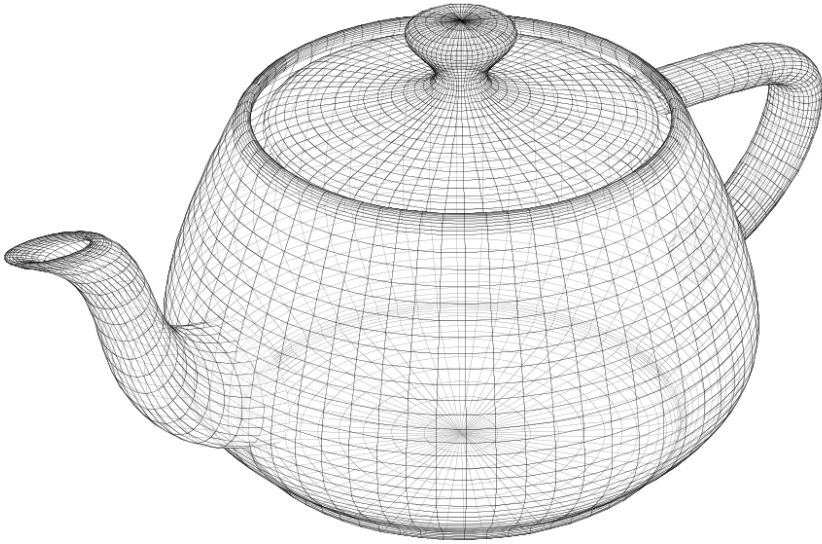


The View Transformation

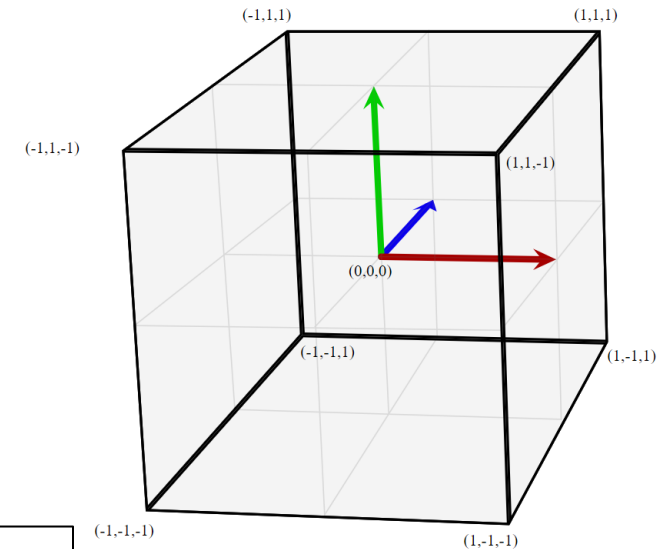


CS 418: Interactive Computer Graphics
Professor Eric Shaffer

WebGL and Camera Views

WebGL renders geometry in a $2 \times 2 \times 2$ axis aligned box centered at the origin

- This is like having a camera in a fixed position
- So...how can we render arbitrary views of our geometry?

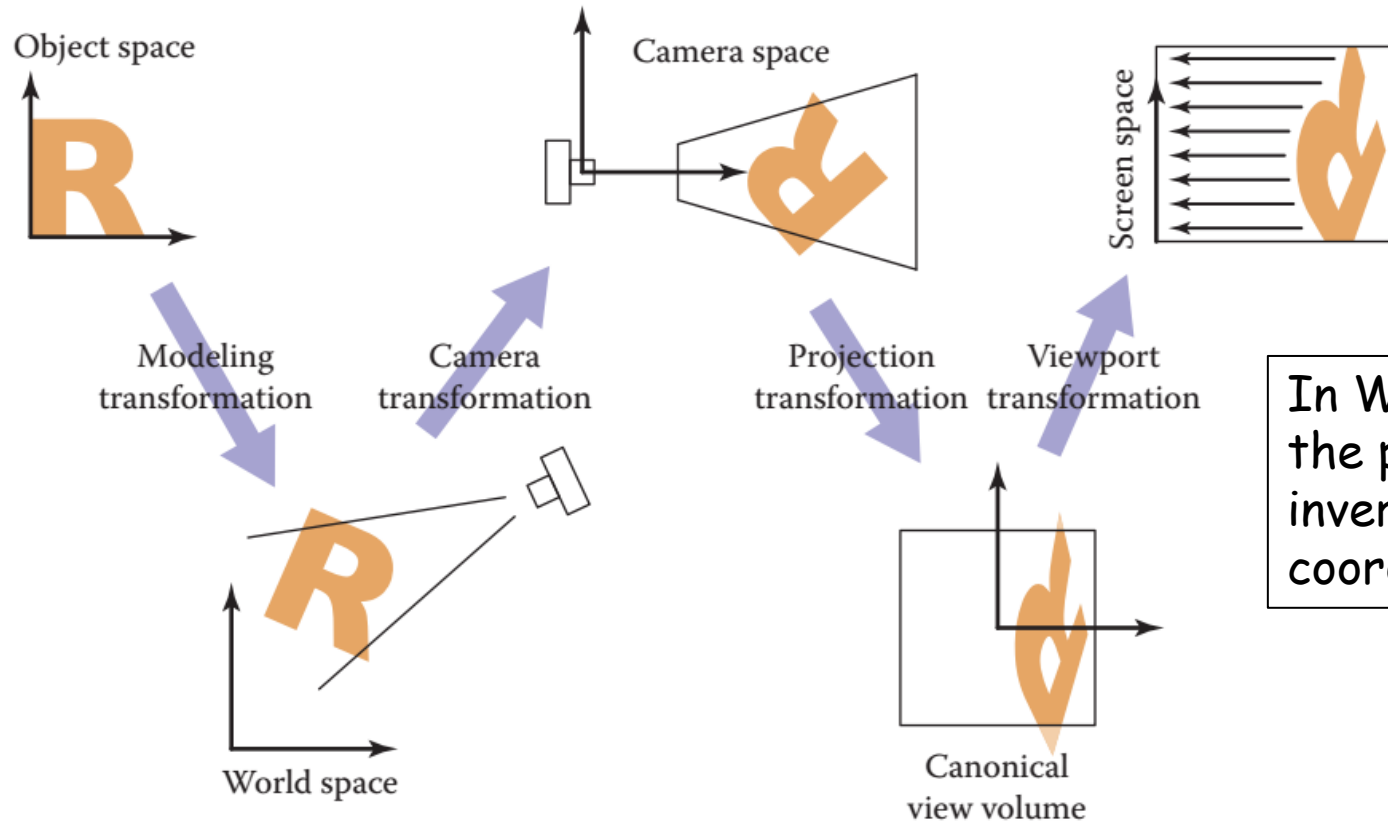


Bonus Questions:

What is the "handedness" of WebGL clip space?

What is the handedness of the world coordinate system used by WebGL systems?

Graphics Pipeline

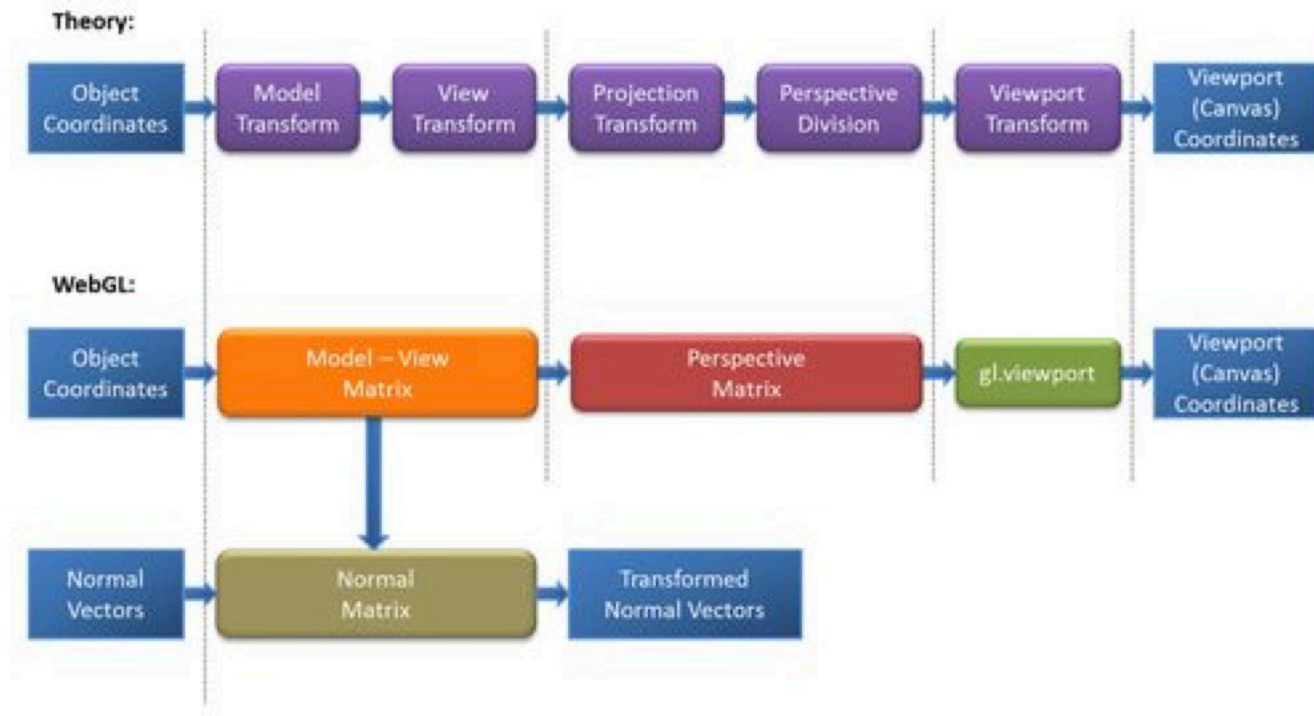


We will call the **camera transformation** the **view transformation**

In WebGL convention, the projection transformation inverts the handedness of the coordinate system

The canonical view volume is a 2x2x2 box centered at the origin with coordinates ranging from $[-1, -1, -1]$ to $[1, 1, 1]$

Graphics Pipeline and WebGL

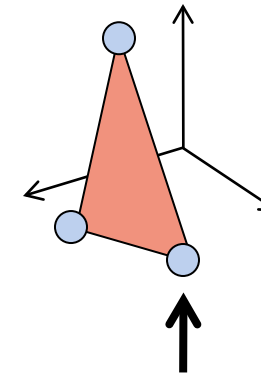
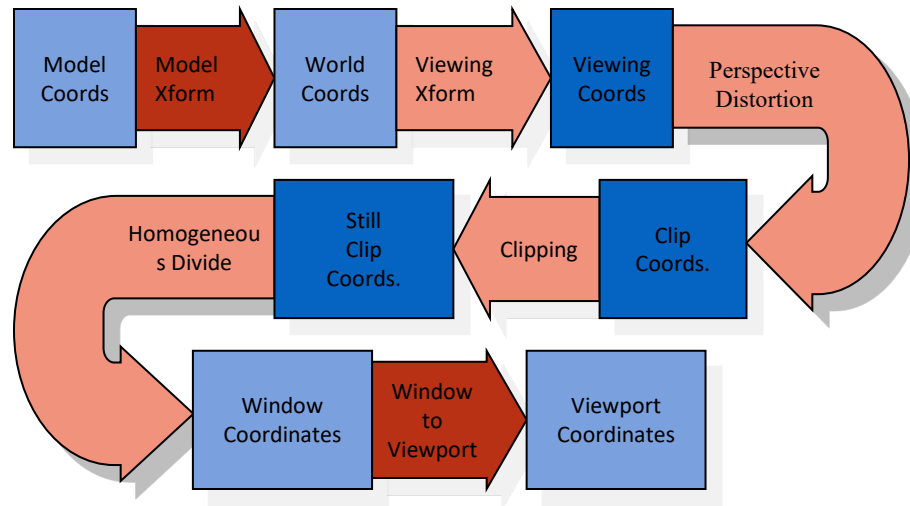


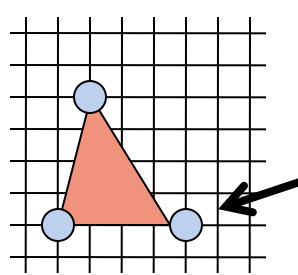
From WebGL Beginner's Guide by Cantor and Jones

What boxes in the theory diagram does this code correspond to?

```
gl_Position = uPMatrix*uMVMMatrix*vec4(aVertexPosition, 1.0);
```

Graphics Pipeline





A 2D grid with a red triangle and its vertices marked by blue dots. An arrow points from the equation below towards this triangle.

$$\begin{bmatrix} x_s \\ y_s \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \text{W2V} \\ \text{Persp} \\ \text{View} \\ \text{Model} \end{bmatrix} \begin{bmatrix} x_m \\ y_m \\ z_m \\ 1 \end{bmatrix}$$

Viewing

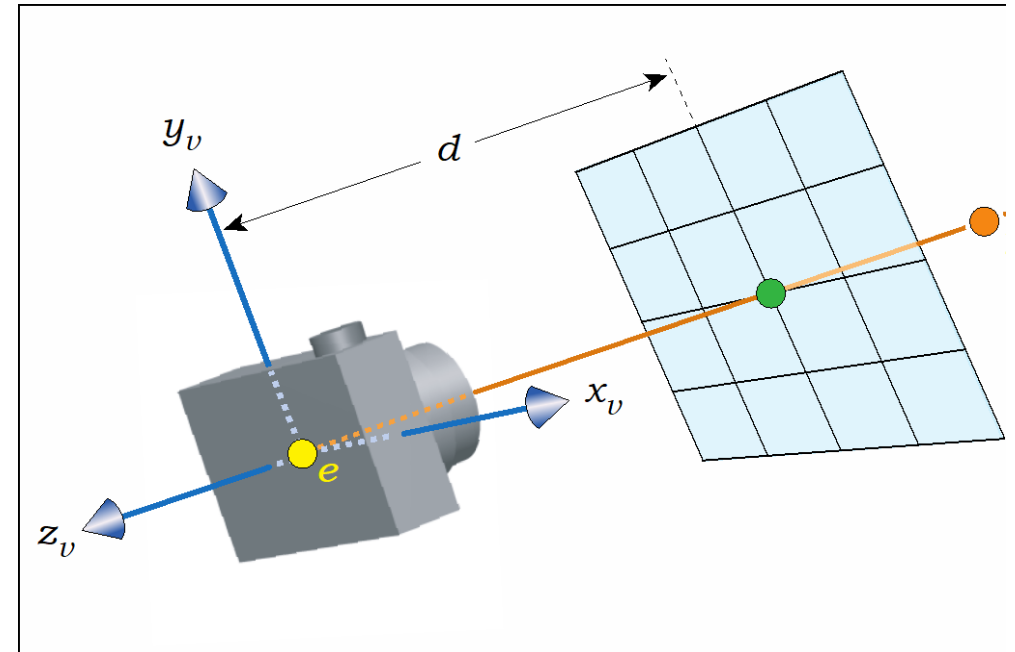
We often will want to allow the view of our 3D scene to change

We can do so using by applying affine transformations to the geometry

A *view matrix* is functionally equivalent to setting up a camera location in world space.

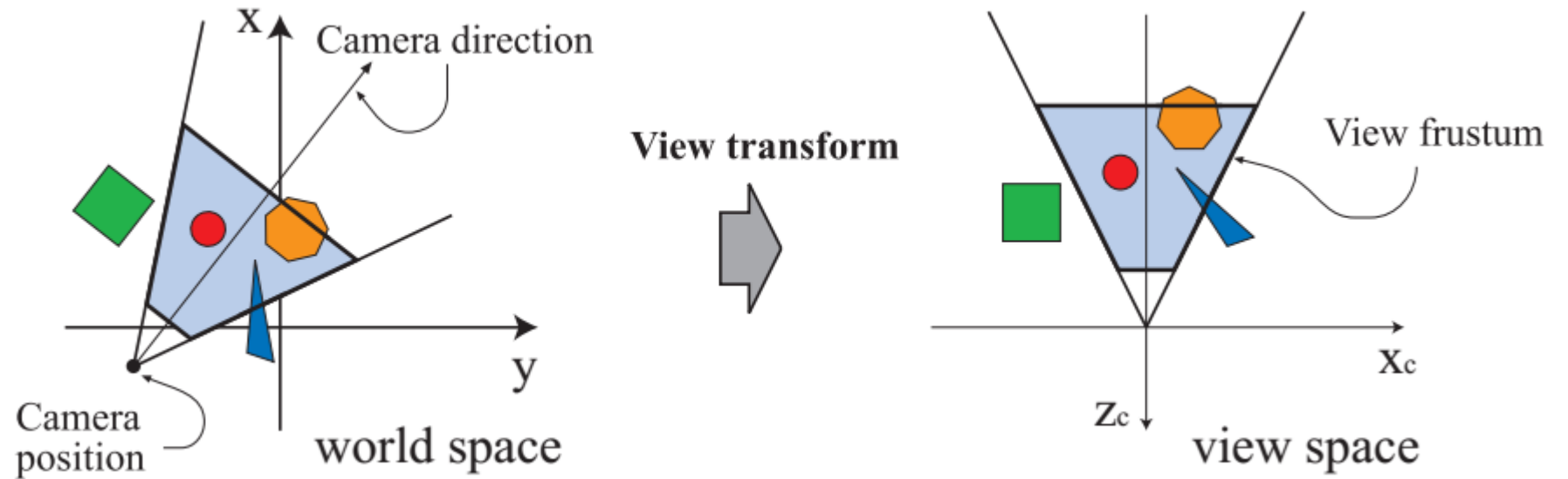
It is a transformation matrix like the Model matrix, **but**

- ❑ Happens after the modeling transformation
- ❑ It applies the same transformations equally to every object



Viewing

The engines don't move the ship at all.
The ship stays where it is and the engines move the universe around it.
-- *Futurama*

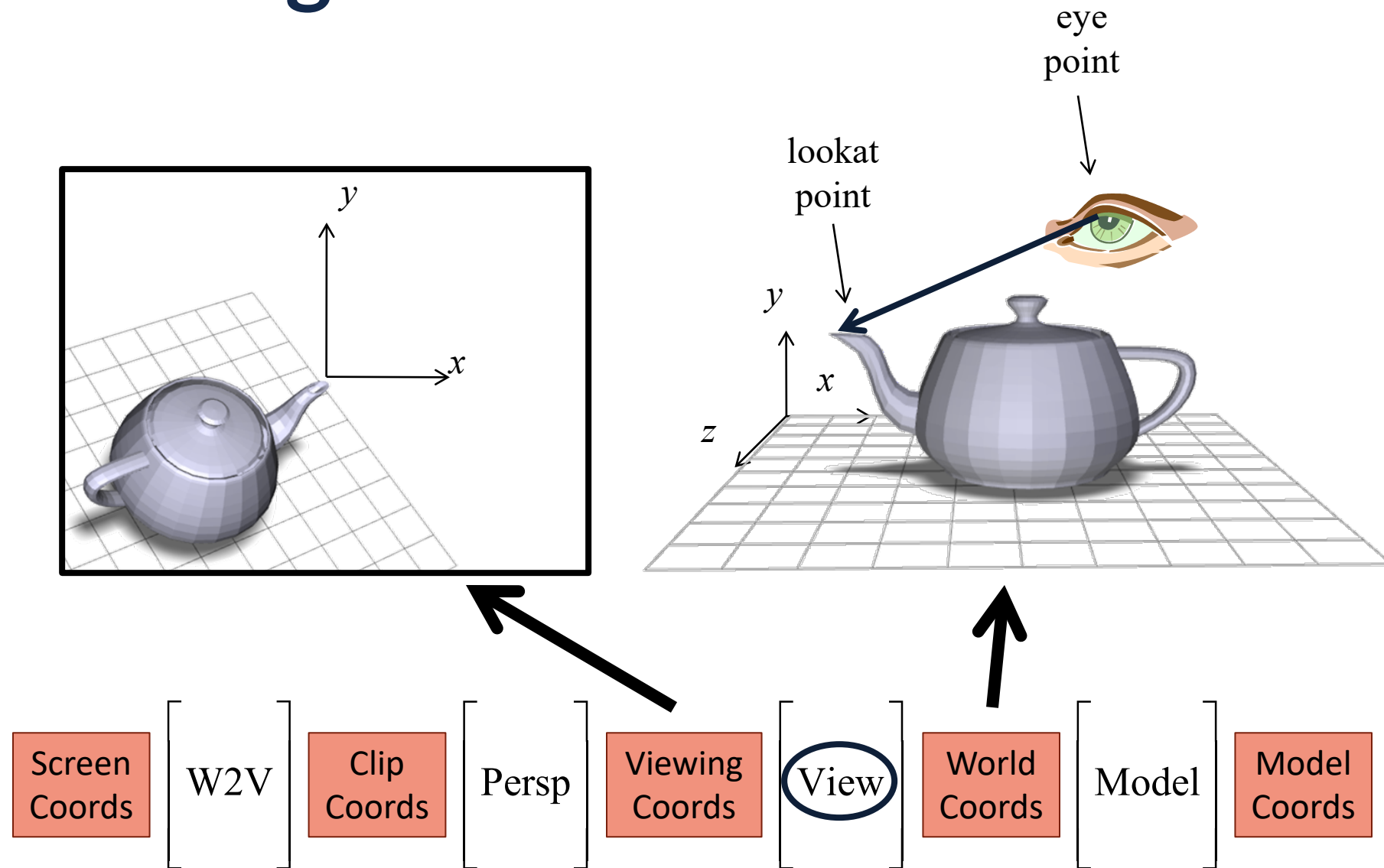


WebGL really only ever renders one viewpoint...looking down the Z axis through the $[-1,-1,-1] \times [1,1,1]$ view volume

So, to see a different view, we need to move everything in the world so that

1. The camera location is moved to the origin
2. The view direction of camera lines up with the z axis

Viewing Transformation



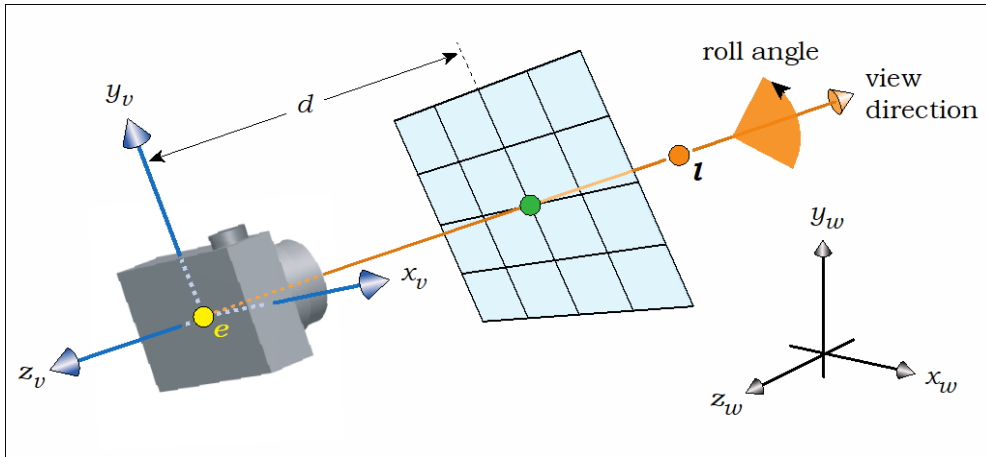
Creating a LookAt Function

Suppose we want to implement a function that sets up a view

There are lots of possible ways to do this...we'll choose a simple ***lookat camera***

The API we create will require someone using the function to specify:

- The **eyepoint** (or camera location)
- The **lookat point** (a point in the view direction)
- An **"up" vector** that we use to specify rotation around the view vector



Deriving the Viewing Transformation

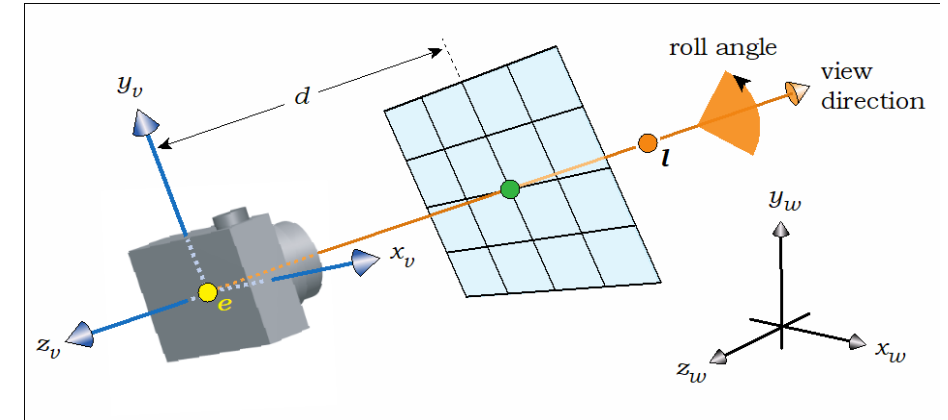
One way to think about what you are doing

- Translate the eyepoint to the origin
- Rotate so that
 - lookat vector aligns with $-z$ axis (OpenGL/WebGL)
 - up aligns with y

We move all objects (the world) this way...

- Another way to think of it

- Create an orthonormal basis with eye at the origin
- And vectors u, v, w as the basis vectors
- ...and then align u, v, w with x, y, z



Local Frames

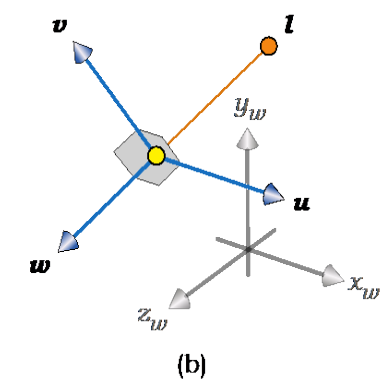
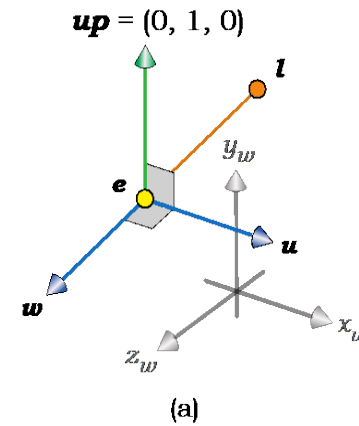
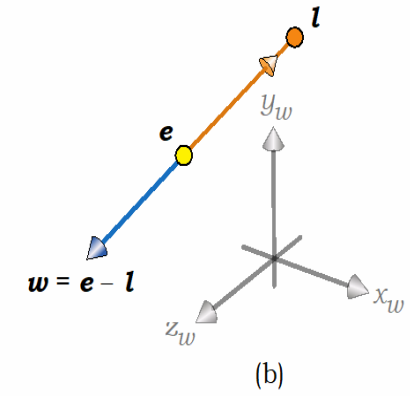
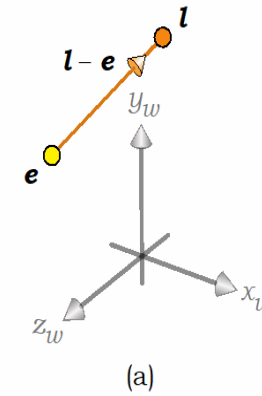
A frame has an origin point and set of basis vectors

Any point can be expressed as coordinates in such a frame

For example $(0,0,0)$ and $\langle 1,0,0 \rangle, \langle 0,1,0 \rangle, \langle 0,0,1 \rangle$

- And an example of a point in that space:

$$(4,0,0) = (0,0,0) + 4 \langle 1,0,0 \rangle + 0 \langle 0,1,0 \rangle + 0 \langle 0,0,1 \rangle$$

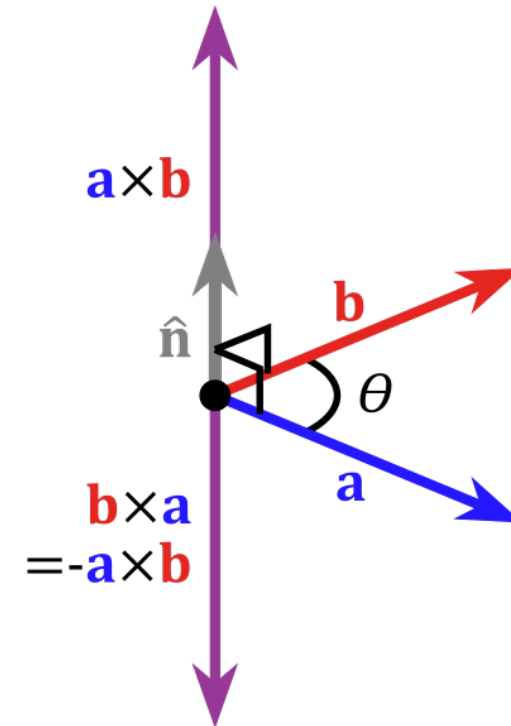


Cross Product of Two Vectors

$$\mathbf{a} = \langle a_1, a_2, a_3 \rangle$$

$$\mathbf{b} = \langle b_1, b_2, b_3 \rangle$$

$$\mathbf{a} \times \mathbf{b} = \langle a_2 b_3 - b_2 a_3, a_3 b_1 - b_3 a_1, a_1 b_2 - b_1 a_2 \rangle$$



Important Property:

The cross product yields a vector orthogonal to the original two vectors

The Orthonormal Basis for View Space

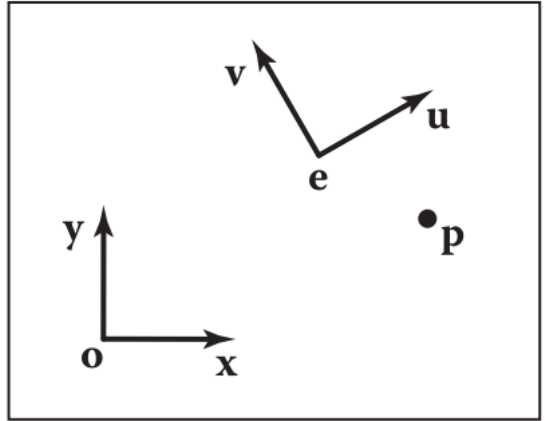
$$l = \text{lookatPoint} - \text{eyepoint}$$

- Let l be the lookat vector...then $w = -\frac{l}{\|l\|}$
- If t is the up direction $u = \frac{t \times w}{\|t \times w\|}$
- And then $v = w \times u$
- The view matrix is then:

$$M_{view} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Why do we negate the lookat vector when computing w ?

Moving Between Frames in 2D



How would we convert from (x,y) to (u,v)?

To convert coordinates from (u,v) space to (x,y) we can:

$$\begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_e \\ 0 & 1 & y_e \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_u & x_v & 0 \\ y_u & y_v & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_p \\ v_p \\ 1 \end{bmatrix} = \begin{bmatrix} x_u & x_v & x_e \\ y_u & y_v & y_e \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_p \\ v_p \\ 1 \end{bmatrix}$$

This can be written as

$$\mathbf{p}_{xy} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{e} \\ 0 & 0 & 1 \end{bmatrix} \mathbf{p}_{uv}$$

View Transformation

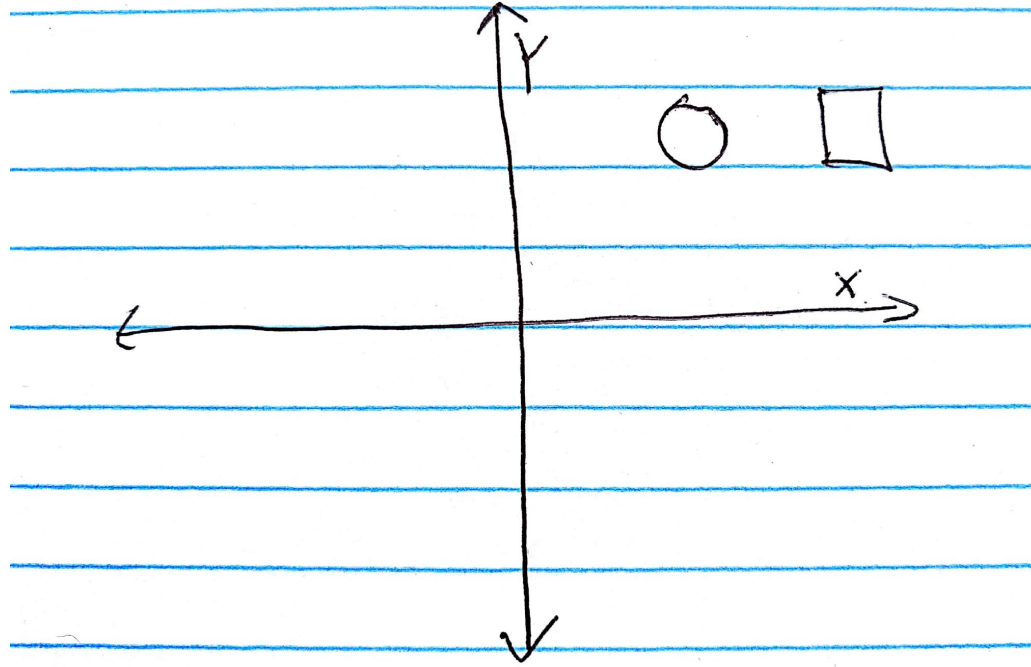
You can now look at your scene from any

- Position
- Orientation (almost)
 - What lookat and up vector pair won't work?

...just uses a matrix multiplication

$$M_{view} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

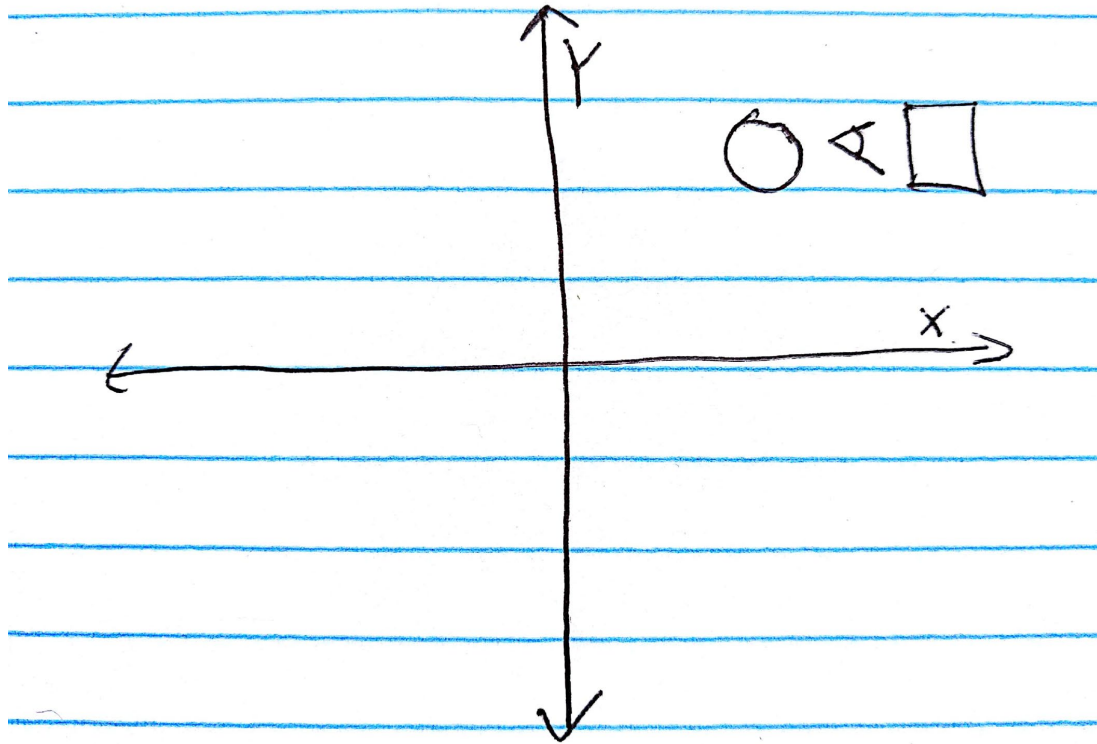
Let's review viewing....



We start out by setting up our geometry in world coordinates

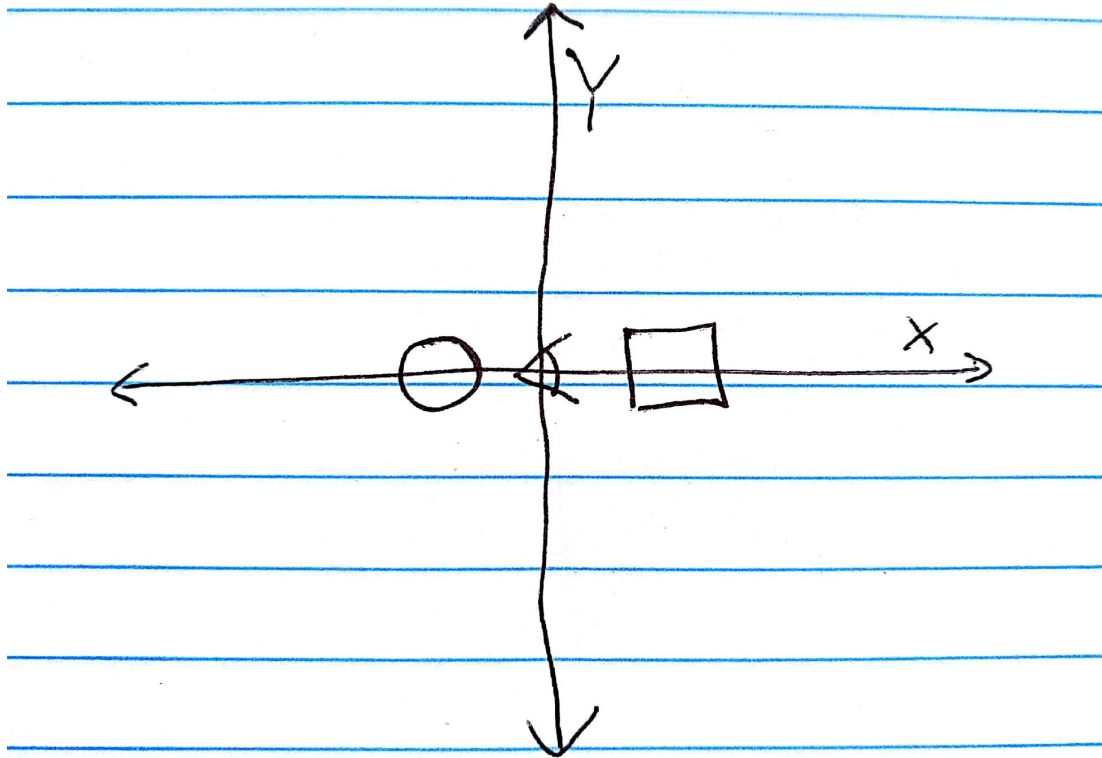
Which transformation does this?

The view transformation



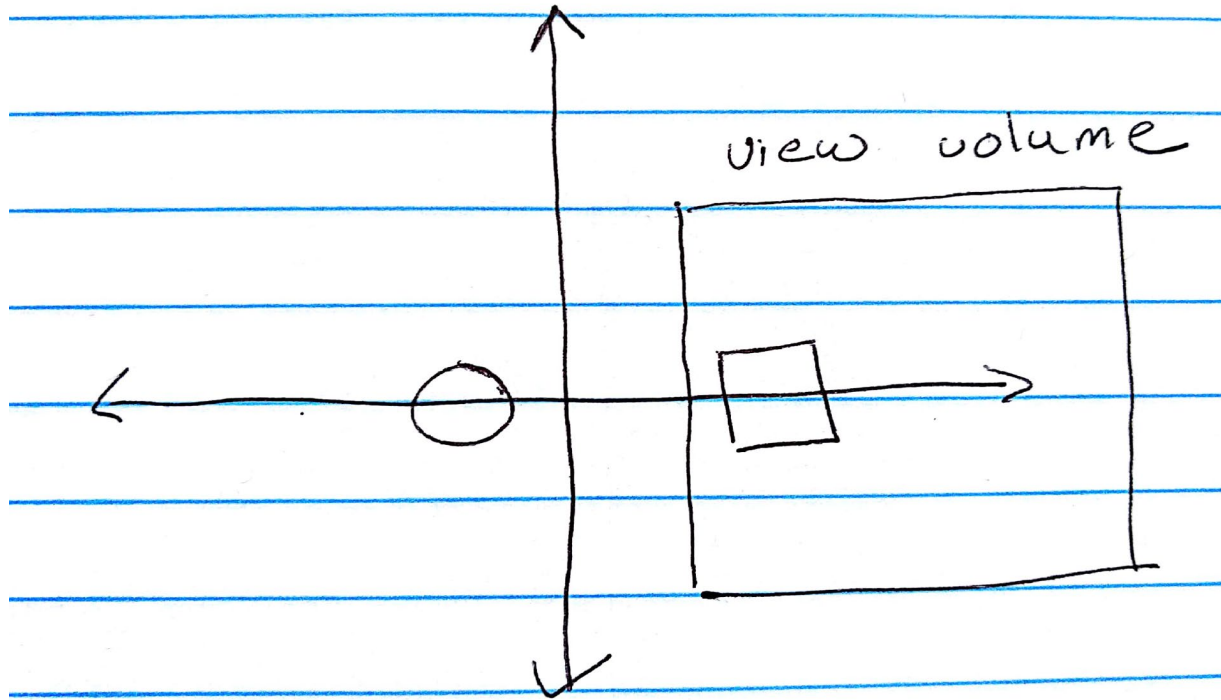
We pick a specific viewing position and direction in worldspace

The view transformation



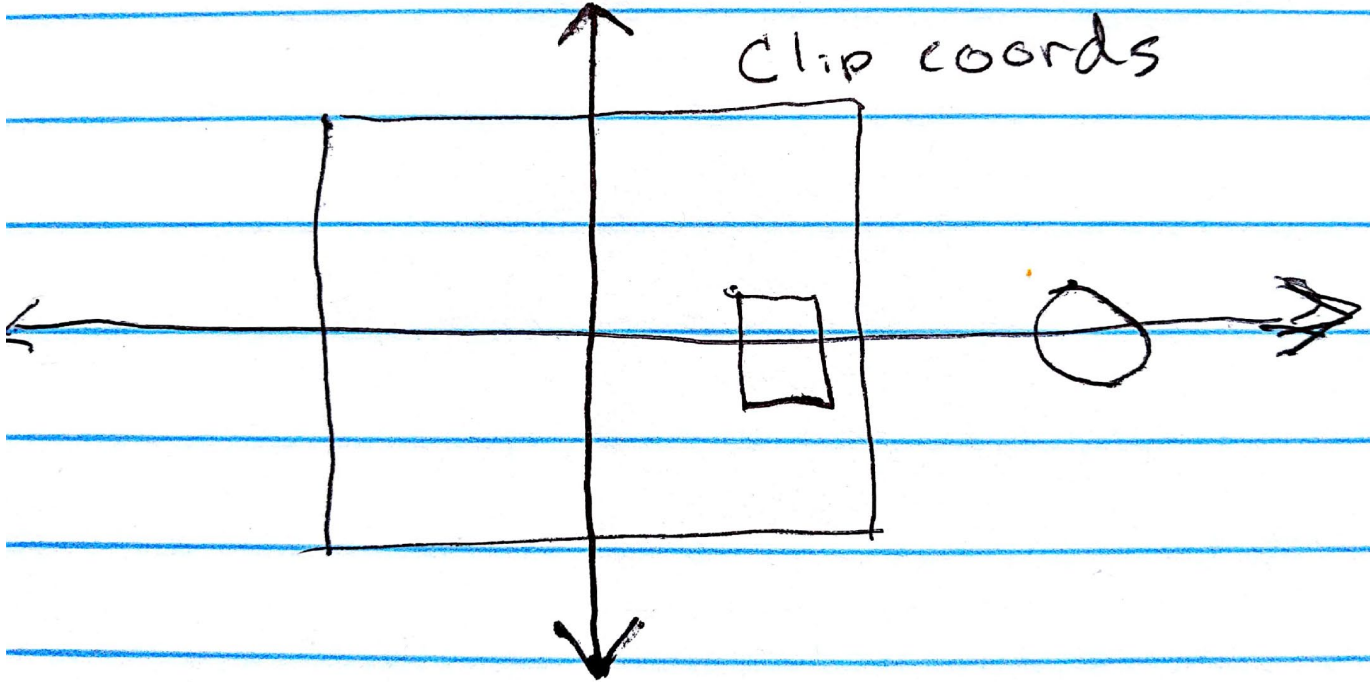
We transform the world so the view position is at the origin.
The view direction is down the $-z$ axis in WebGL

The projection transformation



We pick a viewing volume.
This is specified in ***viewing coordinates***.

The projection transformation



Our view volume is transformed to fit in the WebGL view volume

The WebGL view volume is a box with clip planes at -1 , $+1$

The z coordinates are negated to flip the z -axis