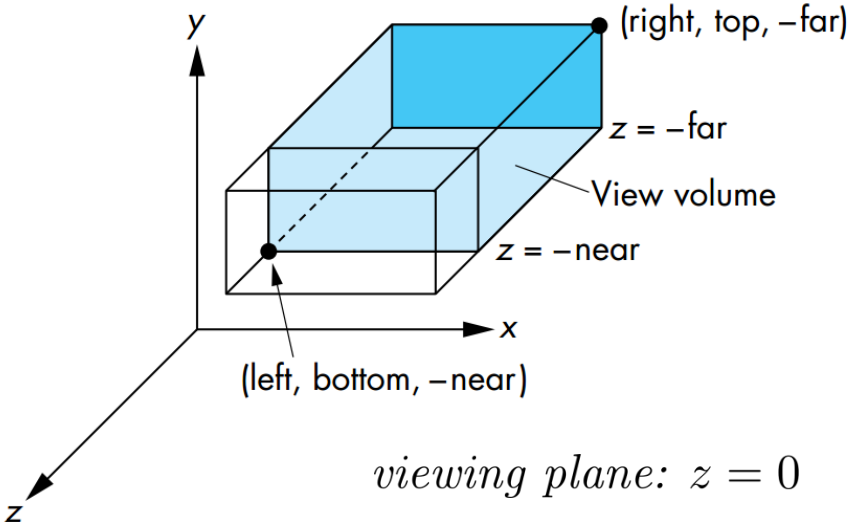


Perspective Projection

projectors meet at a point

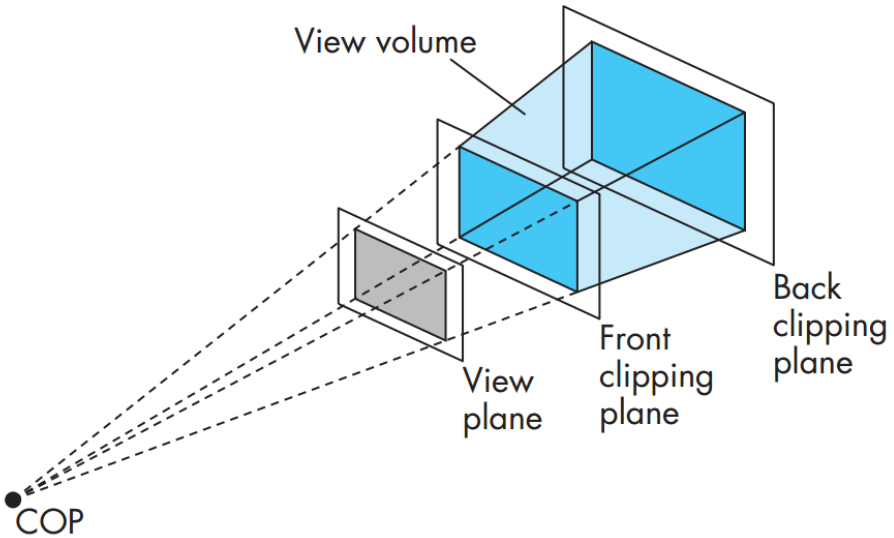
Viewing

Orthographic:



viewing plane: $z = 0$

Perspective:

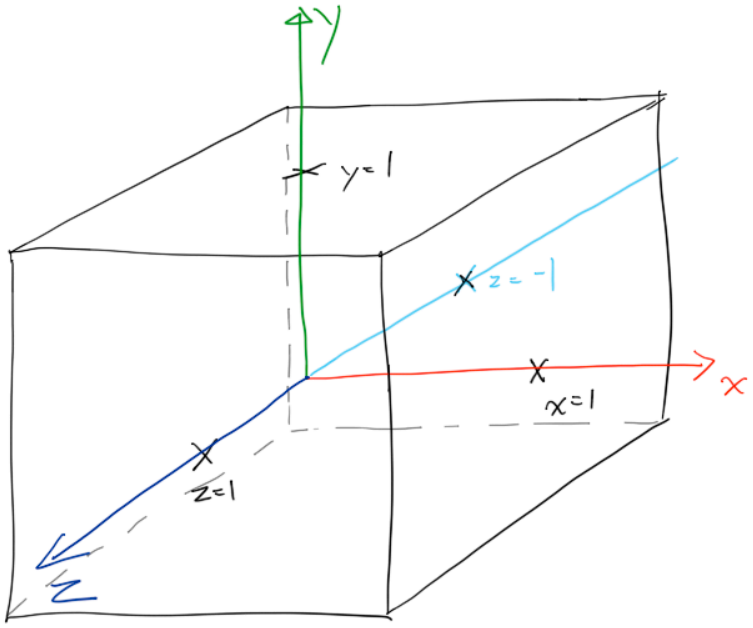


In both cases there is a notion of a view volume

Canonical view volume

The *canonical view volume* in WebGL is the cube with side length 2 centered at the origin.

$$[-1, 1] \times [-1, 1] \times [-1, 1]$$



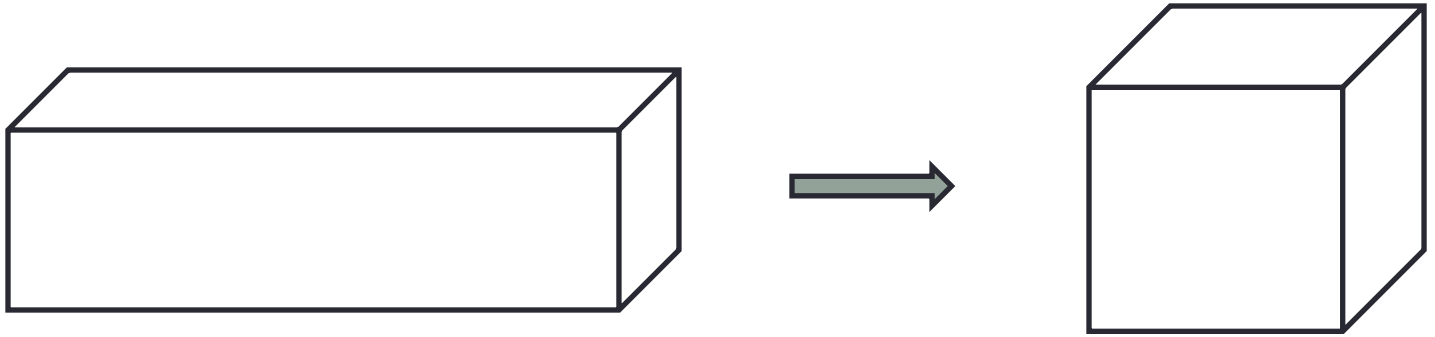
- Everything outside the canonical view volume is clipped.
- Everything inside the canonical view volume is **projected orthographically**.

Projection Normalization

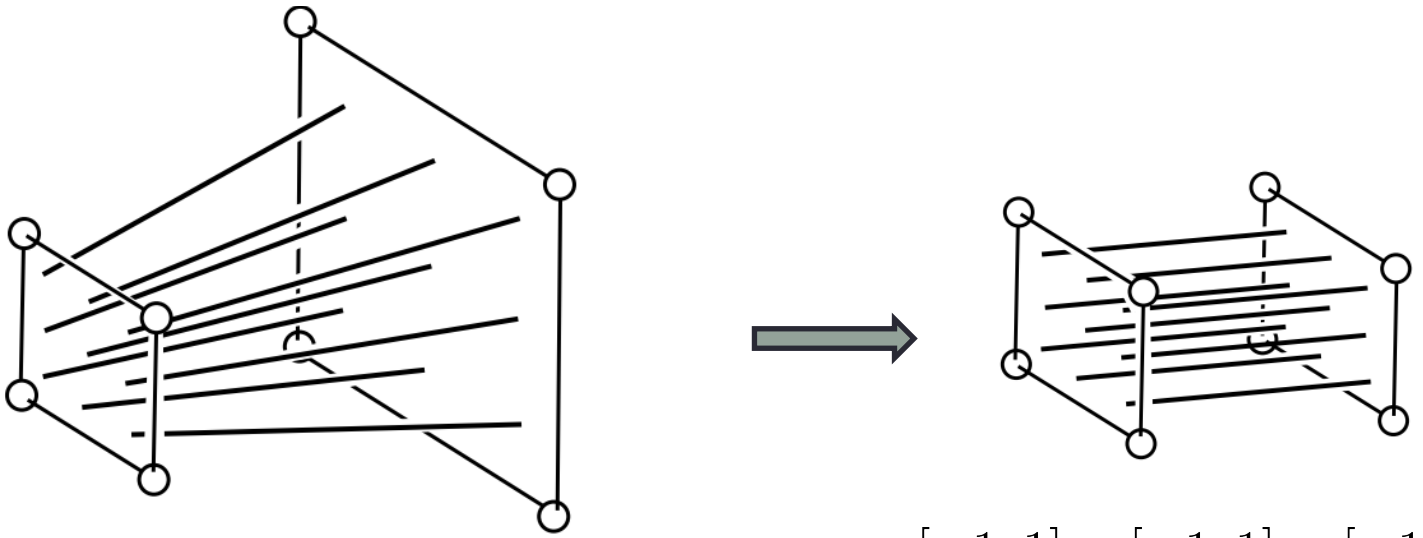
Need to map the intended view volume to canonical volume

Intended view volume *Canonical view volume*

Orthographic



Perspective

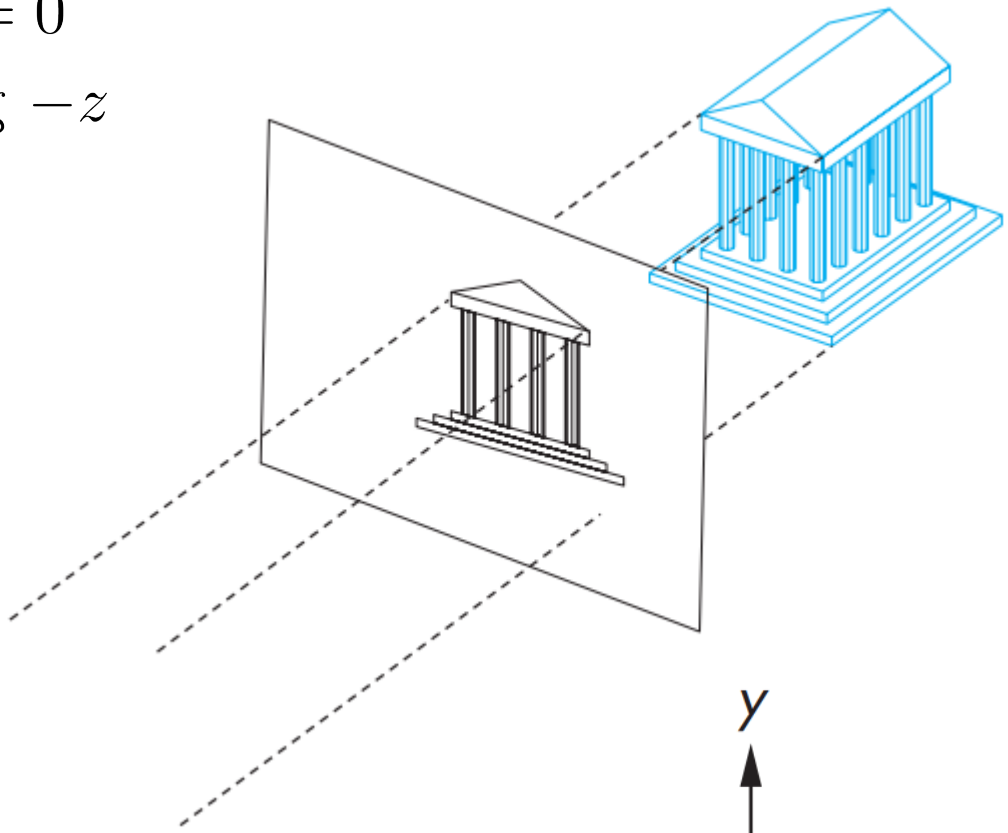


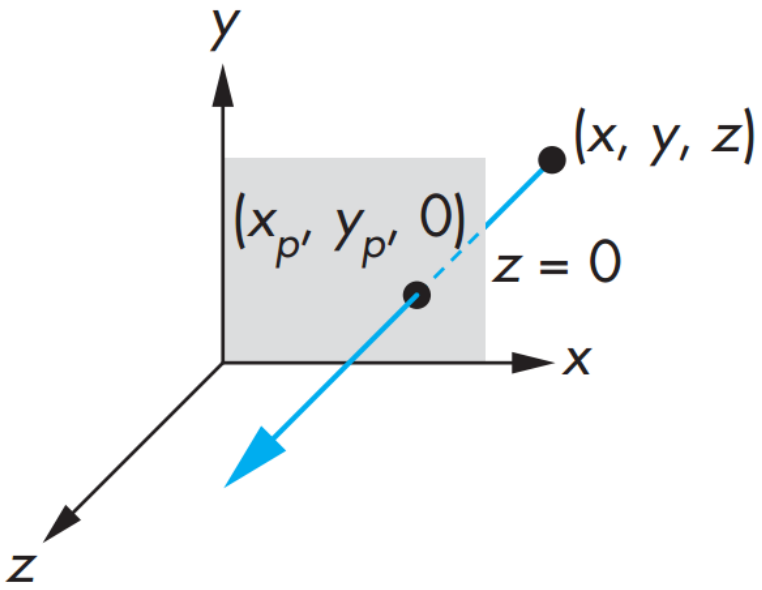
$[-1,1] \times [-1,1] \times [-1,1]$

Orthographic Projection Transformation

Projection plane: $z = 0$

Gaze direction: along $-z$





(x, y, z) is mapped to (x_p, y_p, z_p).

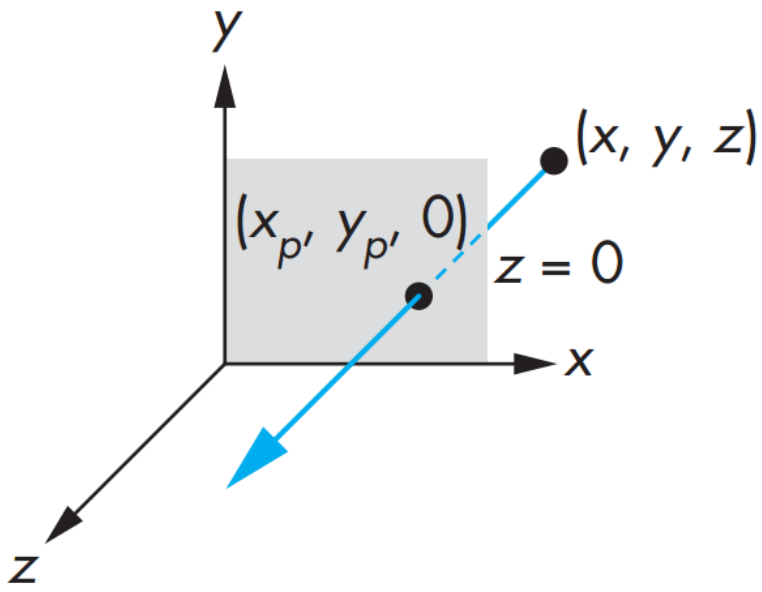
$x_p = x, y_p = y, z_p = 0$

Orthographic Projection Transformation

(x, y, z) is mapped to (x_p, y_p, z_p).

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

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

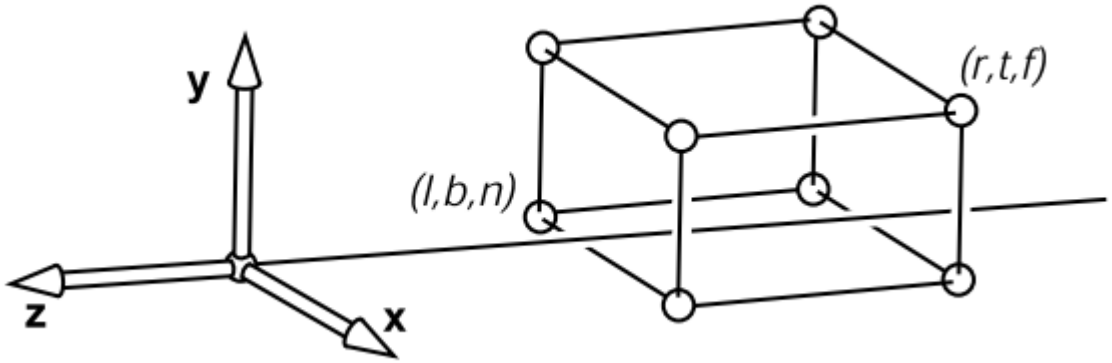


This is done in hardware after everything outside the cube $[-1, 1] \times [-1, 1] \times [-1, 1]$ (canonical view volume) is clipped.

Need to map the intended view volume to the canonical view volume.

Projection Normalization

Need to map the intended view volume to the canonical view volume.



$x = l \equiv$ left plane,

$x = r \equiv$ right plane,

$y = b \equiv$ bottom plane,

$y = t \equiv$ top plane,

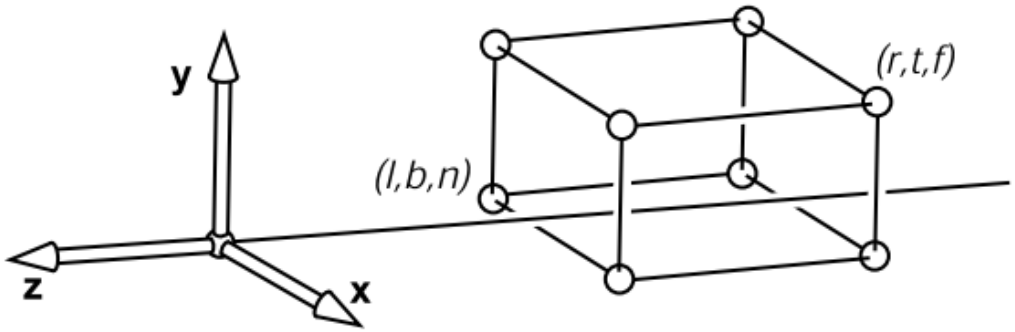
$z = n \equiv$ near plane,

$z = f \equiv$ far plane.

Note: $n > f$

We want to map this to the canonical view volume: $[-1, 1] \times [-1, 1] \times [-1, 1]$

Orthographic Projection Transformation



$\Longrightarrow [-1,1] \times [-1,1] \times [-1,1]$

Initial center of the cube: $(\frac{r+l}{2}, \frac{t+b}{2}, \frac{n+f}{2})$.

Scaling along x : $\frac{2}{r-l}$ Scaling along y : $\frac{2}{t-b}$ Scaling along z : $\frac{2}{n-f}$

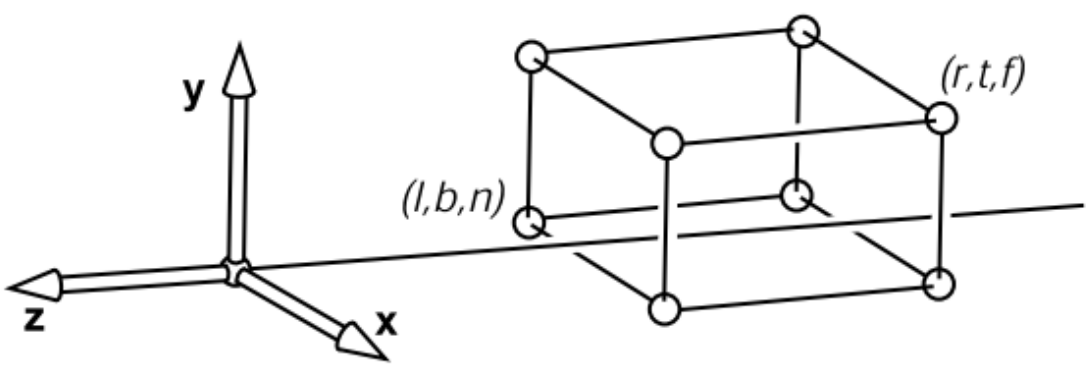
After scaling the center moves to: $(\frac{r+l}{r-l}, \frac{t+b}{t-b}, \frac{n+f}{n-f})$.

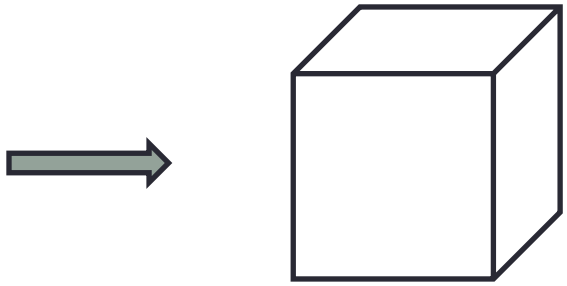
We need to move it to $(0,0,0)$

$$M_{orth} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Projection Normalization

Orthographic projection:

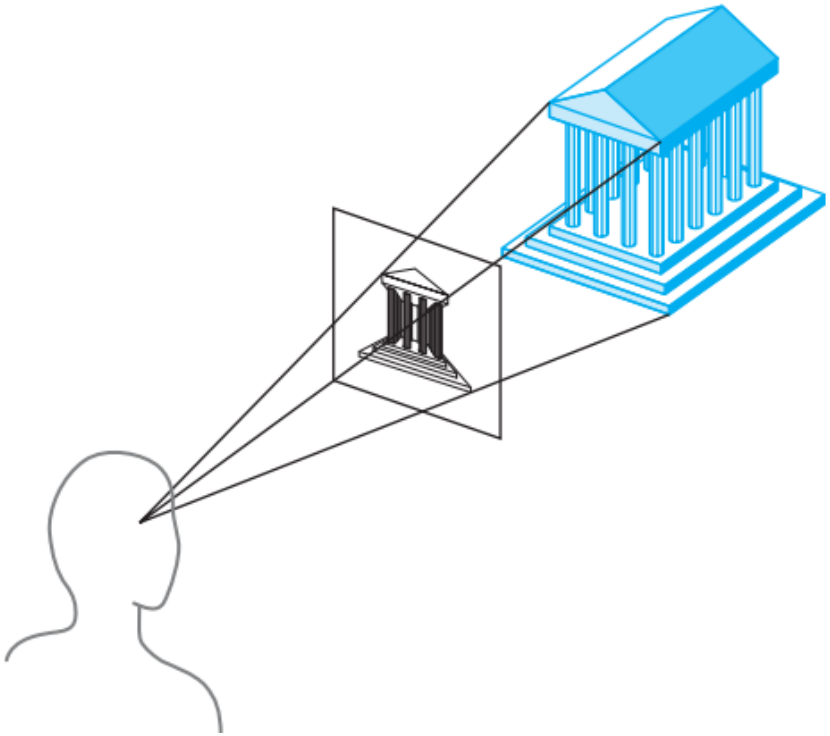




$$[-1,1] \times [-1,1] \times [-1,1]$$

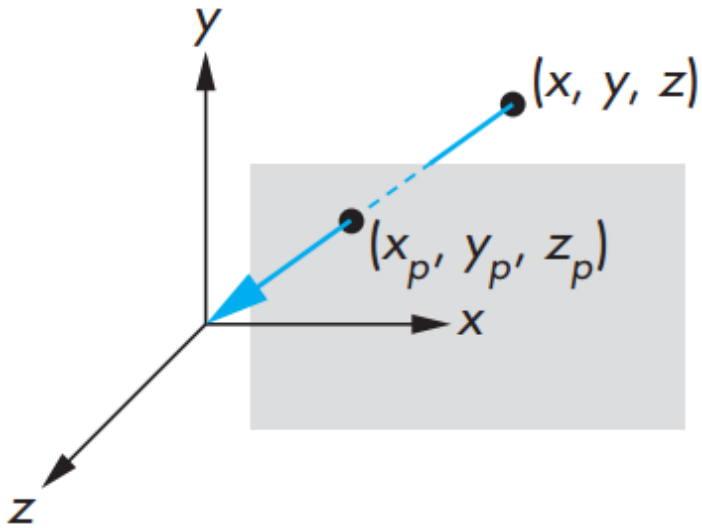
$$M_{orth} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Perspective Projection Transformation



projectors converge at the center of projection: eye/camera location

Projection plane: $z = d \quad (d < 0)$



Need to figure out: x_p, y_p and z_p

Points on the ray emanating from the origin and passing through (x, y, z) are of the form $(\alpha x, \alpha y, \alpha z)$ for some $\alpha \geq 0$.

This ray intersects the plane $z = d$ at $\alpha = d/z$.

So, $(x_p, y_p, z_p) = (\frac{x}{z/d}, \frac{y}{z/d}, \frac{z}{z/d})$

Simple Perspective Projection

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

Is there a matrix M s.t. $M \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \\ 1 \end{bmatrix}$?

No! - division by z is non-linear.

Called **non-uniform foreshortening**

images of objects farther is reduced in size compared to images of objects closer to the center of projection

Simple Perspective Projection

Solution: extend our use of homogeneous coordinates

Earlier we represented a point (x, y, z) by $(x, y, z, 1)$ in homogeneous coordinates.

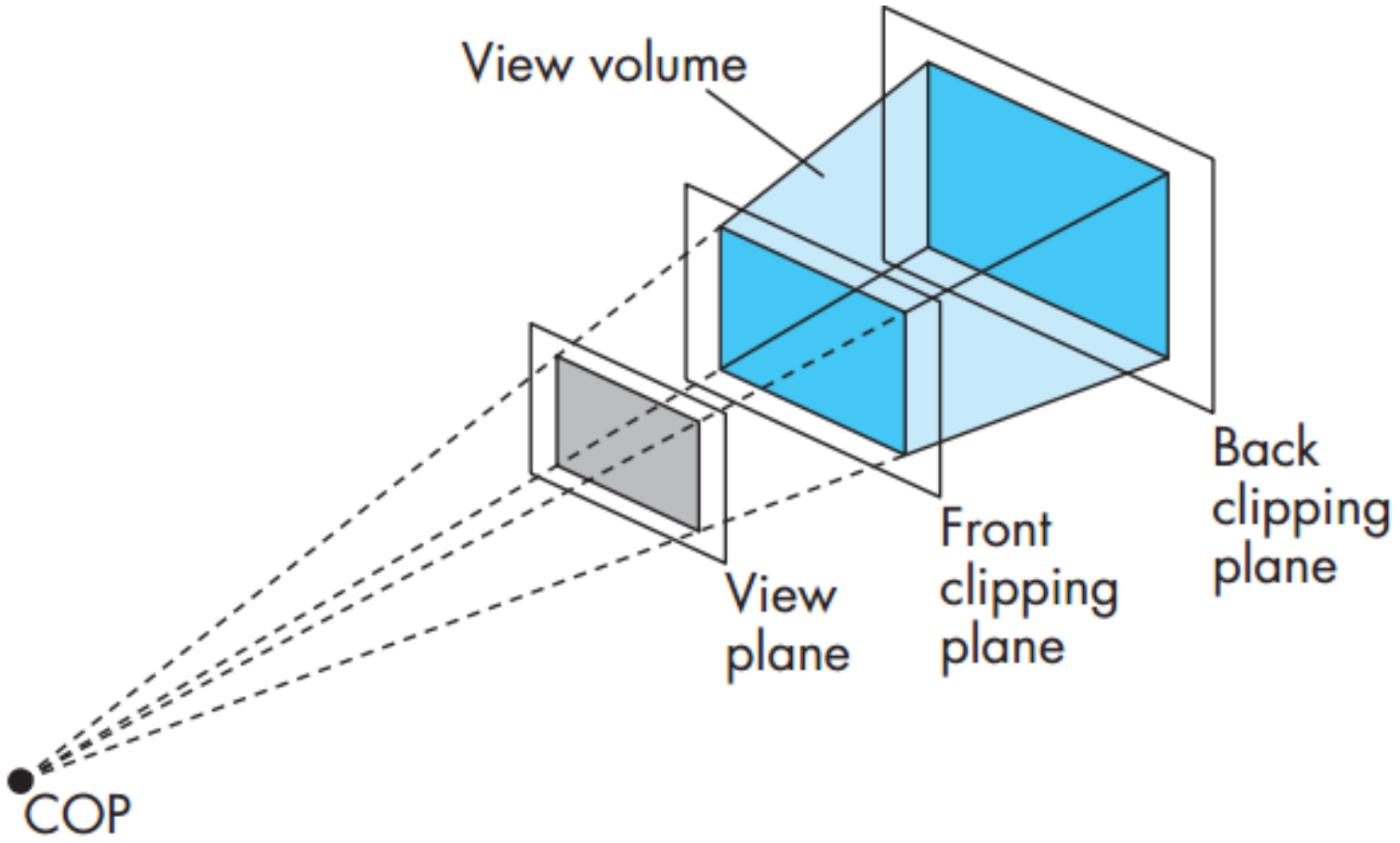
Now we allow it to be represented by any vector (wx, wy, wz, w), where $w \neq 0$.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

represents the point $(\frac{x}{z/d}, \frac{y}{z/d}, d)$

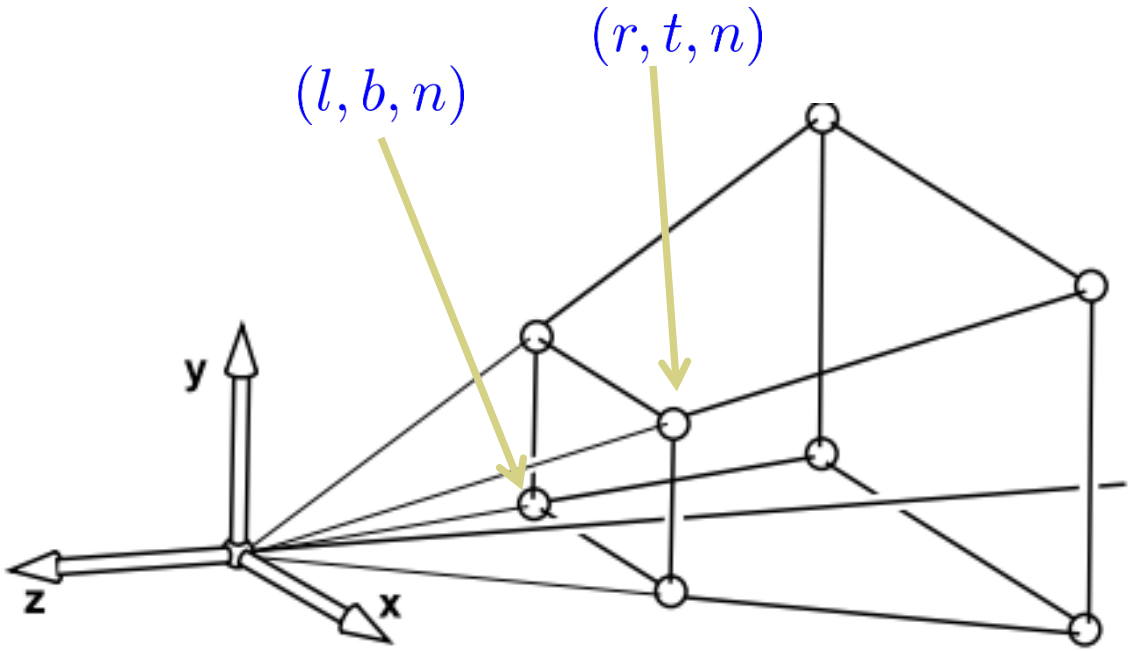
Dividing by the fourth coordinate in the end is called **perspective division**.

Perspective Projection Normalization



The viewing volume is a Frustum - a truncated pyramid.

Specifying the Frustum



To specify the frustum we specify:

- *the near and the far planes*
- *the rectangle in which the frustum intersects the near plane*

The six numbers l, r, t, b, n and f specify the frustum.

Perspective Projection Normalization

Lets use the near plane ($z = n$) as the projection plane.

From simple projective transformation, our matrix is:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/n & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ z/n \end{bmatrix}$$

represents the point:

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

If we only care about the x and y coordinates, we can use a more general matrix too:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & 1/n & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ \alpha z + \beta \\ z/n \end{bmatrix}$$

represents the point:

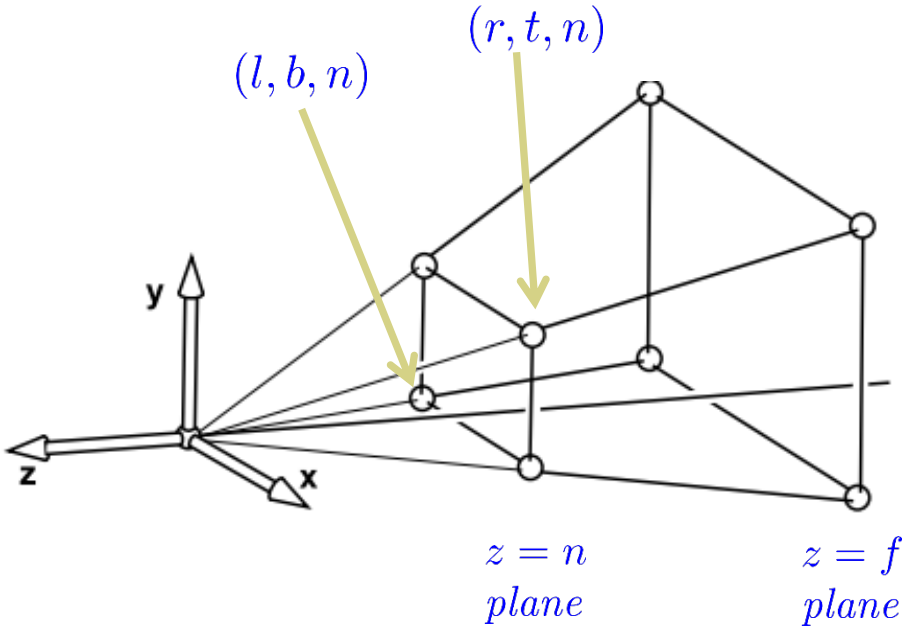
$$\left(\frac{x}{z/n}, \frac{y}{z/n}, \alpha n + \frac{\beta n}{z}\right)$$

This retains information about the z-coordinate. Useful for depth testing.

Perspective Projection Normalization

(x, y, z)

$$\Big(\frac{x}{z/n}, \frac{y}{z/n}, \alpha n + \frac{\beta n}{z}\Big)$$



We choose α and β in such a way that the near and far planes stay where they are.

$$\Longrightarrow \quad \alpha n + \frac{\beta n}{n} = n \quad \text{and} \quad \alpha n + \frac{\beta n}{f} = f$$

$$\Longrightarrow \quad \alpha = 1 + \frac{f}{n} \quad \text{and} \quad \beta = -f$$

Perspective Projection Normalization

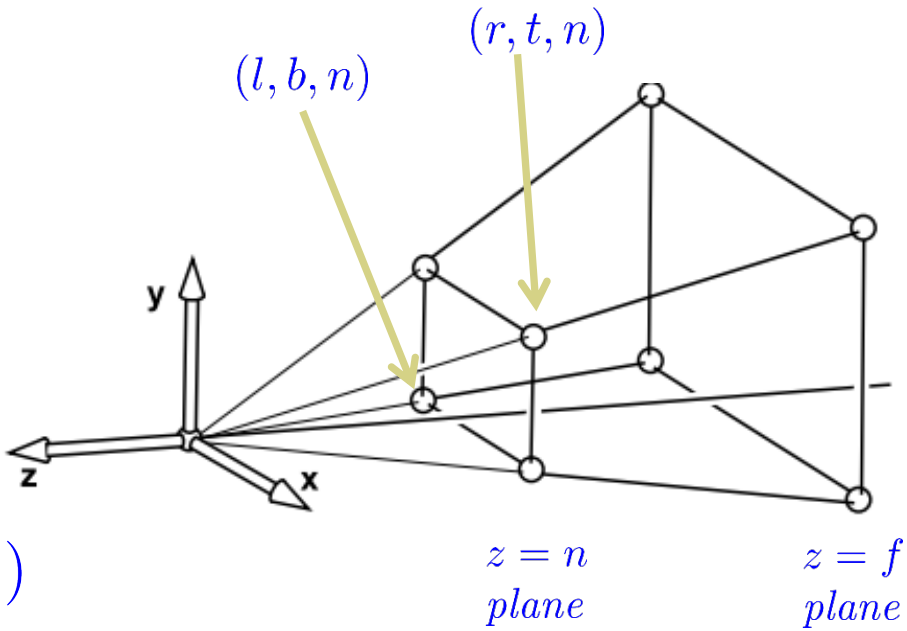
(x, y, z)

$\Downarrow$

$(\frac{x}{z/n}, \frac{y}{z/n}, \alpha n + \frac{\beta n}{z})$

$= (\frac{x}{z/n}, \frac{y}{z/n}, (1 + \frac{f}{n})n + \frac{-fn}{z})$

$= (\frac{x}{z/n}, \frac{y}{z/n}, n + f - \frac{fn}{z})$



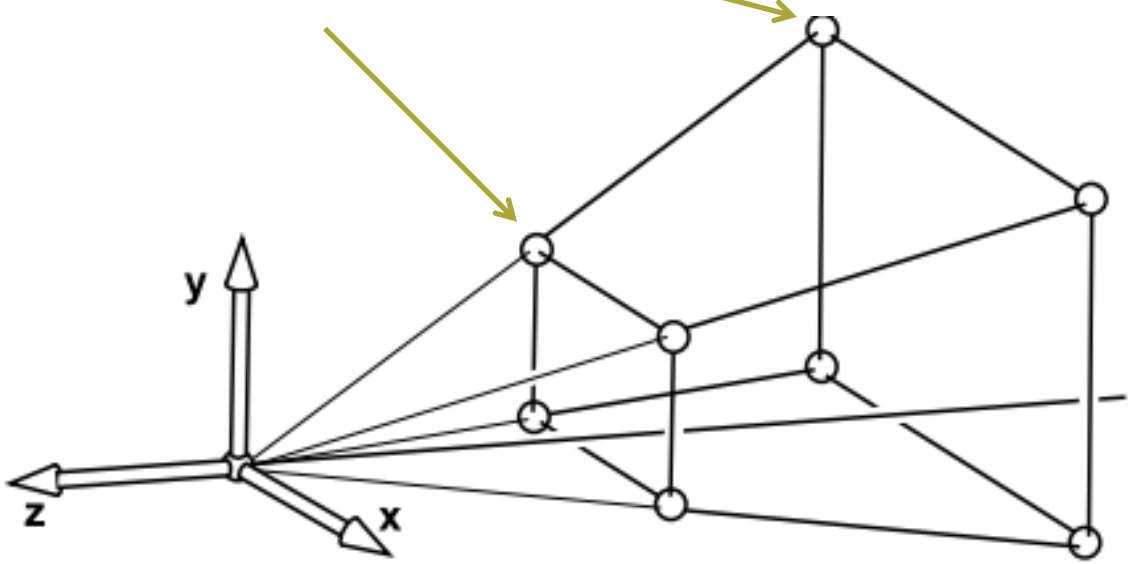
The matrix for doing this transformation is:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & 1/n & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 + \frac{f}{n} & -f \\ 0 & 0 & 1/n & 0 \end{bmatrix} = P$$

Perspective Projection Normalization

After the transformation:

x and y coordinates of these two points are the same.

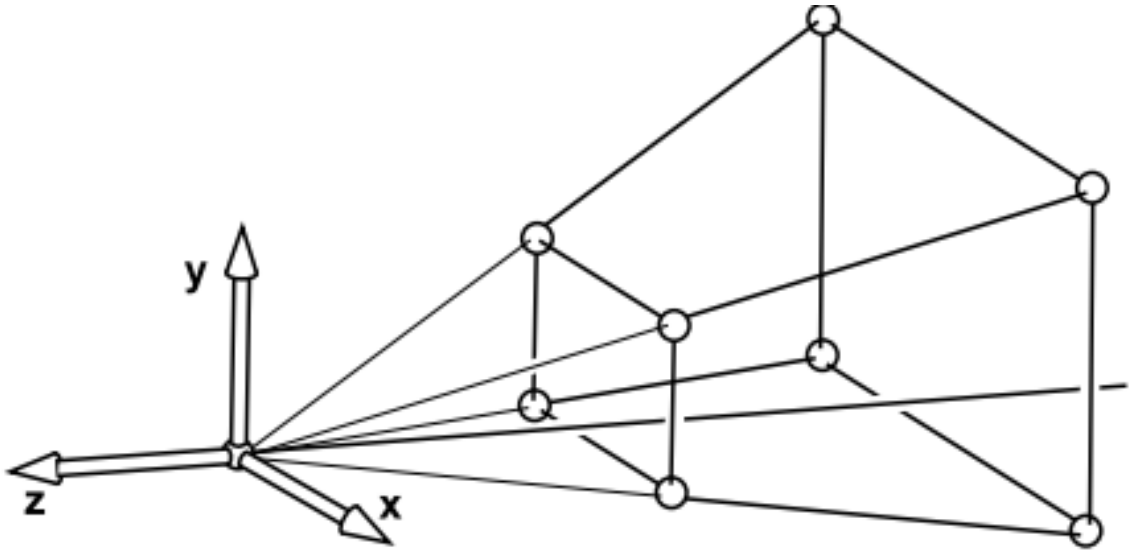


$z = n$ *plane*

$z = f$ *plane*

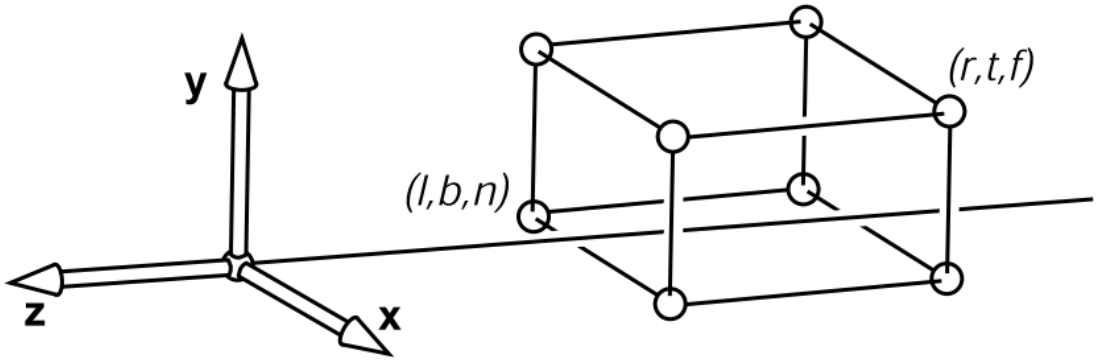
The coordinates of the points in the near plane don't change.

Perspective Projection Normalization



$z = n$ $z = f$

plane *plane*



Perspective Projection Normalization

$$(x, y, z) \quad \Longrightarrow \quad \left(\frac{x}{z/n}, \frac{y}{z/n}, n + f - \frac{fn}{z}\right)$$

The z-coordinates of points in the near and far planes don't change.

The z-coordinates of the other points do change.

Consider two points (x_1, y_1, z_1) and (x_2, y_2, z_2) with $z_1 > z_2$

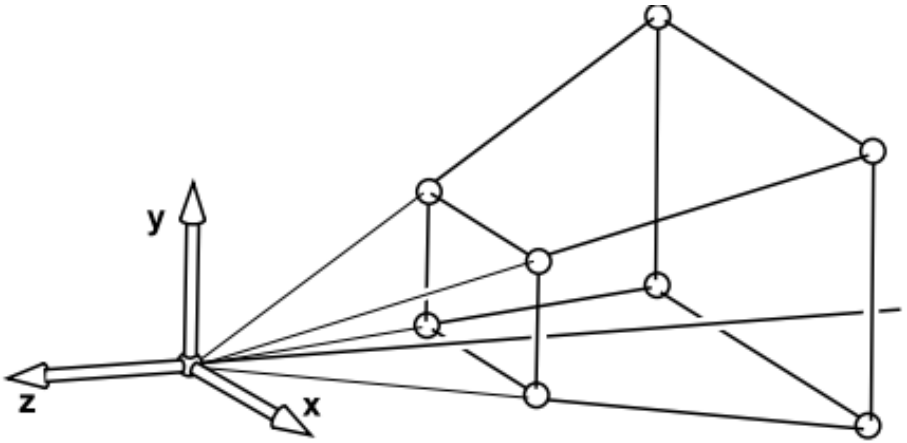
$$z_1' = n + f - \frac{fn}{z_1} \qquad z_2' = n + f - \frac{fn}{z_2}$$

$$\Longrightarrow \; z_1' - z_2' = \frac{fn}{z_2} - \frac{fn}{z_1} = \frac{fn(z_1 - z_2)}{z_1 z_2} > 0$$

$$z_1 > z_2 \iff z_1' > z_2'$$

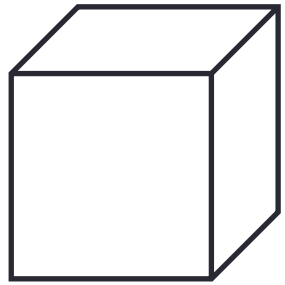
This means that the order of points along z-axis does not change. So, we can still use it for depth testing.

Perspective Projection Normalization

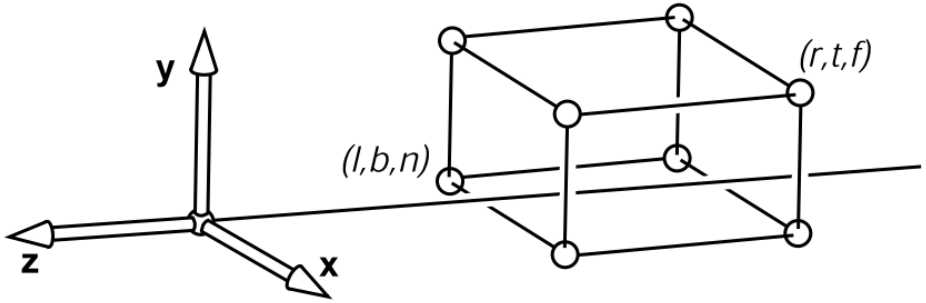


$$M_{per} = M_{orth} P$$

P



$[-1,1] \times [-1,1] \times [-1,1]$



M_{orth}

Final Perspective Matrix

$$M_{per} = M_{orth} P$$

$$= \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1+\frac{f}{n} & -f \\ 0 & 0 & 1/n & 0 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{2}{r-l} & 0 & -\frac{r+l}{n(r-l)} & 0 \\ 0 & \frac{2}{t-b} & -\frac{t+b}{n(t-b)} & 0 \\ 0 & 0 & \frac{n+f}{n(n-f)} & -\frac{2f}{n-f} \\ 0 & 0 & 1/n & 0 \end{bmatrix}$$

Final Perspective Matrix

Sometimes, every entry is multiplied by n.

$$M_{per} = \begin{bmatrix} \frac{2n}{r-l} & 0 & -\frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & -\frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{n+f}{n-f} & -\frac{2fn}{n-f} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Its the same because there is perspective division at the end.

Perspective Matrix

Perspective Matrix:

$$M_{per} = \begin{bmatrix} \frac{2n}{r-l} & 0 & -\frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & -\frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{n+f}{n-f} & -\frac{2fn}{n-f} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = M_{per} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Observe that: $w' = z$

For points in front of camera: $z < 0$.

So, for those points: $w' < 0$

Problem: WebGL clips out points with negative w-coordinate!

Perspective Normalization

Solution: Multiply every entry in the matrix by -1.

Due to perspective divide this does not change transformation but makes the w-coordinate of points in front of the camera positive.

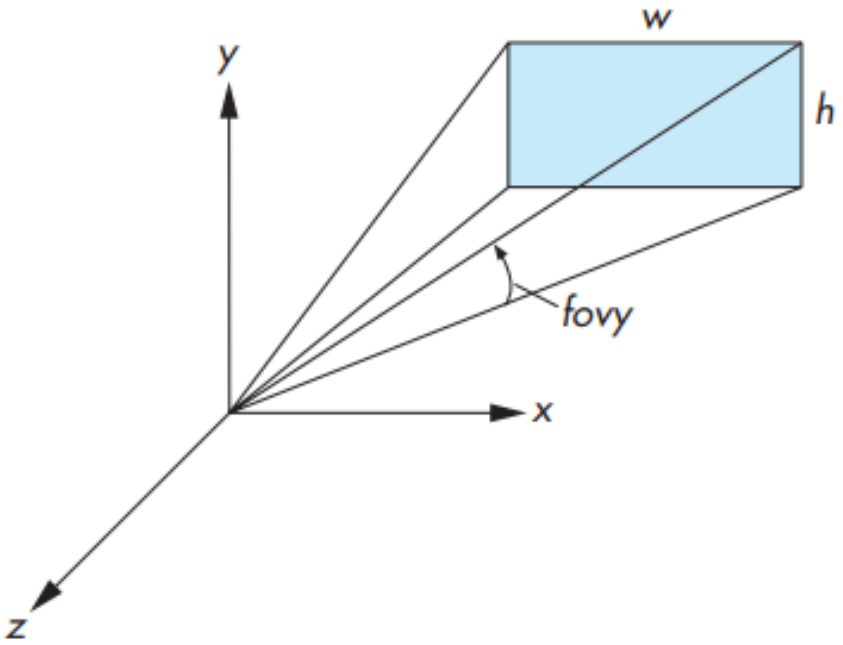
Perspective Matrix
to be used in WebGL coding:

$$M_{per} = \begin{bmatrix} -\frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & -\frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{n+f}{n-f} & \frac{2fn}{n-f} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Note: Sometimes n and f are treated as distances to the near and far planes respectively. In that case the planes are $z = -n$ and $z = -f$.

Specifying a symmetric frustum

Specified using: `fovy` angle, `aspect` $= w/h$, `near` and `far`.



The frustum is symmetric around the $-z$ axis.

fovy = angle between the top and bottom planes

(not between the lines)

`near` *and* `far` *are distances to the near and far planes.*

Conversion to previous parameters:

$n = -$`near` $\quad t =$ `near` $tan(fovy/2) \quad r = t\cdot$`aspect`

$f = -$`far` $\quad b = -t \quad l = -r$