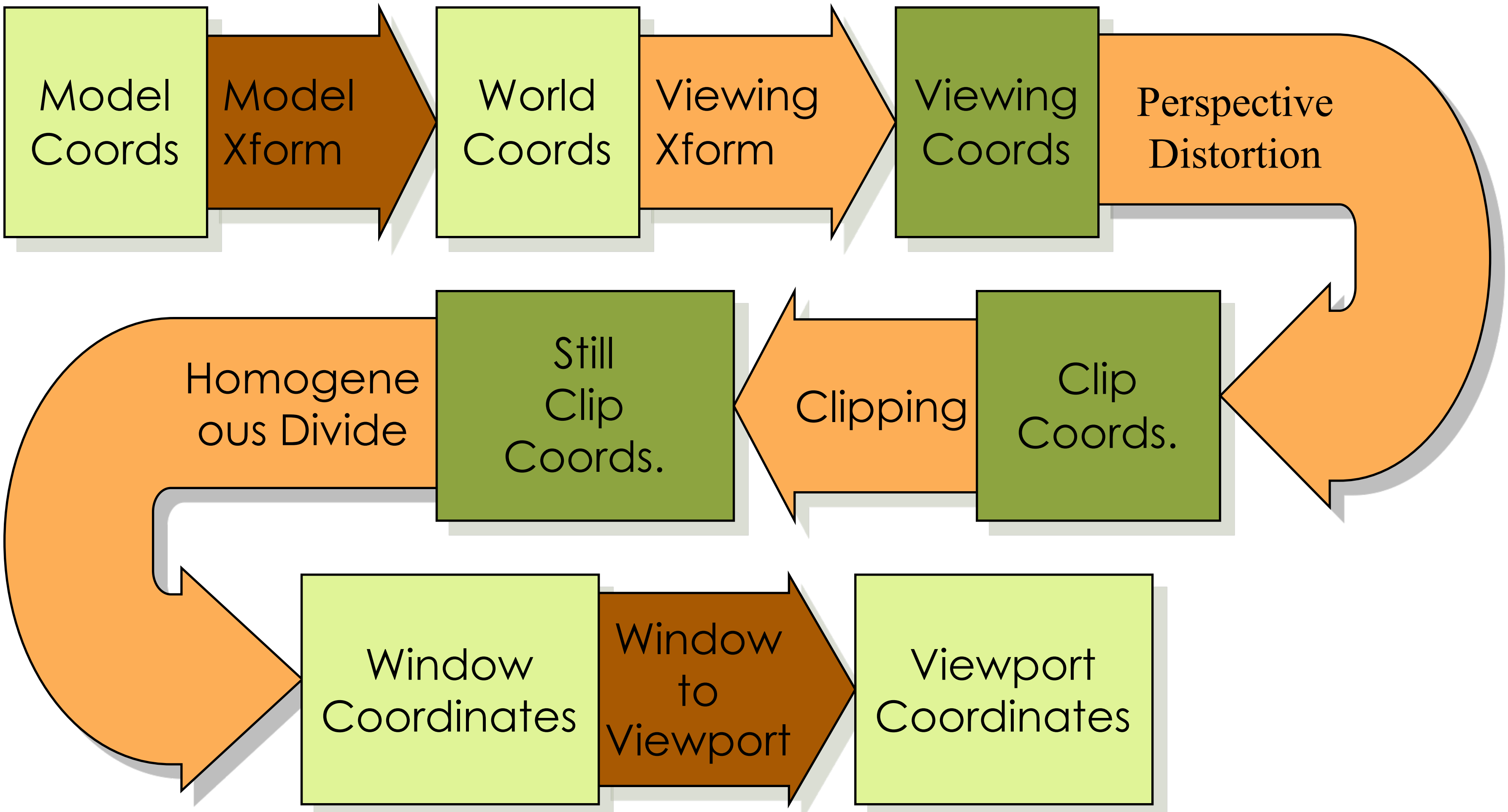
CS 418: Interactive Computer Graphics

Viewing

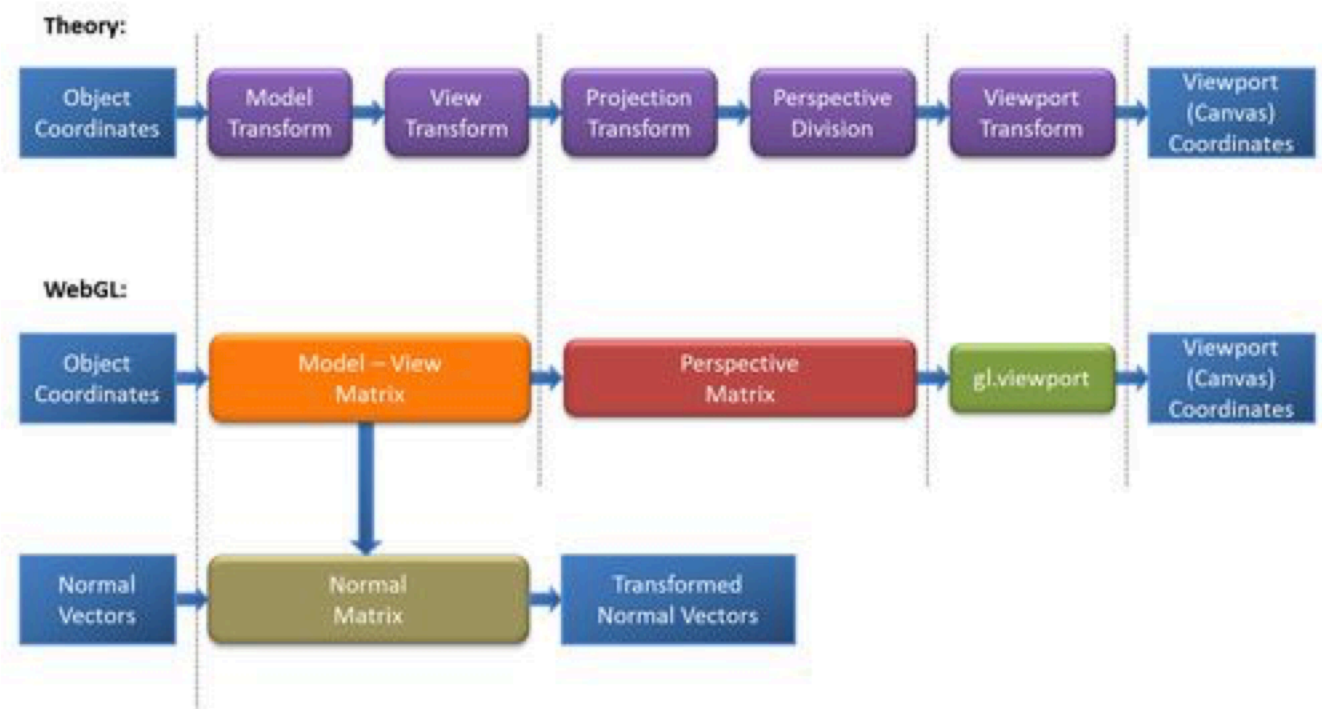
Eric Shaffer

Based on John Hart's CS 418 Slides

Graphics Pipeline



Graphics Pipeline and WebGL



From WebGL Beginner's Guide by Cantor and Jones

Which of the following mean the same thing?

- See if you can guess…
  - Camera transformation
  - Eye transformation
  - View transformation
  - Camera space
  - Eye space
  - Clip space
  - Normalized device coordinates
  - Viewport transformation
  - Windowing transformation
  - Screen space
  - Pixel coordinates
  - Viewport coordinates

Computer graphics has non-standardized vocabulary

- Camera transformation
- Eye transformation
- View transformation
- Camera space
- Eye space
- Clip space
- Normalized device coordinates
- Viewport transformation
- Windowing transformation
- Screen space
- Pixel coordinates
- Viewport coordinates

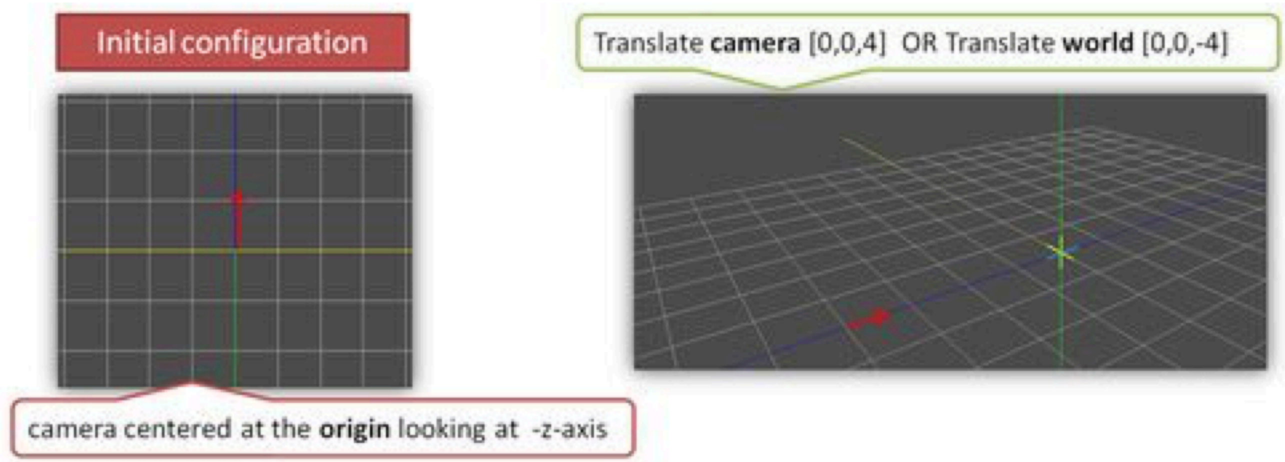
So don't be afraid to ask someone what something means if you haven't heard a term before

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
  - Happens after the modeling transformation
- A ***view matrix*** is functionally equivalent to a camera
- It does the same thing as a model matrix,
  - But it applies the same transformations equally to every object
  - Moving the whole world 5 units towards us = walking 5 units forwards

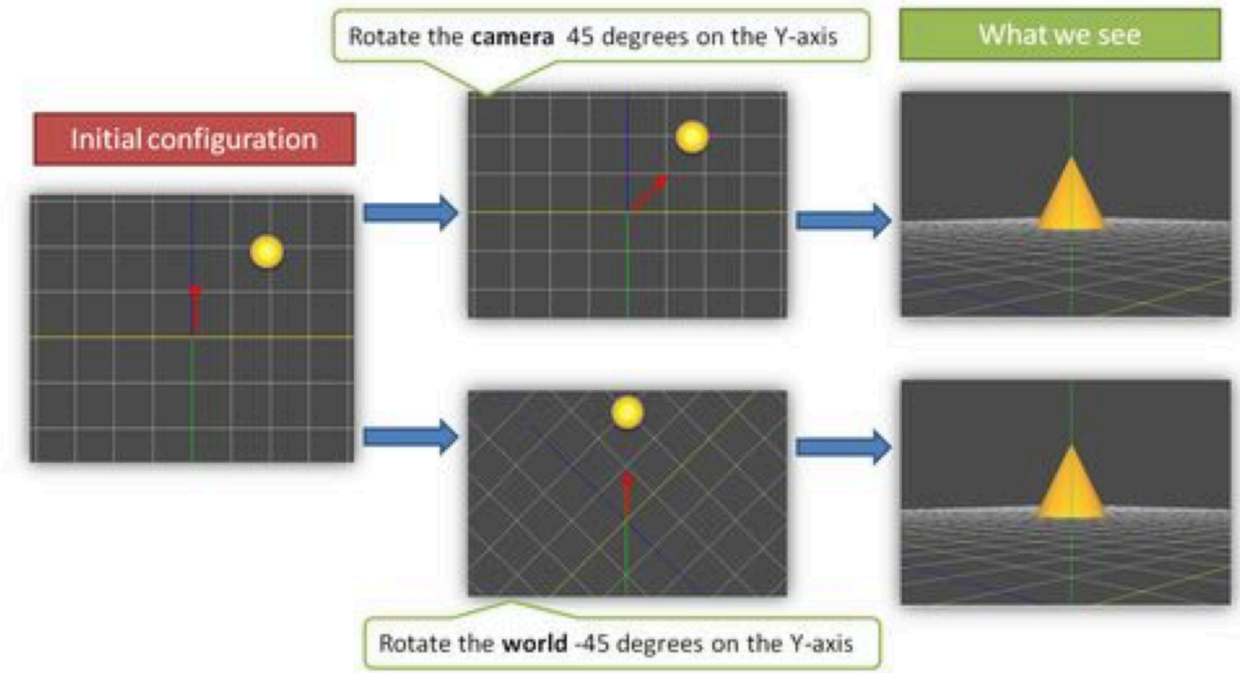
> The engines don't move the ship at all. The ship stays where it is and the engines move the universe around it.
> -- *Futurama*

Example



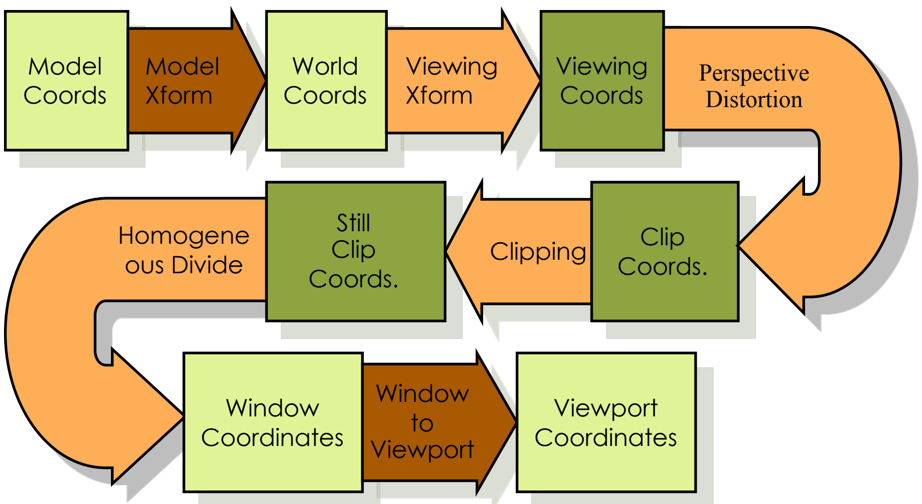
From WebGL Beginner's Guide by Cantor and Jones

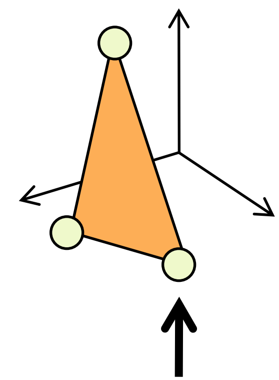
Example

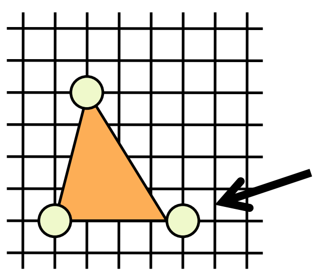


From WebGL Beginner's Guide by Cantor and Jones

Graphics Pipeline

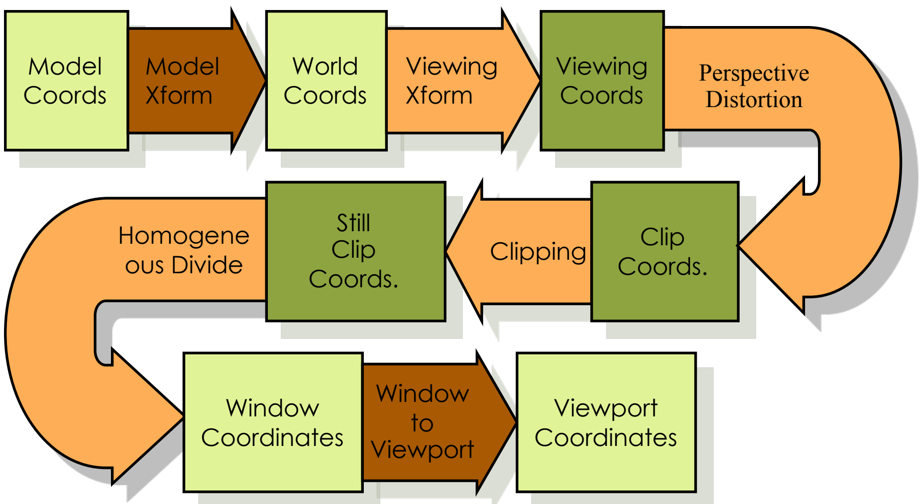


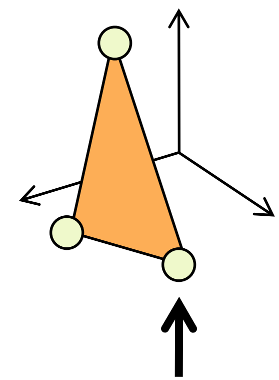


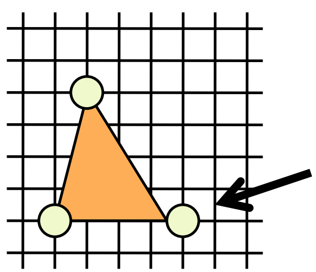


$$\begin{bmatrix} x_s \\ y_s \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} & & \\ & \text{W2V} & \\ & & \end{bmatrix} \begin{bmatrix} & & \\ & \text{Persp} & \\ & & \end{bmatrix} \begin{bmatrix} & & \\ & \text{View} & \\ & & \end{bmatrix} \begin{bmatrix} & & \\ & \text{Model} & \\ & & \end{bmatrix} \begin{bmatrix}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

Graphics Pipeline







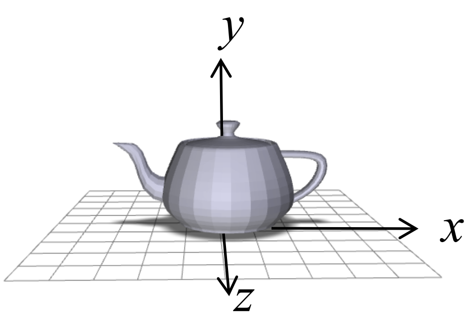
$$\begin{bmatrix} x_s \\ y_s \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} & & & \\ & \mathbf{M} & & \\ & & & \end{bmatrix} \begin{bmatrix} x_m \\ y_m \\ z_m \\ 1 \end{bmatrix}$$

Transformation Order

```
glutSolidTeapot(1);
```

```
glRotate3f(-90, 0,0,1);
glTranslate3f(0,1,0);
glutSolidTeapot(1);
```

```
glTranslate3f(0,1,0);
glRotate3f(-90, 0,0,1);
glutSolidTeapot(1);
```

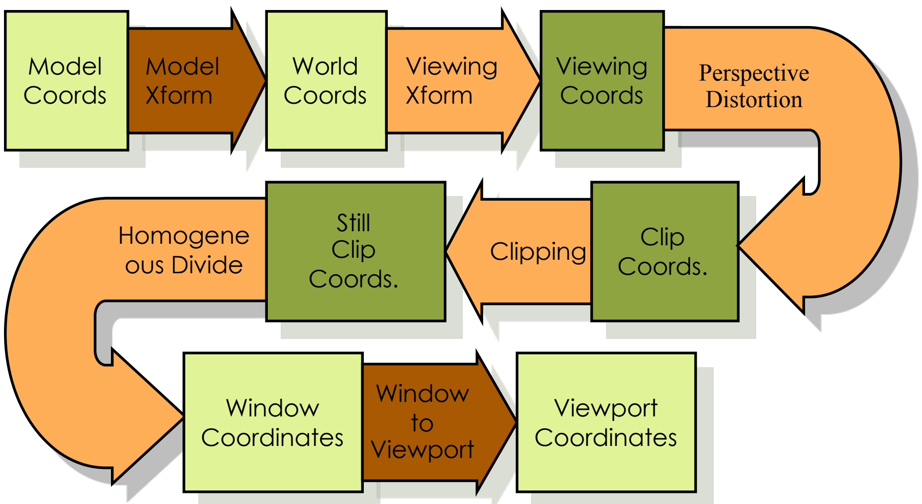


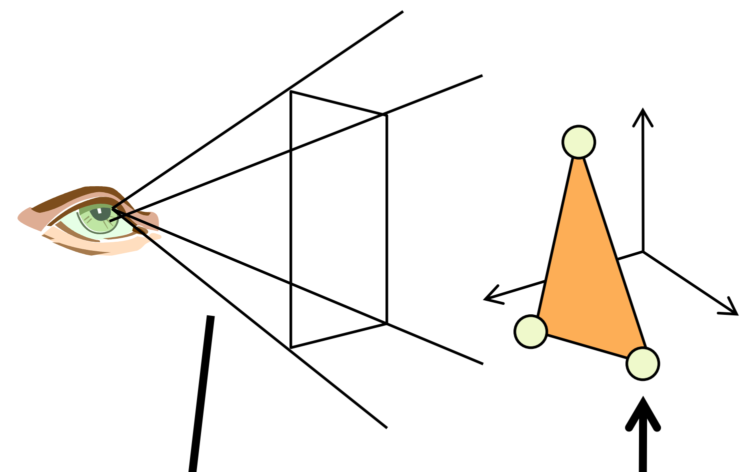
$$\begin{bmatrix} x_s \\ y_s \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{M} \end{bmatrix} \begin{bmatrix} x_m \\ y_m \\ z_m \\ 1 \end{bmatrix}$$

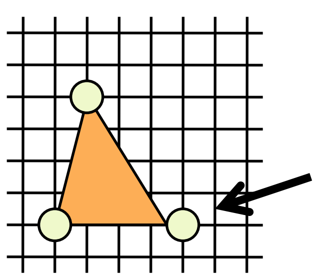
$$\begin{bmatrix} x_s \\ y_s \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{M} \end{bmatrix} \begin{bmatrix} \mathrm{R} \end{bmatrix} \begin{bmatrix} \mathrm{T} \end{bmatrix} \begin{bmatrix} x_m \\ y_m \\ z_m \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x_s \\ y_s \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{M} \end{bmatrix} \begin{bmatrix} \mathrm{T} \end{bmatrix} \begin{bmatrix} \mathrm{R} \end{bmatrix} \begin{bmatrix} x_m \\ y_m \\ z_m \\ 1 \end{bmatrix}$$

Viewing Transformation

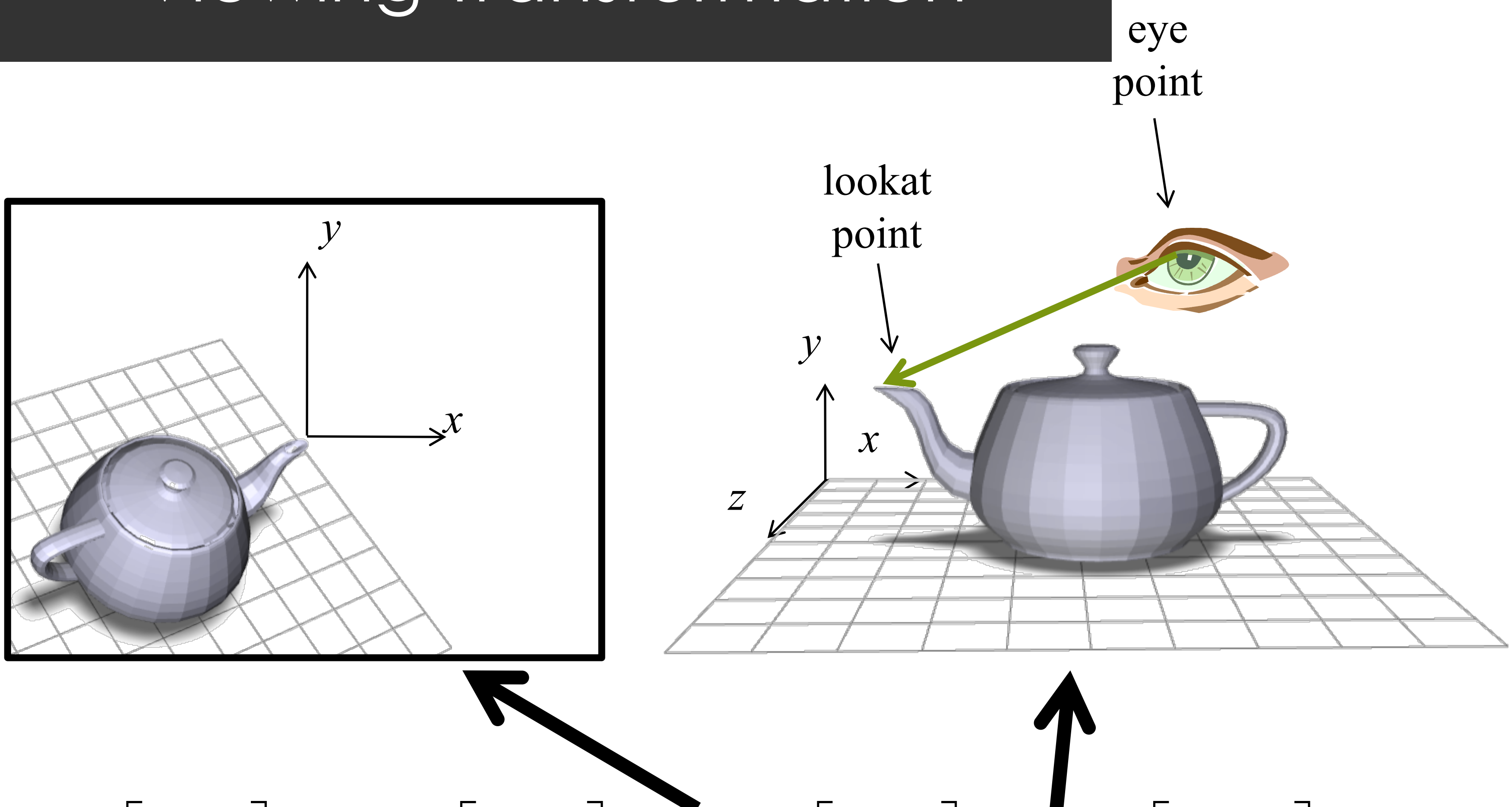






$$\begin{bmatrix} x_s \\ y_s \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \quad \mathrm{W2V} \quad \end{bmatrix} \begin{bmatrix} \quad \mathrm{Persp} \quad \end{bmatrix} \begin{bmatrix} \quad \mathrm{View} \quad \end{bmatrix} \begin{bmatrix} \quad \mathrm{Model} \quad \end{bmatrix} \begin{bmatrix} x_m \\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

Viewing Transformation



Screen Coords	W2V	Clip Coords	Persp	Viewing Coords	View	World Coords	Model	Model Coords

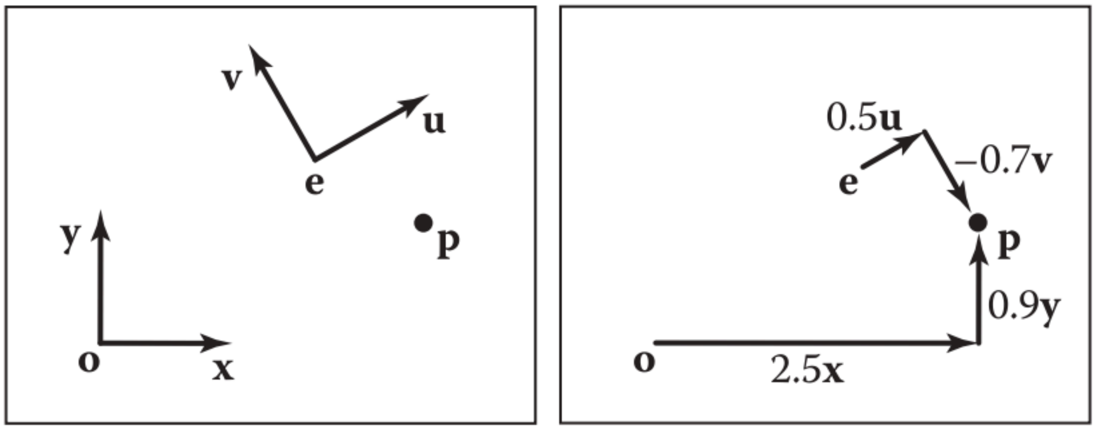
Deriving the Viewing Transformation

- One way to think about what you are doing
  - Translate the eyepoint to the origin
  - Rotate so that lookat vector aligns with –z axis
  - …and up aligns with y
- Another way to think of it
  - We create an orthonormal basis with eye at the origin
  - And vectors u,v,w as the basis vectors
  - …and then aligning u,v,w with x,y,z

Constructing a Local Frame

- 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 $<1,0,0>$, $<0,1,0>$,$<0,0,1>$
  - And a point:
    $(4,0,0)= (0,0,0) + 4 <1,0,0> + 0 <0,1,0> + 0 <0,0,1>$

Example in 2D



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

Forming the Orthonormal Basis for View Space

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

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

Why is the matrix inverted?

View Transformation

- You can now look at your scene from any
  - Position
  - Orientation (almost)
    - What lookat an up vector pair won't work?
- …just uses a matrix multiplication

glMatrix lookAt transform

{mat4} **mat4.lookAt(out, eye, center, up)**

Generates a look-at matrix with the given eye position, focal point, and up axis

Parameters:
{mat4} out
mat4 matrix will be written into

{vec3} eye
Position of the viewer

{vec3} center
Point the viewer is looking at

{vec3} up
vec3 pointing up

Returns:
{mat4} out

glMatrix can compute a lookAt transformation for you

There are other cameras you can use besides lookAt-Style cameras