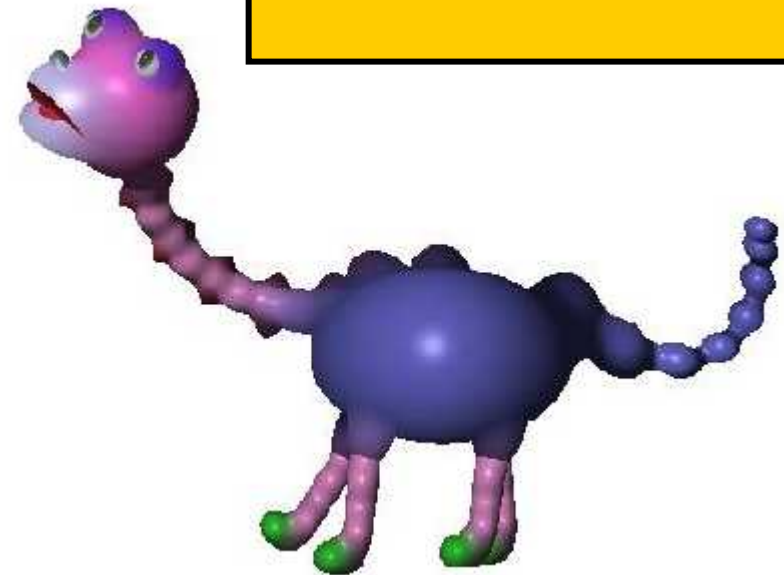
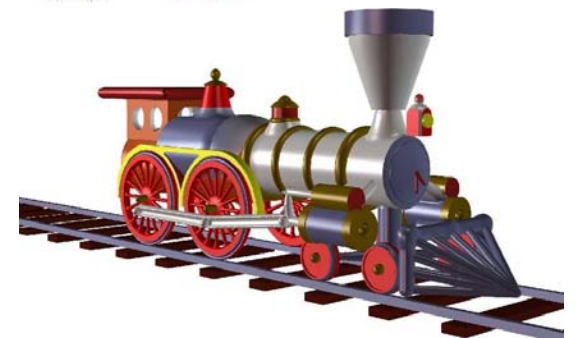
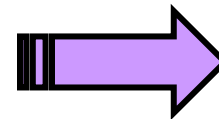


CSC 305
The Graphics
Pipeline-1

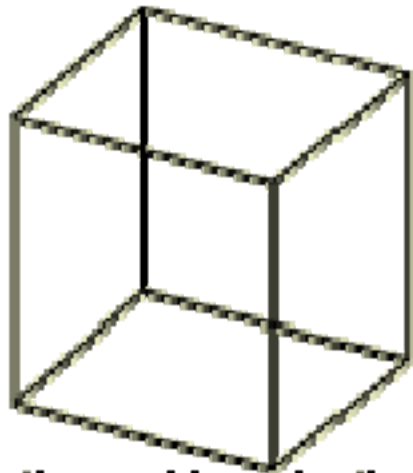
by Brian Wyvill



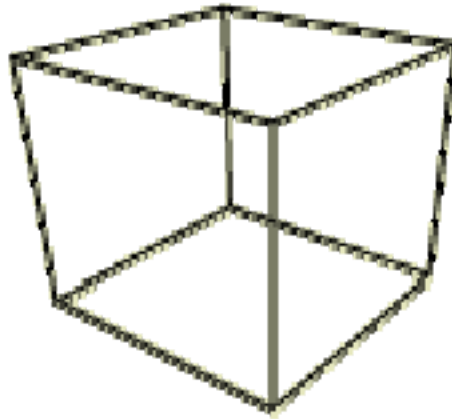
The University of Victoria
Graphics Group



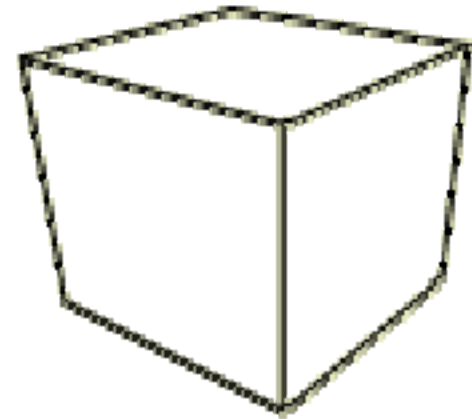
Perspective Viewing Transformation



orthographic projection



perspective projection

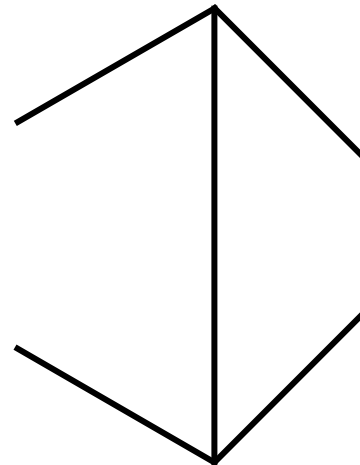
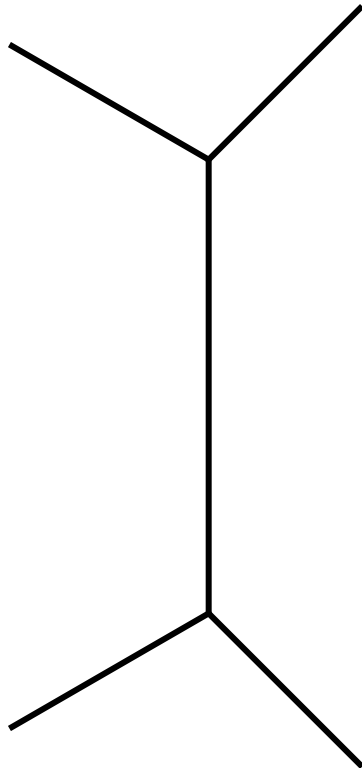


hidden lines removed

- Tools for creating and manipulating a “camera” that produces pictures of a 3D scene
- Viewing transformations and projections
- Perform culling or back-face elimination



The Illusion of Depth

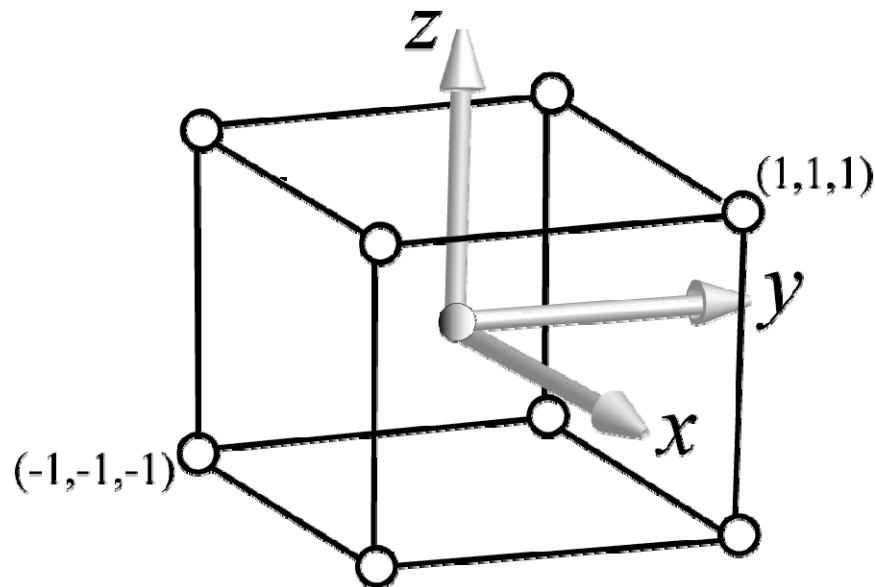


Perception is an active organising process
Many cues to depth!



Canonical View Volume

(as per p160 Shirley)



Object to map lines in the canonical view volume to the screen.

$(x,y,z) \in [-1, 1]^3$
(as in interval $[a,b]$)

For now assume that all line segments are in the view volume (clipping later!)

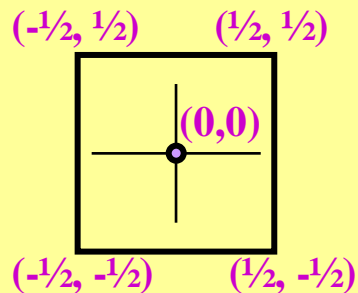
Note Y will be up in next edition of Shirley!



Mapping to the *screen*

(as per p161 Shirley)

Pixel Geometry



Screen n_x by n_y pixels

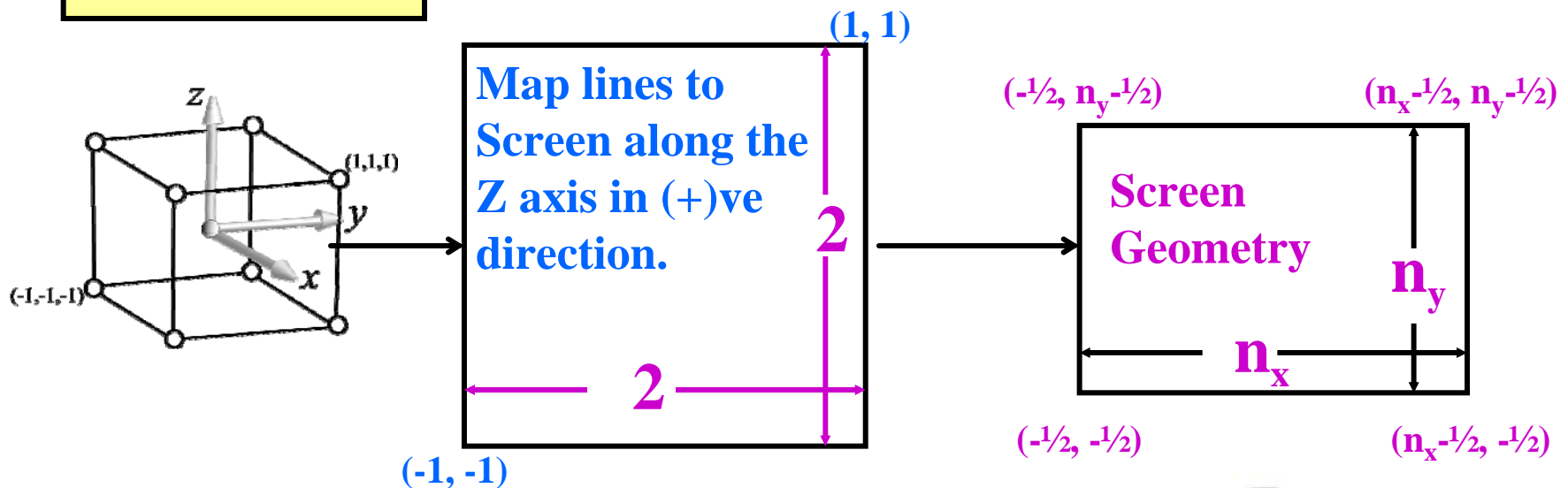
$x = -1 \rightarrow$ left side of screen

$x = +1 \rightarrow$ right side of screen

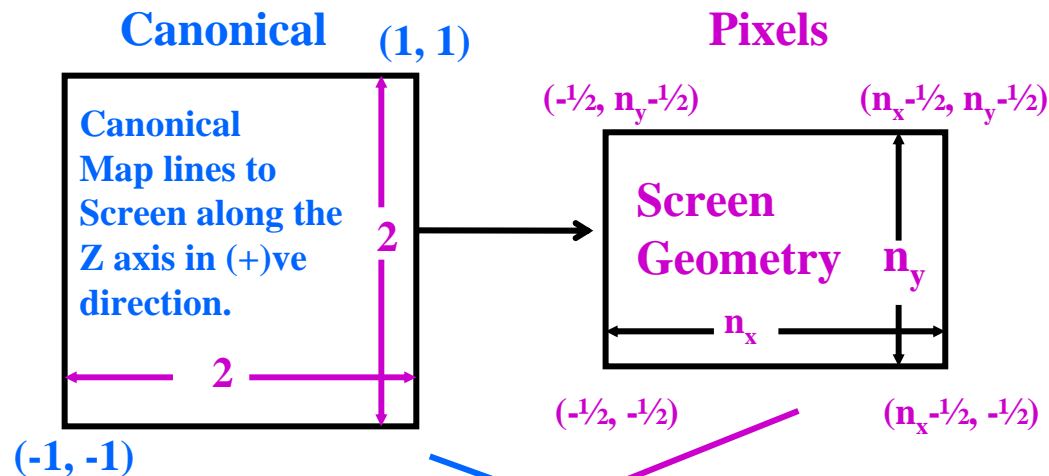
$y = -1 \rightarrow$ bottom of screen

$y = +1 \rightarrow$ top of screen

Maps square $[-1, +1]^2$ to non-square scales S_x and S_y will be defined.



As Before a Window to Viewport Transform



$$\begin{bmatrix} x_{\text{pixel}} \\ y_{\text{pixel}} \\ 1 \end{bmatrix} = \begin{bmatrix} n_x/2 & 0 & (n_x-1)/2 \\ 0 & n_y/2 & (n_y-1)/2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\text{canonical}} \\ y_{\text{canonical}} \\ 1 \end{bmatrix}$$

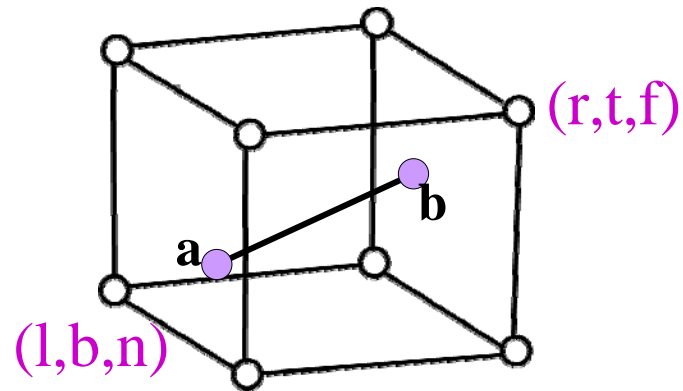
For now simply
ignore
The z-values



Orthographic Projection

What happens if we don't have a canonical view volume?

(also, what happened to Z in the previous example? p162)



$(l,b,n) = (\text{lower, bottom, near})$

$(r,t,f) = (\text{right, top, front})$

Find matrix M s.t.

$M.a$ and $M.b$ are in canonical view volume. e.g lines in Orthographic view volume above

Bounded by axis aligned planes

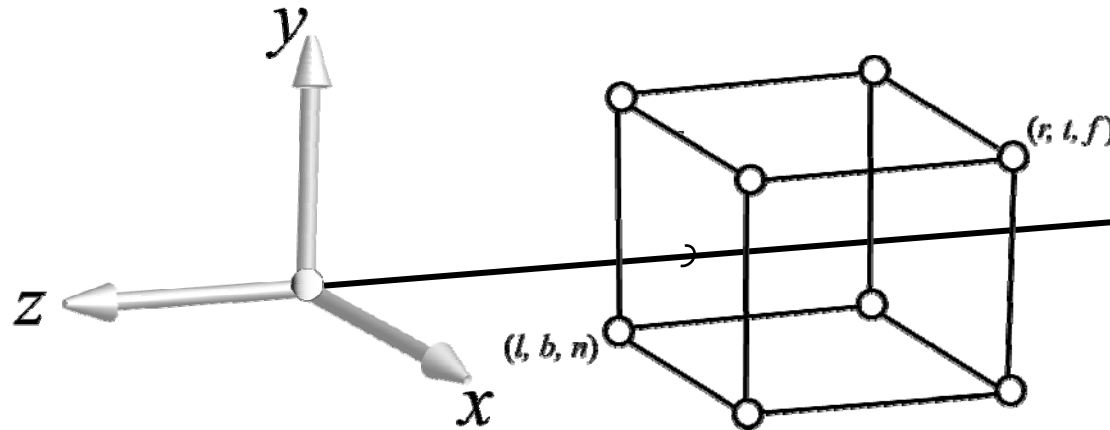
Left plane $\rightarrow X \rightarrow$ Right Plane

Bottom plane $\rightarrow y \rightarrow$ Top Plane

Near plane $\rightarrow Z \rightarrow$ Far Plane



Orthographic View Volume



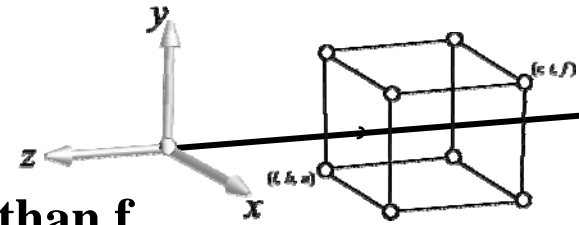
Right Handed System
Gaze (or camera view)
along $-Z$ direction
(note $n > f$)
 x to the right and y up

The transform we want is just a scale and translate.
It takes:

transforms to		transforms to	
$y = b$	\rightarrow	$y = -1$,	$y = t \rightarrow y = +1$
$x = l$	\rightarrow	$x = -1$,	$x = r \rightarrow x = +1$
$z = n$	\rightarrow	$z = 1$,	$z = f \rightarrow z = -1$



Scale then translate



n is less (-)ve than f

$$\begin{bmatrix} x_{\text{canonical}} \\ y_{\text{canonical}} \\ z_{\text{canonical}} \\ 1 \end{bmatrix} = \begin{bmatrix} 2/(r-l) & 0 & 0 & 0 \\ 0 & 2/(t-b) & 0 & 0 \\ 0 & 0 & 2/(n-f) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -(l+r)/2 \\ 0 & 1 & 0 & -(b+t)/2 \\ 0 & 0 & 1 & -(n+f)/2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

The earlier transform (p6) took canonical coords and put them on the Screen. Combining that with the above (and adding in the canonical z):

$$\mathbf{M}_0 = \begin{bmatrix} n_x/2 & 0 & 0 & (n_x-1)/2 \\ 0 & n_y/2 & 0 & (n_y-1)/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2/(r-l) & 0 & 0 & 0 \\ 0 & 2/(t-b) & 0 & 0 \\ 0 & 0 & 2/(n-f) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -(l+r)/2 \\ 0 & 1 & 0 & -(b+t)/2 \\ 0 & 0 & 1 & -(n+f)/2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Orthographic Projection:

Now we can find pixel coordinates from some user defined view volume coordinates:

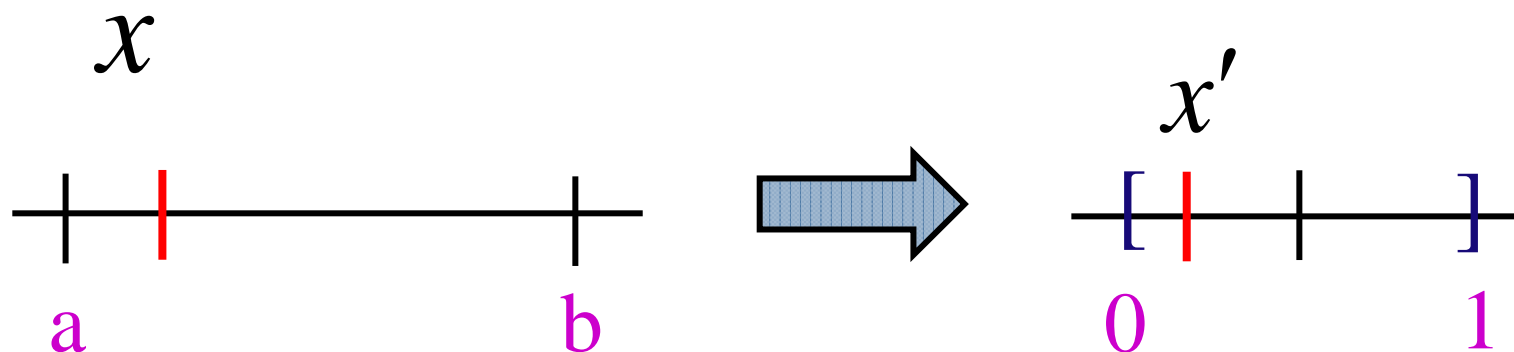
$$\begin{bmatrix} x_{\text{canonical}} \\ y_{\text{canonical}} \\ z_{\text{canonical}} \\ 1 \end{bmatrix} = \mathbf{M}_0 \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Note that z will be in [-1,1] useful later for z-buffer



Another Way of looking at this

Map: $[a,b] \Rightarrow [0,1]$



Map: [a,b] \Rightarrow [0,1]

- Translate to Origin

$$[a, b] \rightarrow [a - a, b - a] = [0, b - a]$$

- Map x to translated interval

$$x \rightarrow x - a$$



Map: [a,b] \Rightarrow [0,1]

- **Normalize the interval**

$$[0, b - a] \rightarrow \frac{1}{b - a} [a - a, b - a] = [0, 1]$$

- **Map x to normalized interval**

$$x \rightarrow \frac{x - a}{b - a}$$



Scale and translate

$$\underbrace{\begin{bmatrix} \left(\frac{1}{b-a}\right) & 0 \\ 0 & 1 \end{bmatrix}}_{S_x\left(\frac{1}{b-a}\right)} \underbrace{\begin{bmatrix} 1 & -a \\ 0 & 1 \end{bmatrix}}_{T_x(-a)} \begin{bmatrix} x \\ 1 \end{bmatrix} = \begin{bmatrix} \left(\frac{x-a}{b-a}\right) \\ 1 \end{bmatrix}$$

This is a homogeneous form for 1D



Map: [a,b] \Rightarrow [0,1]

$$\begin{bmatrix} \frac{1}{b-a} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -a \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \left(\frac{x-a}{b-a} \right) \\ y \\ 1 \end{bmatrix}$$



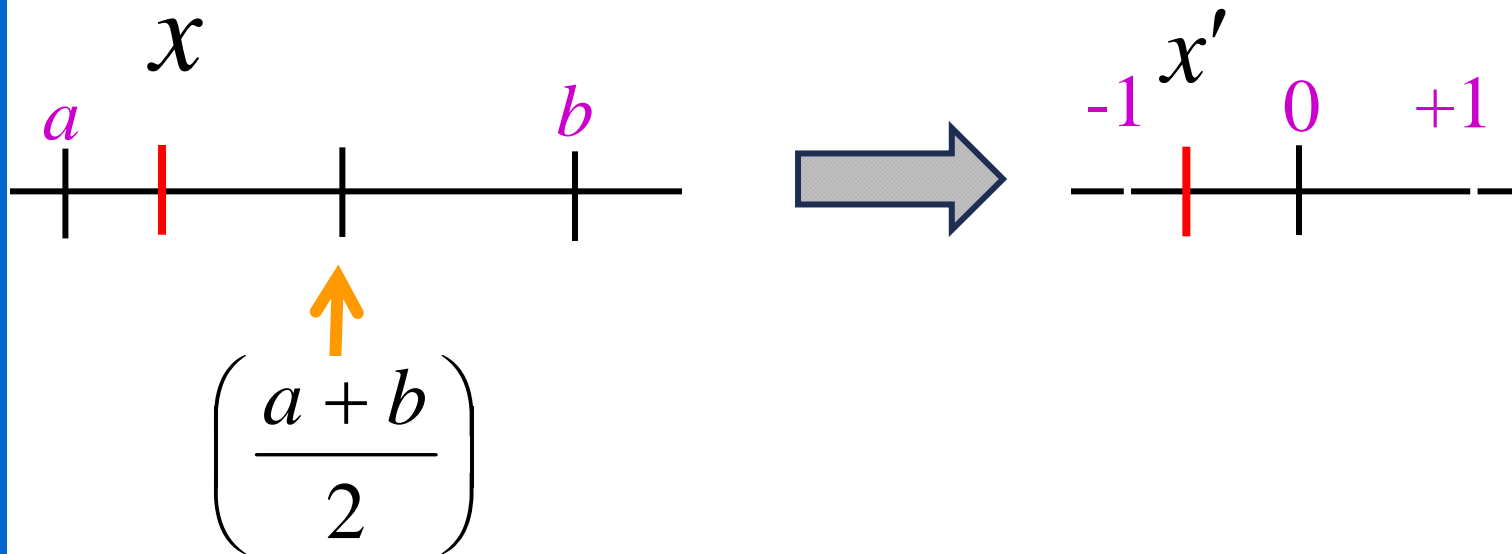
$$S_x \left(\frac{1}{b-a} \right)$$

$$T_x(-a)$$

*(starts to look
like a window to
viewport
transformation)*



Map: $[a,b] \Rightarrow [-1,1]$



Map: $[a,b] \Rightarrow [-1,1]$

- **Translate center of interval to origin**

$$x \rightarrow \left[x - \frac{a+b}{2} \right]$$

- **Normalize interval to $[-1,1]$**

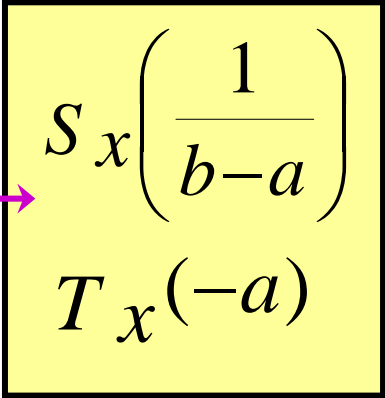
$$\left[x - \frac{a+b}{2} \right] \rightarrow \frac{2}{b-a} \left[x - \frac{a+b}{2} \right]$$



Map: $[a,b] \Rightarrow [c,d]$

- **First map $[a,b]$ to $[0,1]$**

— (We already did this)


$$S_x\left(\frac{1}{b-a}\right)$$
$$T_x(-a)$$

- **Then map $[0,1]$ to $[c,d]$**



Map: $[0,1] \Rightarrow [c,d]$

- **Scale $[0,1]$ by $[c,d]$**
- **Then translate by c**
- **That is, in 1D homogeneous form:**

$$\underbrace{\begin{bmatrix} 1 & c \\ 0 & 1 \end{bmatrix}} \underbrace{\begin{bmatrix} (d-c) & 0 \\ 0 & 1 \end{bmatrix}} \begin{bmatrix} x \\ 1 \end{bmatrix} = \begin{bmatrix} (d-c)x + c \\ 1 \end{bmatrix}$$



All Together: Map: [a,b] \Rightarrow [c,d]

$$\underbrace{\begin{bmatrix} 1 & c \\ 0 & 1 \end{bmatrix} \begin{bmatrix} (d-c) & 0 \\ 0 & 1 \end{bmatrix}}_{\text{2. Map } [0,1] \rightarrow [c,d]} \underbrace{\begin{bmatrix} \left(\frac{1}{b-a} \right) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -a \\ 0 & 1 \end{bmatrix}}_{\text{1. Map } [a,b] \rightarrow [0,1]} \begin{bmatrix} x \\ 1 \end{bmatrix}$$

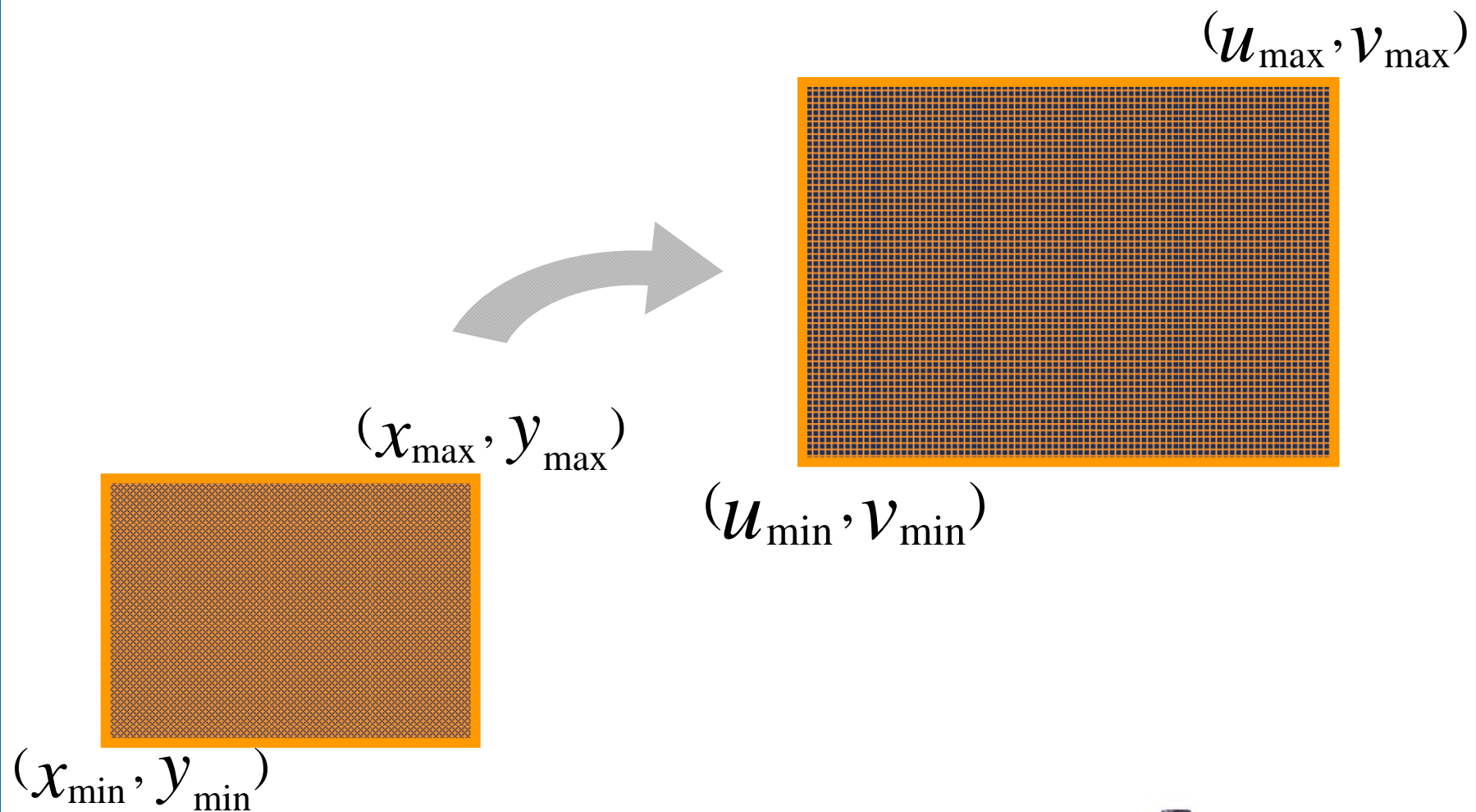
2. Map [0,1] \rightarrow [c,d]

1. Map [a,b] \rightarrow [0,1]

$$= \begin{bmatrix} 1 & c \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \left(\frac{d-c}{b-a} \right) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -a \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix}$$



Now Map Rectangles



Transformation in x and y

$$\begin{bmatrix} 1 & 0 & u_{\min} \\ 0 & 1 & v_{\min} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \lambda_x & 0 & 0 \\ 0 & \lambda_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_{\min} \\ 0 & 1 & -y_{\min} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

where, $\lambda_x = \left(\frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} \right)$, $\lambda_y = \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}}$



This is a Viewport Transformation

- Good for mapping objects from one coordinate system to another
- This is what we do with windows and viewports



Window to Viewport Transform Revisited:

$$\begin{bmatrix} 1 & 0 & u_{\min} \\ 0 & 1 & v_{\min} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \lambda_x & 0 & 0 \\ 0 & \lambda_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_{\min} \\ 0 & 1 & -y_{\min} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathbf{M}_0 = \begin{bmatrix} n_x/2 & 0 & 0 & (n_x-1)/2 \\ 0 & n_y/2 & 0 & (n_y-1)/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2/(r-l) & 0 & 0 & 0 \\ 0 & 2/(t-b) & 0 & 0 \\ 0 & 0 & 2/(n-f) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -(l+r)/2 \\ 0 & 1 & 0 & -(b+t)/2 \\ 0 & 0 & 1 & -(n+f)/2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Remember \mathbf{M}_0 ? Another Window to Viewport Transform



Orthographic Projection Algorithm

```
compute  $M_0$ 
for each line segment in 3D ( $a_i, b_i$ ) do {
     $p = M_0 a_i$ 
     $q = M_0 b_i$ 
    drawline( $x_p, y_p, x_q, y_q$ )
}
```

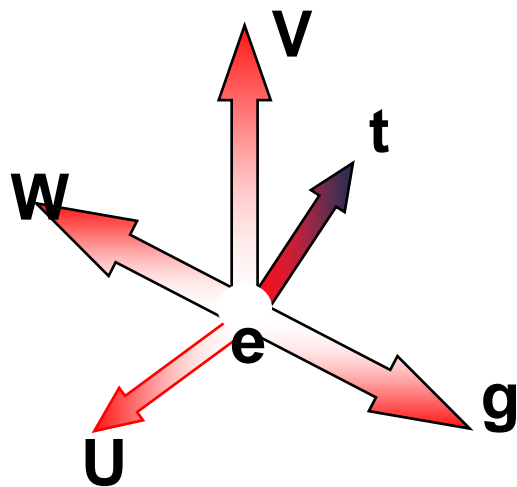


Arbitrary View Positions

e = Eye Position

g = gaze direction

t = view up vector



Derive a coordinate system
with origin e and uvw basis

$$W = -\frac{g}{\|g\|}$$

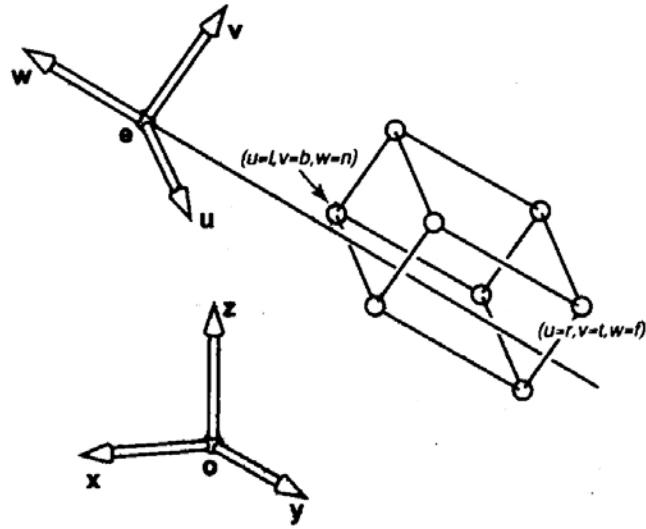
$$U = \frac{t \times W}{\|t \times W\|}$$

$$V = W \times U$$

View up vector points to the sky! Bi-sects the viewers head
as in photography.



Arbitrary View Positions



View volume coordinates:
origin o and xyz axes.

need to convert these to
origin e and uvw axes.

We can use:

$$M_v = \begin{bmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Orthographic Projection Algorithm

Arbitrary View Point

compute M_v

compute M_0

$M = M_0 M_v$

for each line segment in 3D (a_i, b_i) do {

$p = Ma_i$

$q = Mb_i$

 drawline(x_p, y_p, x_q, y_q)

}

