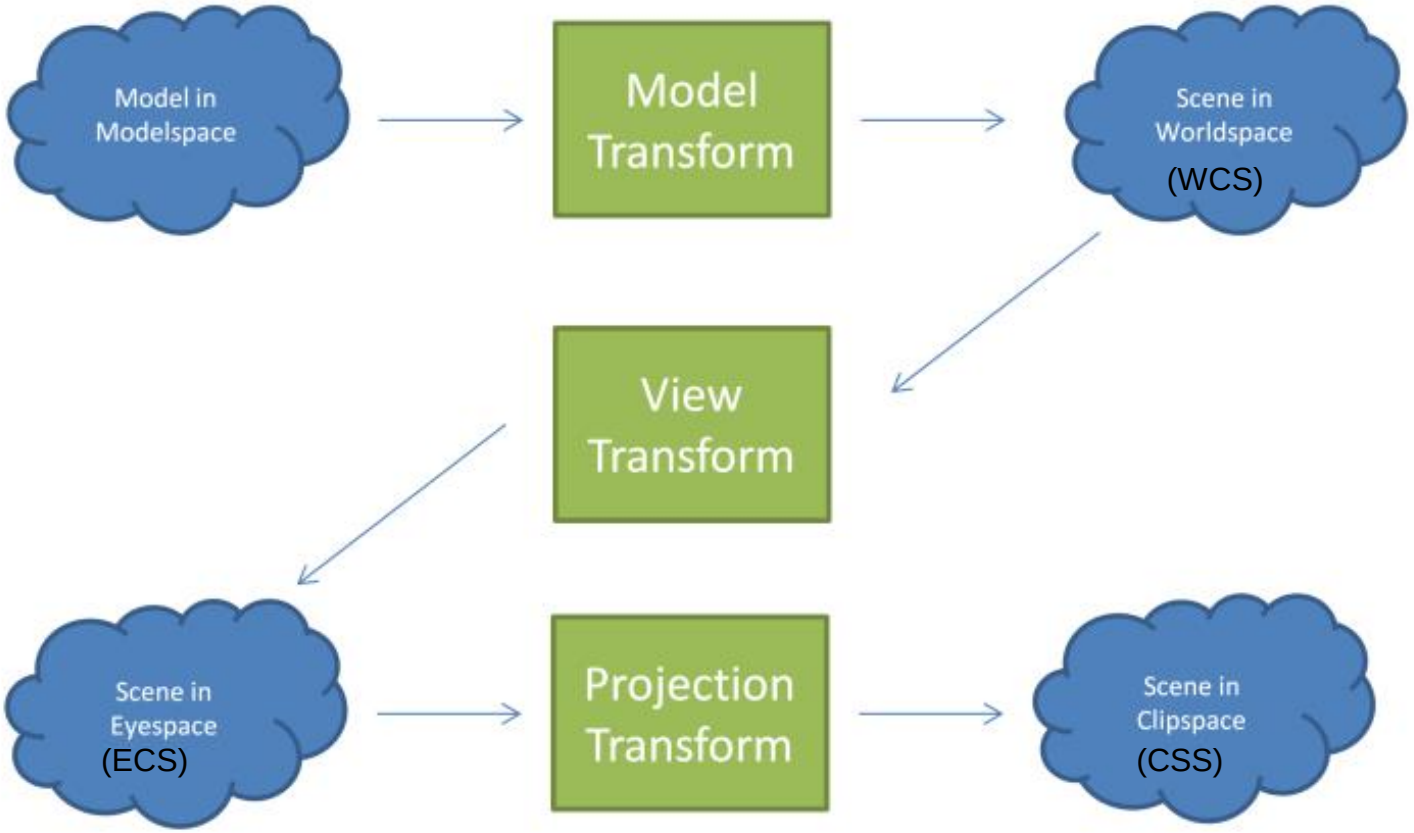
Graphics & Visualization

Chapter 4

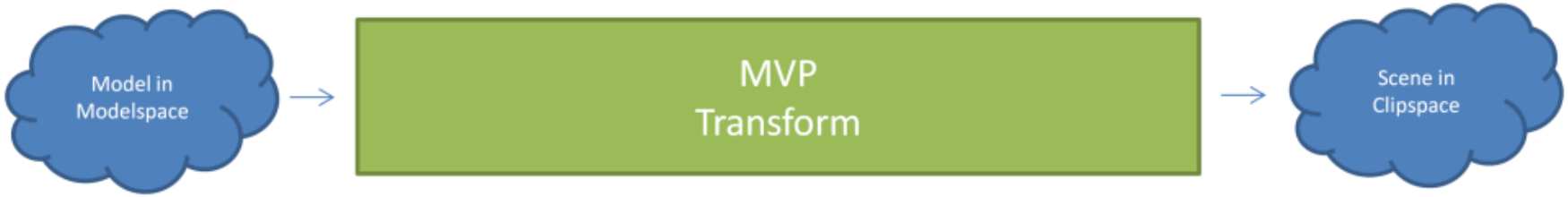
Projections and Viewing Transformations

Coordinate Systems



Coordinate Systems (2)

$$M_{MVP} = M_{Projection} * M_{View} * M_{Model}$$

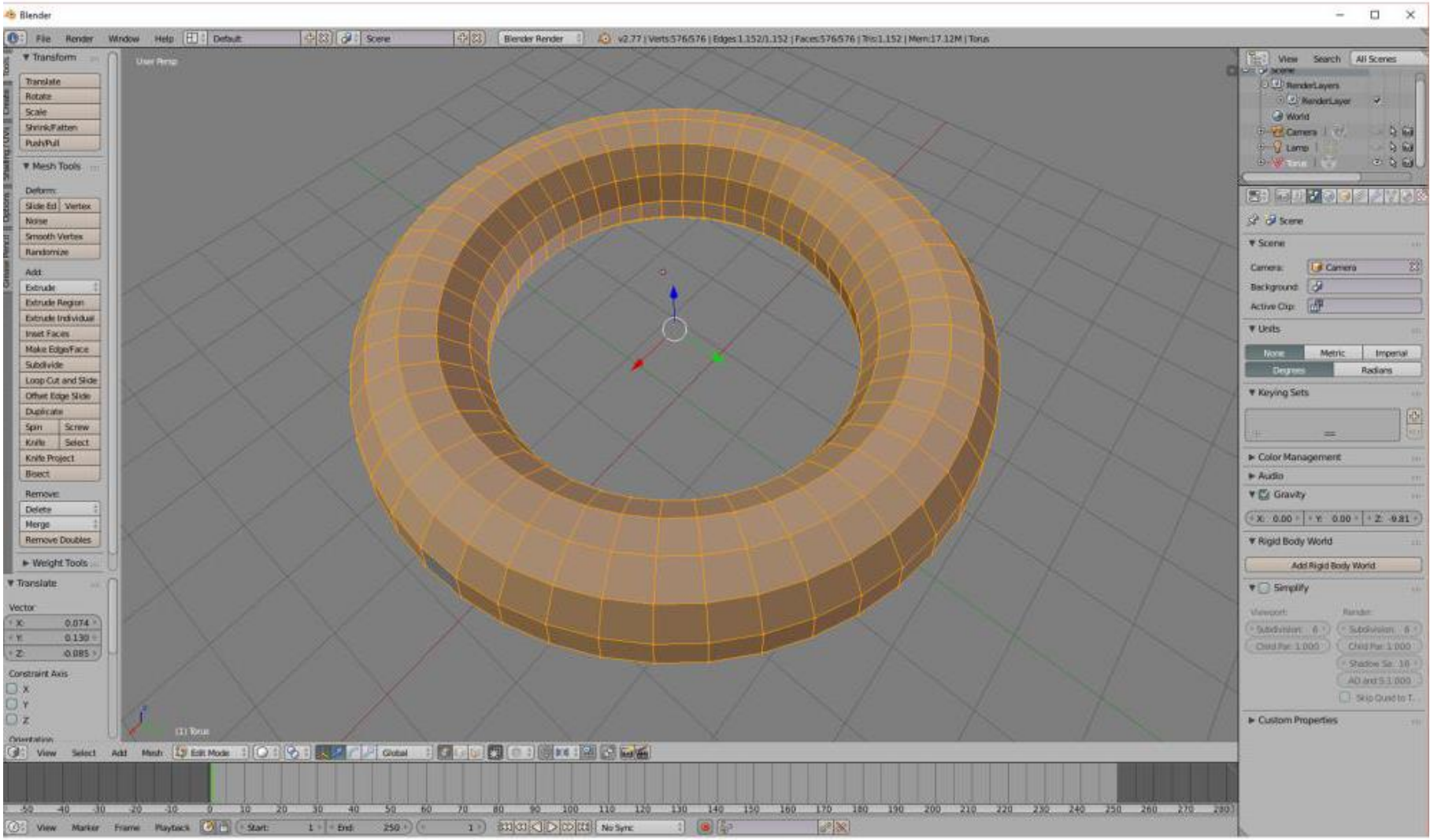


Coordinate Systems (3)

- All objects are initially defined in their own local coordinate system
- These objects are unified in WCS
- The WCS is essentially used to define the model of a 3D synthetic world
- The transition from WCS to ECS:
  - Simplifies a number of operations including culling (e.g. the specification of the clipping bounds by the user) and projection
- The transition from ECS to CSS:
  - Ensures that all objects that survived culling will be defined in a canonical space (usually [-1,1])
- Objects defined in a canonical space:
  - Can easily be scaled to the actual coordinates of any display device or monitor
  - Maintain high floating-point accuracy

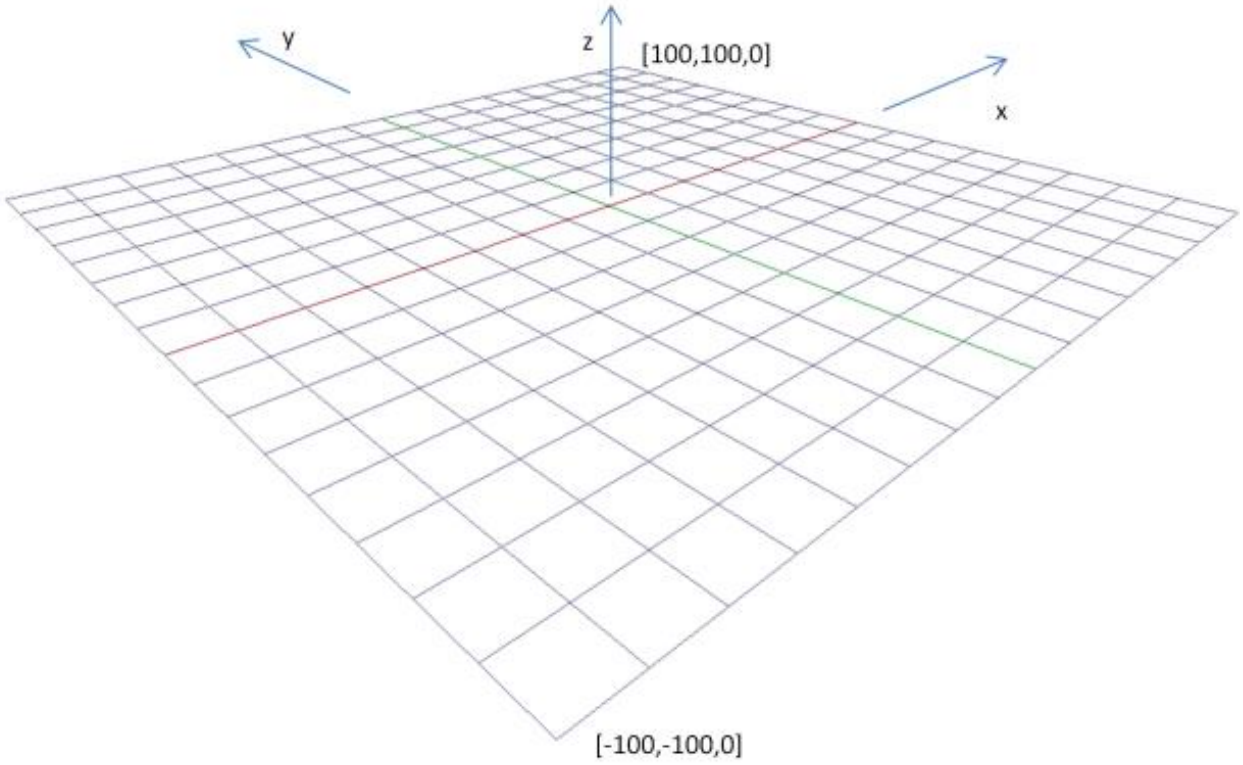
Model Transformation

- Start with Model Coordinate System (MCS)



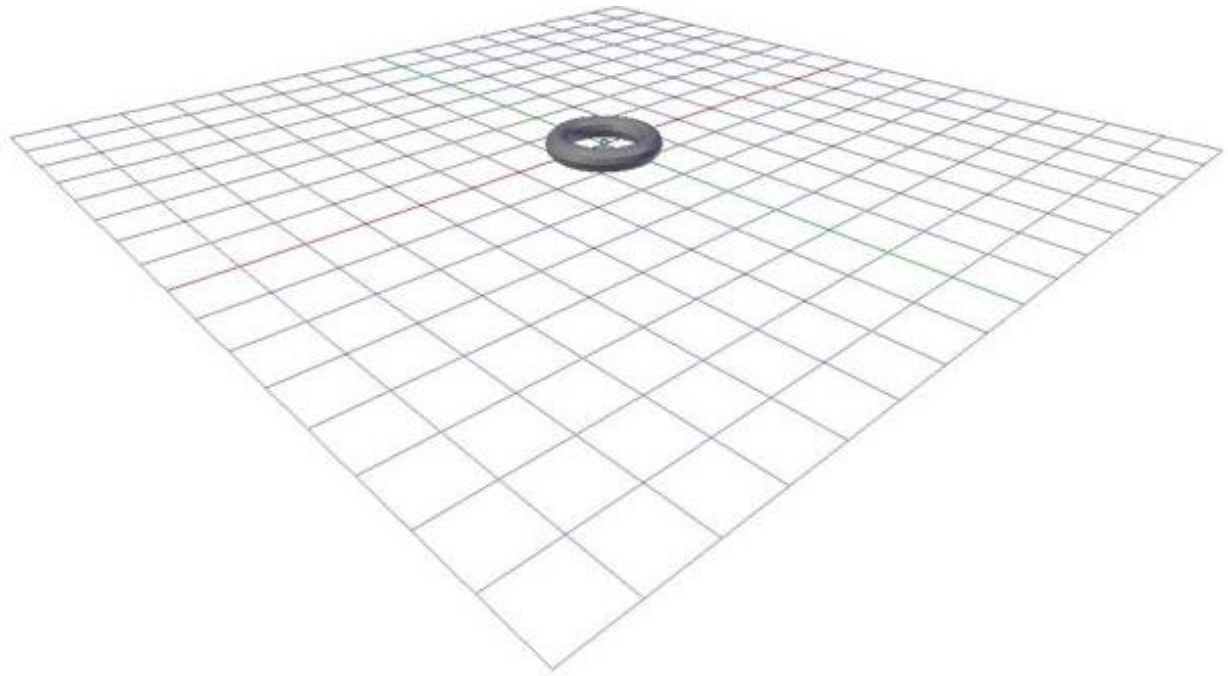
Model Transformation (2)

- Need a common coordinate system for all models: World Coordinate System (WCS)



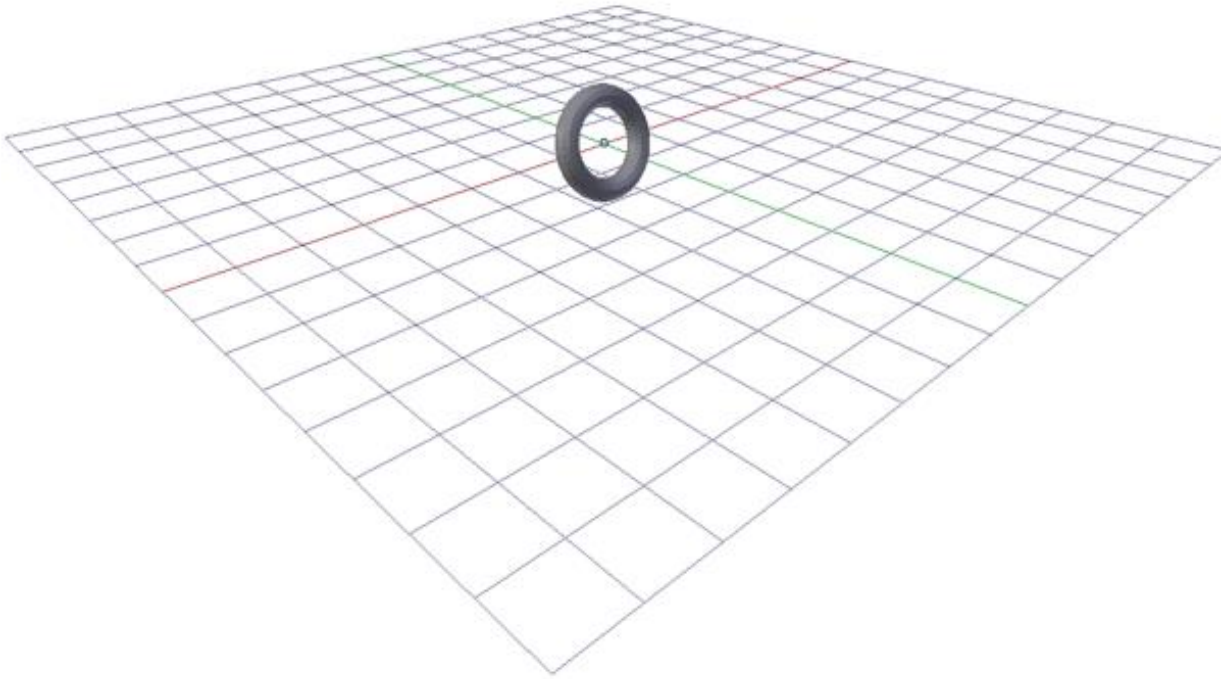
Model Transformation (3)

- First scale model so that it takes appropriate size for WCS



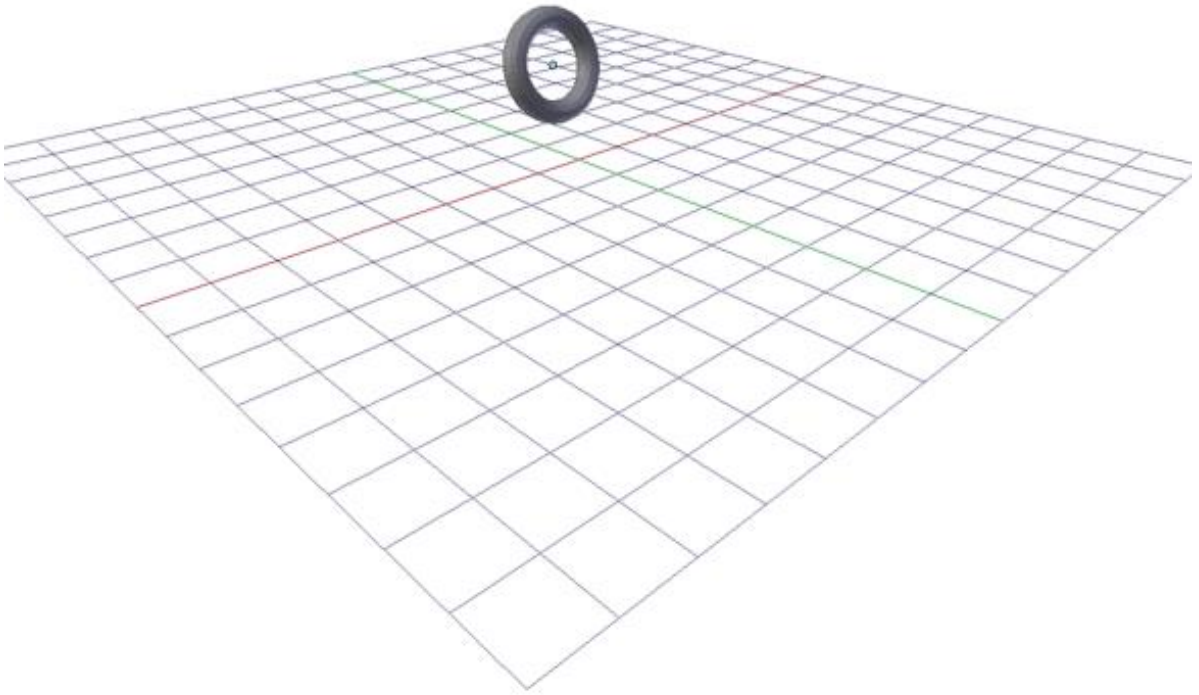
Model Transformation (4)

- Next rotate model so that it has appropriate orientation for WCS



Model Transformation (5)

- Finally translate model so that it is at the appropriate location in WCS

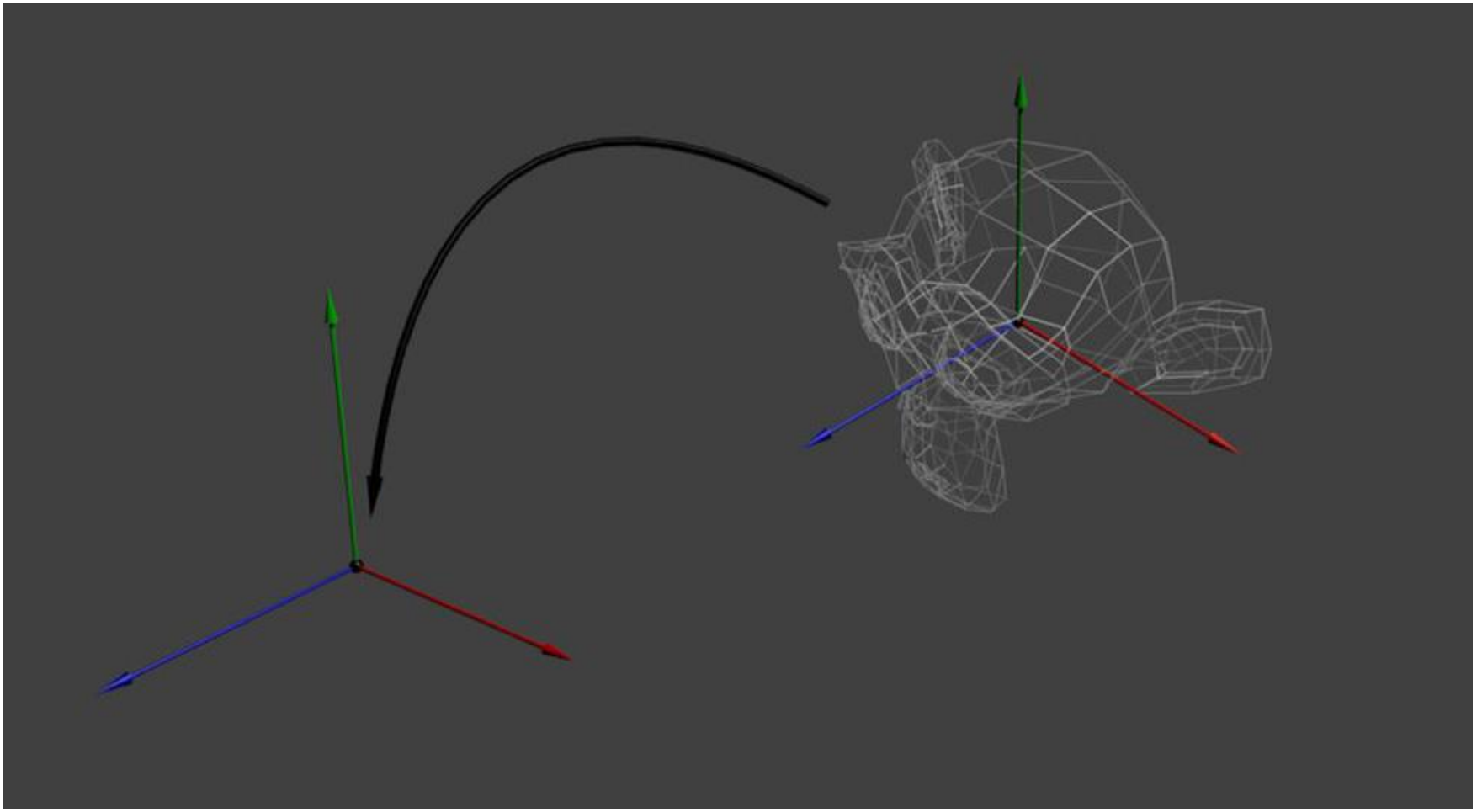


Model Transformation (6)

- Final Matrix

$$M_{Model} = M_{Translate} * M_{Rotate} * M_{Scale}$$

Model Transformation (7)



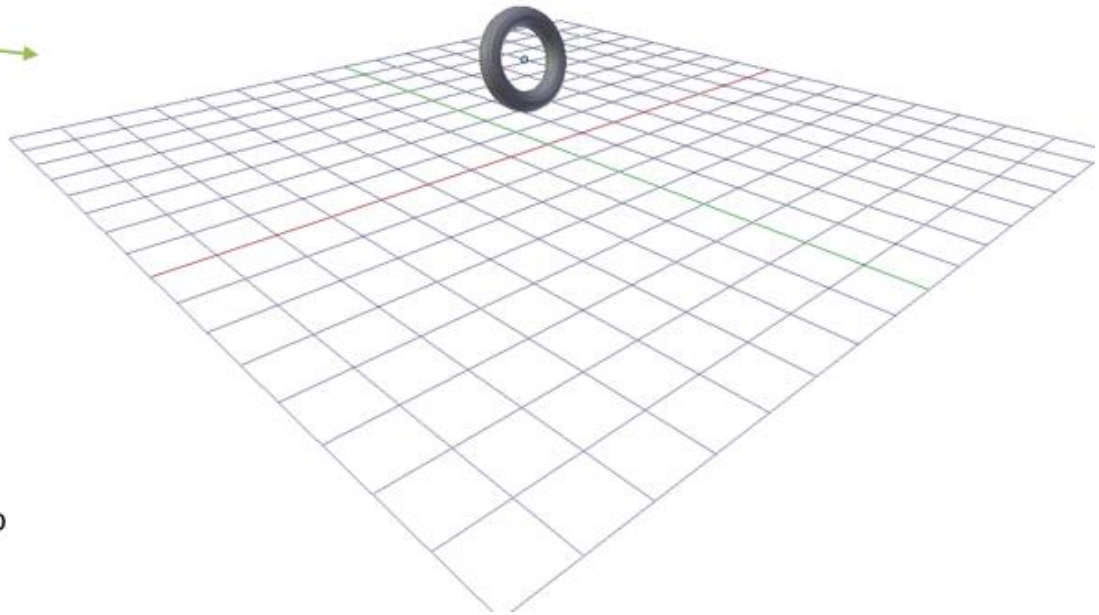
Viewing Transformation (1)

- We define a camera

  At some position, direction in world space

  Also need to define what is "up"

(ECS)

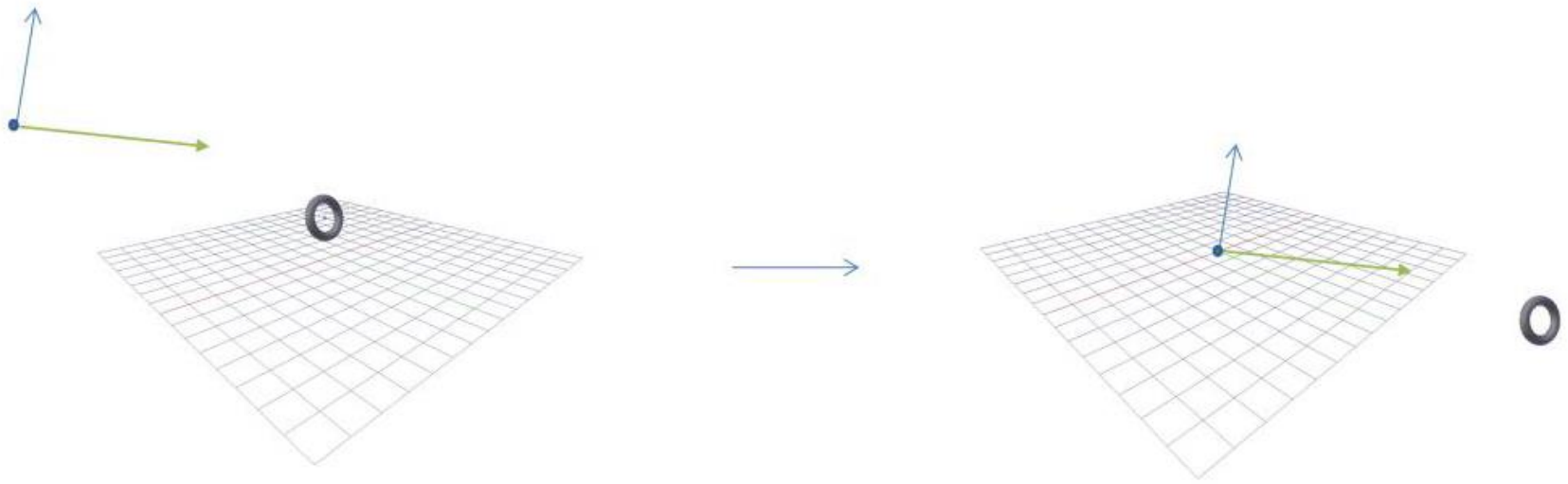


Problem:

Arbitrary camera position, direction, up makes for nasty math down the line

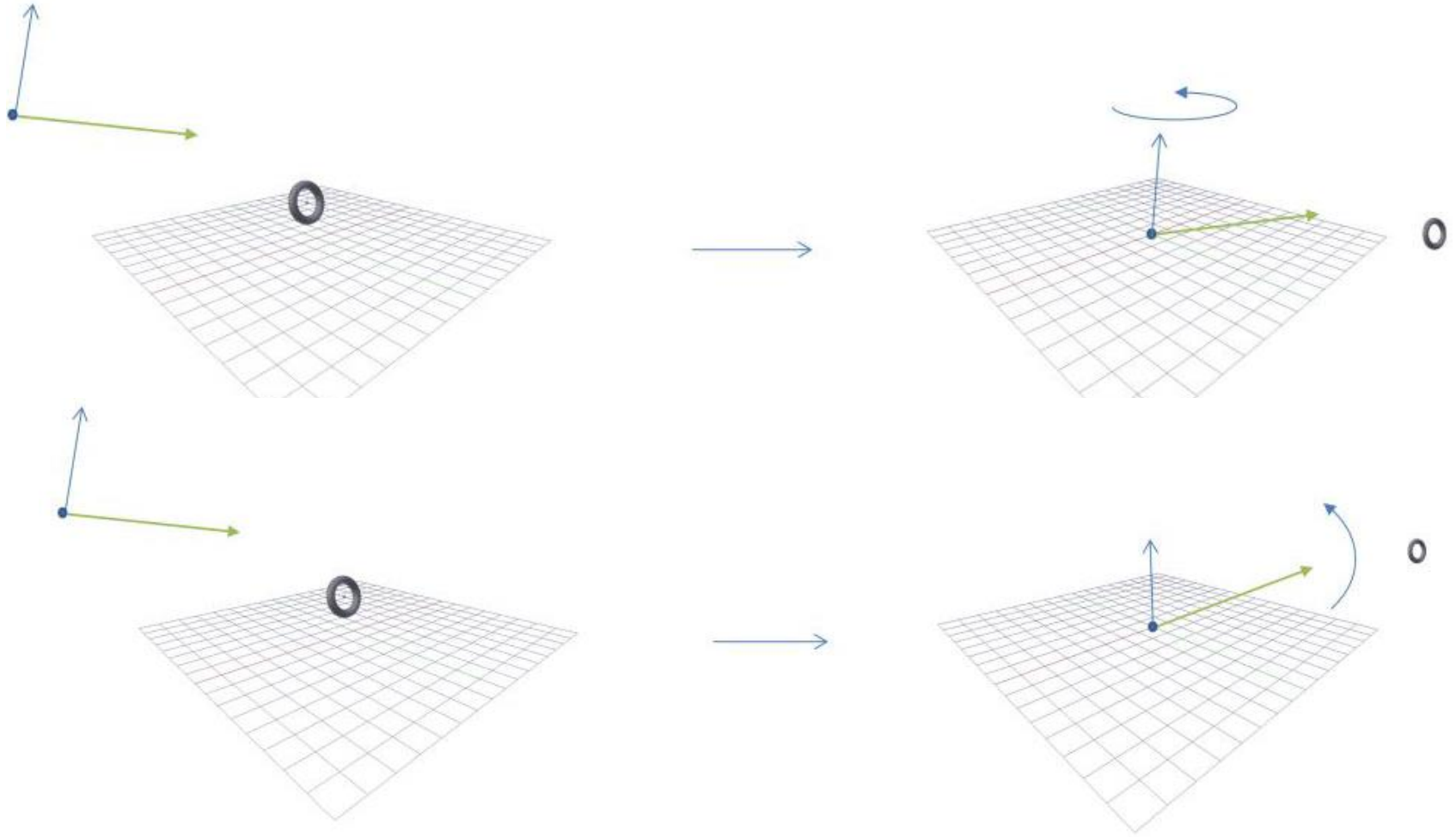
Viewing Transformation (2)

- First, translate every object (including camera) so that the camera is at the origin
- E.g. if the camera is at WCS $[-5,+6,-7]^T$ then translate all objects by $[+5,-6,+7]^T$



Viewing Transformation (3)

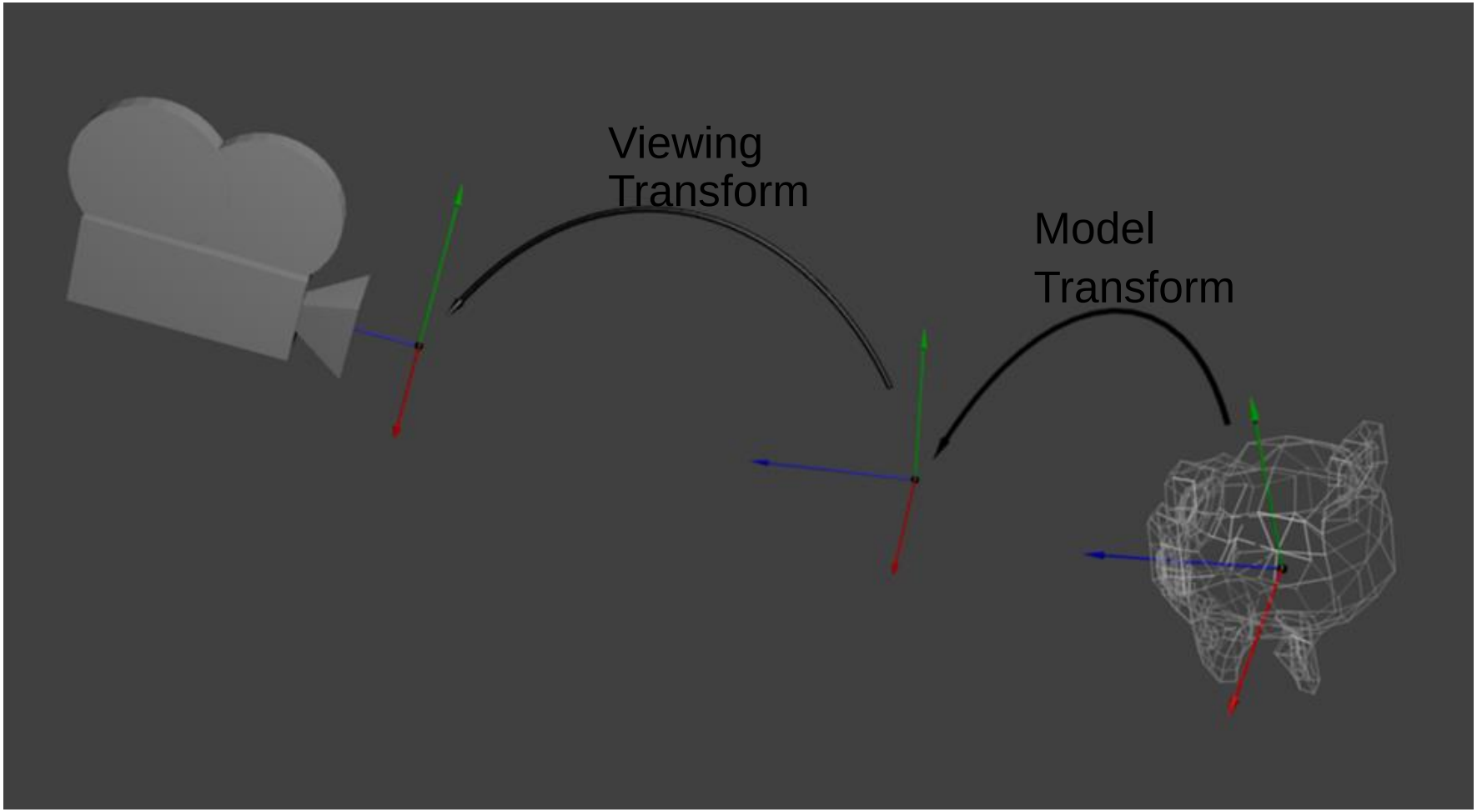
- Next, rotate every object (including camera) so that the camera looks down –Z axis and +Y is the ‘up’ axis



Viewing Transformation (4)

$$M_{View} = M_{-CameraRotate} * M_{-CameraTranslate}$$

Viewing Transformation (5)



Viewing Transformation: details (6)

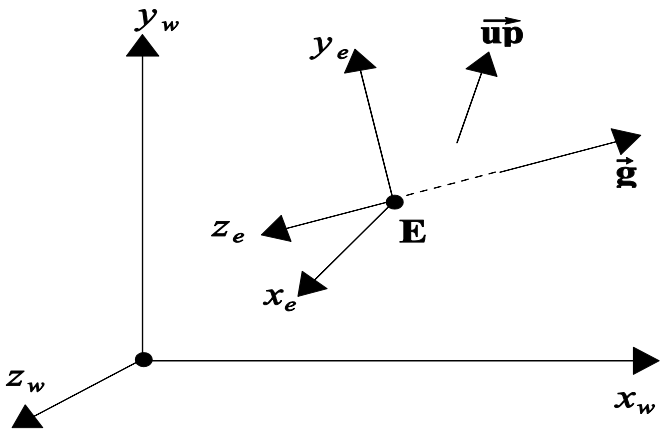
- ECS can be defined within the WCS by:
  - The ECS origin **E**
  - The direction of view $\vec{\mathbf{g}}$
  - The up direction $\overrightarrow{\mathbf{up}}$
- The origin **E** represents the point of view (location of an observer)
- The $\overrightarrow{\mathbf{up}}$ vector
  - defines the up direction
  - need not be perpendicular to $\vec{\mathbf{g}}$
- WCS to ECS conversion steps:
  1. Define the ECS axes x_e, y_e and z_e
  2. Perform the WCS-to-ECS conversion for all objects

Viewing Transformation: details (7)

1. Define the ECS axes

- Sufficient information ($\mathbf{E}, \overrightarrow{\mathbf{up}}, \vec{\mathbf{g}}$) since the coordinate system is right-handed
- Align x_e- and y_e-axes with the WCS axes with the convention:
  - x_e is the horizontal axis & increases to the right
  - y_e is the vertical axis & increases upwards
  - z_e-axis points towards the observer (right-handed ECS) $\overrightarrow{\mathbf{z}_e} = -\vec{\mathbf{g}}$
  - Compute x_e- and y_e-axes as cross products: $\overrightarrow{\mathbf{x}_e} = \overrightarrow{\mathbf{up}} \times \overrightarrow{\mathbf{z}_e}$

$$\overrightarrow{\mathbf{y}_e} = \overrightarrow{\mathbf{z}_e} \times \overrightarrow{\mathbf{x}_e}$$



Viewing Transformation: details (8)

2. WCS-to-ECS conversion
   i. Establish $\mathbf{M}_{\mathbf{WCS}\rightarrow\mathbf{ECS}}$ matrix
   ii. Pre-multiply all object vertices by it

- From [Ex. 3.16] this conversion requires 2 transformations:
  - Translation by $-\vec{\mathbf{E}} = [E_x, E_y, E_z]^T$
  - Change of basis (rotational transformation)
- Let the WCS coordinates of the ECS unit axis vectors be:

$$\mathbf{x_e} = [a_x, a_y, a_z]^T \ , \ \mathbf{y_e} = [b_x, b_y, b_z]^T \text{ and } \mathbf{z_e} = [c_x, c_y, c_z]^T$$

- Then:

$$\mathbf{M}_{\mathbf{WCS}\rightarrow\mathbf{ECS}} = \begin{bmatrix} a_x & a_y & a_z & 0 \\ b_x & b_y & b_z & 0 \\ c_x & c_y & c_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -E_x \\ 0 & 1 & 0 & -E_y \\ 0 & 0 & 1 & -E_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Projection Transform

Projections

- Projection: techniques for the creation of the image of an object onto another simpler object (e.g. line, plane, surface)
- **Projection lines** are defined by:
  - Center of projection
  - Points on the image being projected
- The intersection of a projector with the simpler object (e.g. the plane of projection) forms the image of a point of the original object.
- Projections can be defined in spaces of arbitrary dimension.
- In Computer Graphics & Visualization:
  - Projections are from 3D space onto 2D space
  - The 2D space is referred to as the **plane of projection** & models the output device

Projections(2)

- Two projections are of interest:
  - **Perspective**: the distance of the center of projection from the plane of projection is **finite**
  - **Parallel:** the distance of the center of projection from the plane of projection is **infinite**
- Projective mappings are **not** affine transformations, so they cannot be described by affine transformation matrices.

Differences between affine transformations and projective mappings:

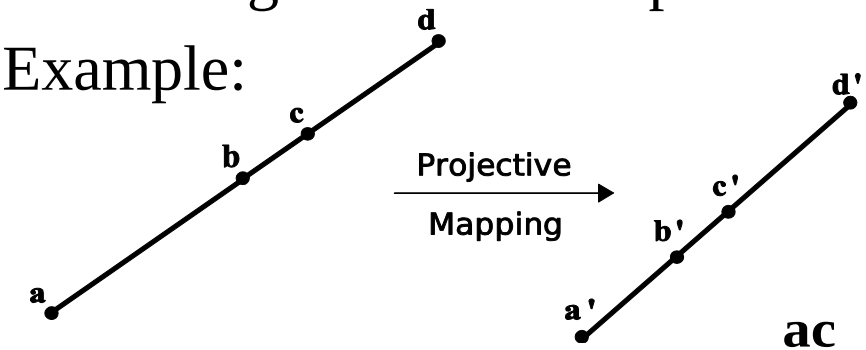
Property preserved	Affine	Projective
Angles	No	No
Distances	No	No
Ratios of distances	Yes	No
Parallel lines	Yes	No
Affine combinations	Yes	No
Straight lines	Yes	Yes
Cross ratios	Yes	Yes

Projections(3)

- Parallel lines: are *not* projected onto parallel lines

  their projections seem to meet at a vanishing point
- Parallel lines are projected onto parallel lines *only when* their plane is parallel to the plane of projection
- A straight line will map to a straight line

Example:



- Ratios of distances are **not** preserved:

$$\frac{\mathbf{ab}}{\mathbf{bd}} \neq \frac{\mathbf{a'b'}}{\mathbf{b'd'}}$$

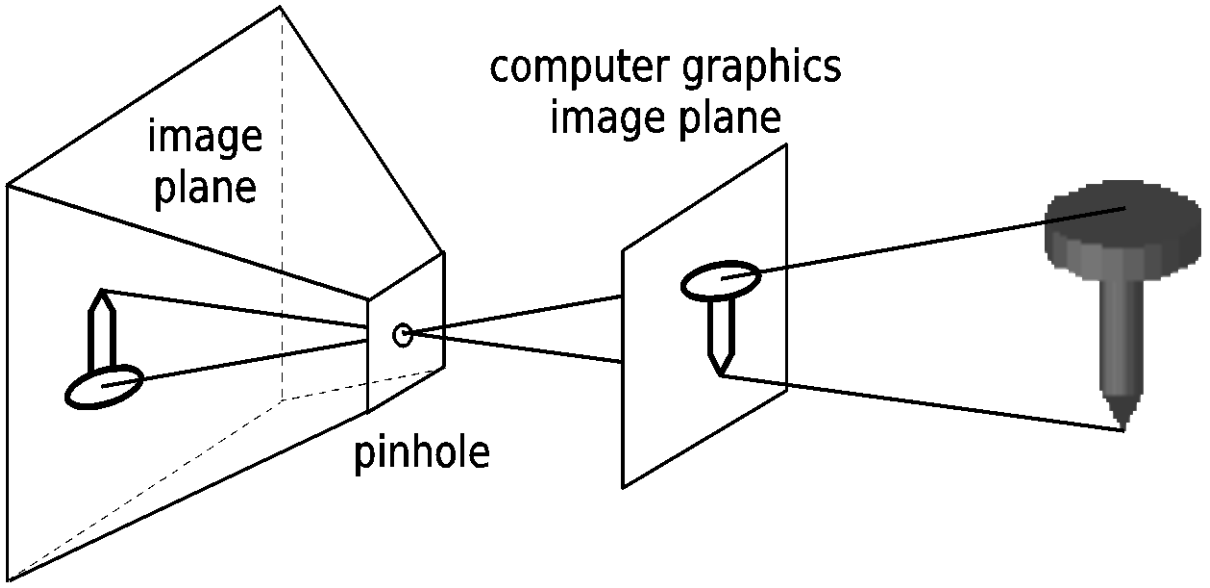
- Cross ratios are preserved:

$$\frac{\dfrac{\mathbf{ac}}{\mathbf{cd}}}{\dfrac{\mathbf{ab}}{\mathbf{bd}}} = \frac{\dfrac{\mathbf{a'c'}}{\mathbf{c'd'}}}{\dfrac{\mathbf{a'b'}}{\mathbf{b'd'}}}$$

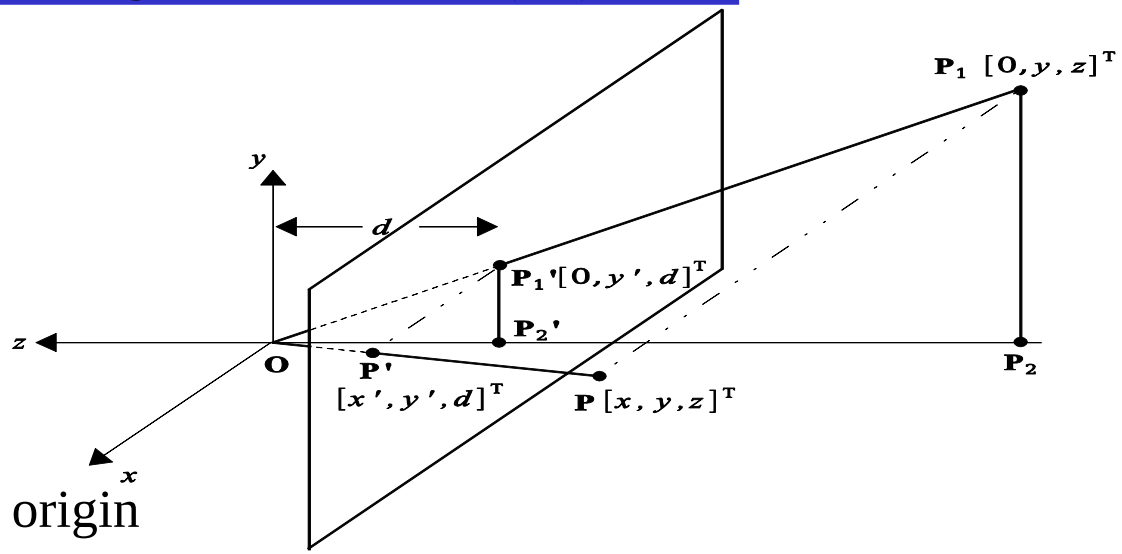
- To fully describe the projective image of a line we need the image of 3 points on the line (affine transformations need only 2)

Perspective Projection

- Models the viewing system of our eyes
- Can be abstracted by a pinhole camera:
  - **Center of projection**: the pinhole
  - **Plane of projection**: the image plane (where the image is formed)
  - Creates an inverted image
  - Derive an upright image by placing the image ‘in front’ of the pinhole



Perspective Projection (2)



- Suppose that:
  - Center of projection is the origin
  - Plane of projection is perpendicular to the negative z-axis at distance d from the center
- A point $\mathbf{P} = [x, y, z]^T$ is projected onto $\mathbf{P}' = [x', y', d]^T$
- $\mathbf{P}_1$ and $\mathbf{P}_1'$ are the projections of $\mathbf{P}$ and $\mathbf{P}'$ onto the *yz*-plane
- $\triangle \mathbf{OP}_1\mathbf{P}_2$ and $\triangle \mathbf{OP}_1'\mathbf{P}_2'$ are similar so: $\dfrac{\mathbf{P}_1'\mathbf{P}_2'}{\mathbf{OP}_2'} = \dfrac{\mathbf{P}_1\mathbf{P}_2}{\mathbf{OP}_2} \Rightarrow \dfrac{y'}{d} = \dfrac{y}{z} \Rightarrow y' = \dfrac{d \cdot y}{z}$
- Similarly : $x' = \dfrac{d \cdot x}{z}$
- The above are not linear equations (division by *z*)

Perspective Projection (3)

- Trick to express the perspective-projection equations in matrix form:

  Use matrix $$\mathbf{P}_{\mathbf{PER}} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{bmatrix}$$

  which alters the homogeneous coordinate and

  maps the coordinates of a point $[x, y, z]^T$ as follows:

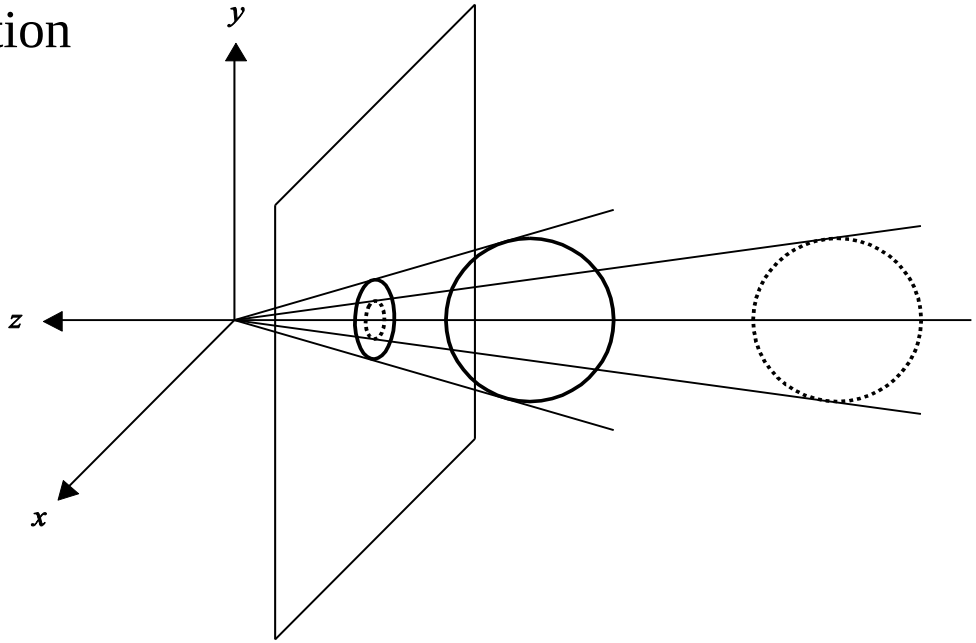
$$\mathbf{P}_{\mathbf{PER}} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \cdot d \\ y \cdot d \\ z \cdot d \\ z \end{bmatrix}$$

- Divide by the homogeneous coordinate: $\begin{bmatrix} x \cdot d \\ y \cdot d \\ z \cdot d \\ z \end{bmatrix} / z = \begin{bmatrix} \frac{x \cdot d}{z} \\ \frac{y \cdot d}{z} \\ d \\ 1 \end{bmatrix}$

  Known as perspective division

Perspective Projection (4)

- Perspective shortening:
  - The size of the projection of an object is inversely proportional to its distance from the center of projection



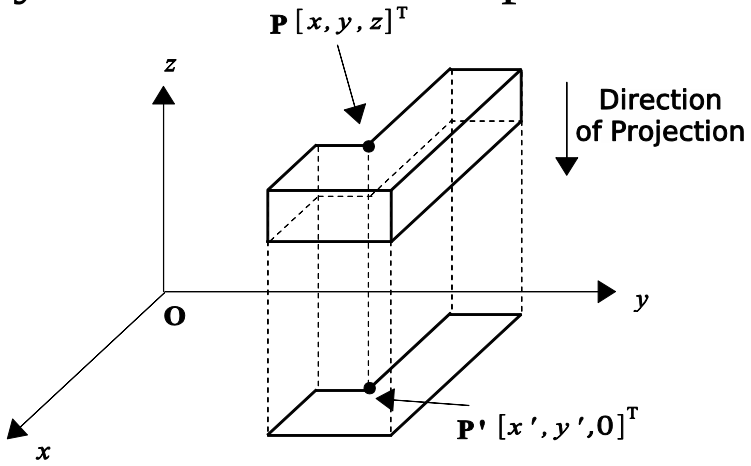
- Known in ancient times, then forgotten
- Leonardo da Vinci studied the laws of perspective
- Older paintings had no perspective (symbolic criteria prevailed)

Parallel Projection

- Center of projection is at **infinite** distance from the projection plane
- Projection lines are **parallel** to each other
- To describe parallel projection one must specify:
  - The direction of projection (a vector)
  - The plane of projection
- Two types of parallel projections:
  - **Orthographic**: the direction of projection is normal to the plane of projection
  - **Oblique**: the direction of projection is not necessarily normal to the plane of projection

Orthographic Projection

- Usually employs one of the main planes as the plane of projection

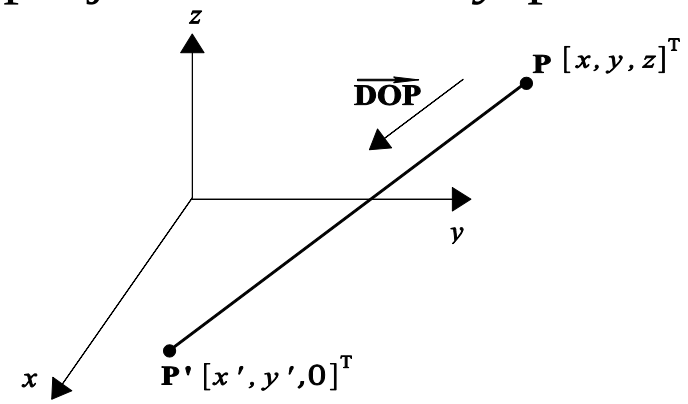


- Suppose the *xy*-plane is used
- A point $\mathbf{P} = [x, y, z]^T$ will be projected onto $\mathbf{P}' = [x', y', z']^T = [x, y, 0]^T$ as follows:

$$\mathbf{P}' = \mathbf{P}_{\mathbf{ORTHO}} \cdot \mathbf{P} \qquad \text{where} \qquad \mathbf{P}_{\mathbf{ORTHO}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Oblique Projection

- Let the direction of projection be: $\overrightarrow{\mathbf{DOP}} = [DOP_x, DOP_y, DOP_z]^T$ and the plane of projection be the *xy*-plane



- The projection $\mathbf{P}' = [x', y', z']^T$ of a point $\mathbf{P} = [x, y, z]^T$ will be

$$\mathbf{P}' = \mathbf{P} + \lambda \cdot \overrightarrow{\mathbf{DOP}} \quad (1) \quad \text{for some scalar } \lambda$$

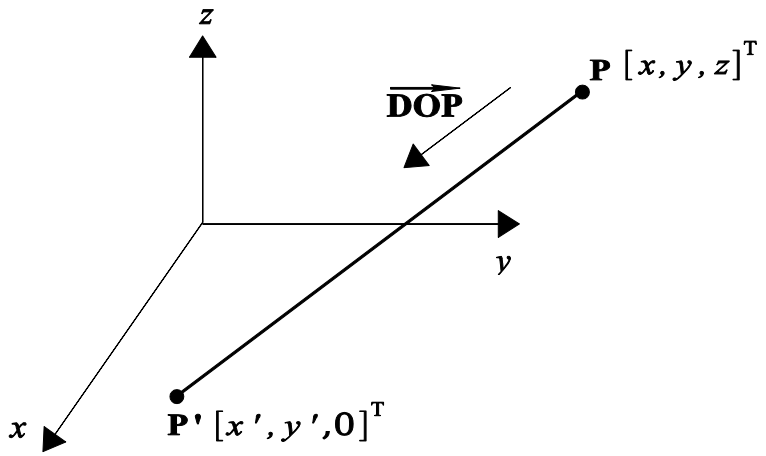
- The *z*-coordinate of $\mathbf{P}'$ is 0, so (1) becomes:

$$0 = z + \lambda \cdot DOP_z \text{ or } \lambda = -\frac{z}{DOP_z}$$

- The other 2 coordinates of $\mathbf{P}'$ are:

$$x' = x + \lambda \cdot DOP_x = x - \frac{DOP_x}{DOP_z} \cdot z \text{ and } y' = y - \frac{DOP_y}{DOP_z} \cdot z$$

Oblique Projection (2)



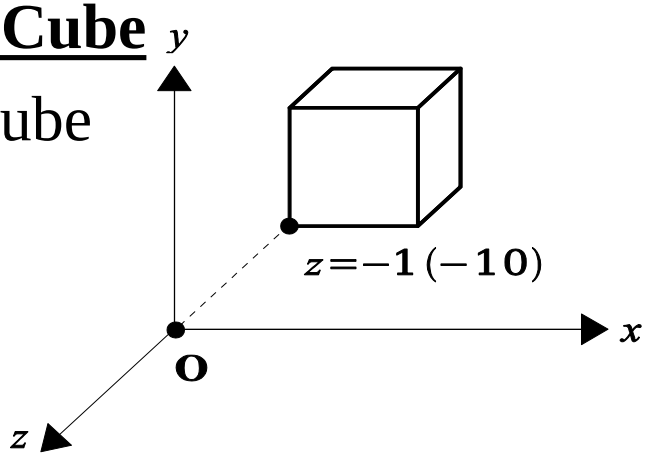
- Matrix form:

$$\mathbf{P}_{\mathbf{OBLIQUE}}(\overrightarrow{\mathbf{DOP}}) = \begin{bmatrix} 1 & 0 & -\frac{DOP_x}{DOP_z} & 0 \\ 0 & 1 & -\frac{DOP_y}{DOP_z} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- $\mathbf{P}' = \mathbf{P}_{\mathbf{OBLIQUE}}(\overrightarrow{\mathbf{DOP}}) \cdot \mathbf{P}$

Projection Example 1

EXAMPLE 1: **Perspective Projection of a Cube**



Determine the perspective projections of a cube of side 1 when

(a) the plane of projection is $z=-1$ and

(b) the plane of projection is $z=-10$.

The cube is placed on the plane of projection.

SOLUTION (a)

- Represent the vertices of the cube as a 4×8 matrix:

$$\mathbf{C} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ -1 & -1 & -1 & -1 & -2 & -2 & -2 & -2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

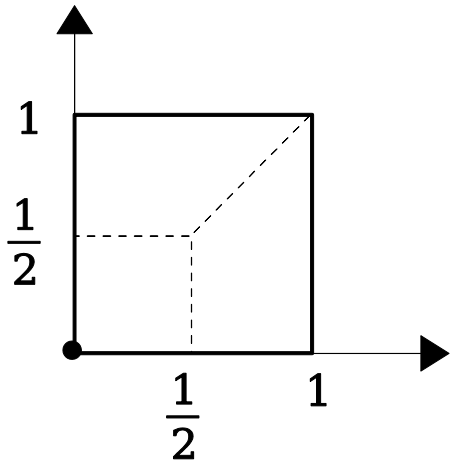
Projection Example 1 (2)

- Multiply the perspective projection matrix (d=-1) by $\boldsymbol{C}$:

$$\mathbf{P}_{\mathbf{PER}} \cdot \mathbf{C} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \mathbf{C} = \begin{bmatrix} 0 & -1 & -1 & 0 & 0 & -1 & -1 & 0 \\ 0 & 0 & -1 & -1 & 0 & 0 & -1 & -1 \\ 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 \\ -1 & -1 & -1 & -1 & -2 & -2 & -2 & -2 \end{bmatrix}$$

- Normalize by the homogeneous coordinates:

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$



Projection Example 1 (3)

(b) The original cube is:

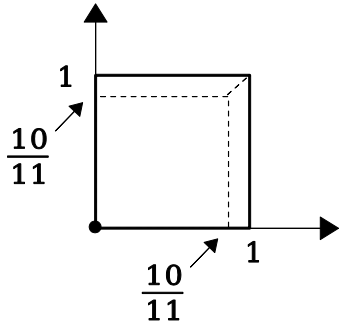
$$\mathbf{C}' = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ -10 & -10 & -10 & -10 & -11 & -11 & -11 & -11 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

- Multiply the perspective projection matrix (d=-10) by **C'**:

$$\begin{bmatrix} -10 & 0 & 0 & 0 \\ 0 & -10 & 0 & 0 \\ 0 & 0 & -10 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \mathbf{C}' = \begin{bmatrix} 0 & -10 & -10 & 0 & 0 & -10 & -10 & 0 \\ 0 & 0 & -10 & -10 & 0 & 0 & -10 & -10 \\ 100 & 100 & 100 & 100 & 110 & 110 & 110 & 110 \\ -10 & -10 & -10 & -10 & -11 & -11 & -11 & -11 \end{bmatrix}$$

- Normalize by the homogeneous coordinates:

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & \frac{10}{11} & \frac{10}{11} & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & \frac{10}{11} & \frac{10}{11} \\ -10 & -10 & -10 & -10 & -10 & -10 & -10 & -10 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$



Projection Example 2

EXAMPLE 2: **Perspective Projection onto an Arbitrary Plane**

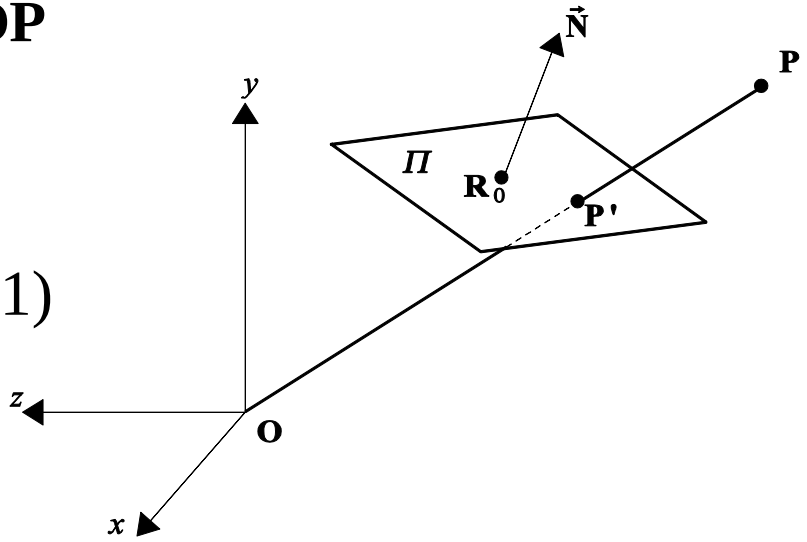
Compute the perspective projection of a point $\mathbf{P}=[x, y, z]^T$ onto the arbitrary plane $\mathbf{\Pi}$ which is specified by the point $\mathbf{R_0}=[x_0,y_0,z_0]^T$ and a normal vector $\vec{\mathbf{N}} = [n_x, n_y, n_z]^T$. The center of projection is **O**.

SOLUTION

- $\mathbf{P}'=[x', y', z']^T$ is the projection of $\mathbf{P}=[x, y, z]^T$
- $\overrightarrow{\mathbf{OP}'}$ and $\overrightarrow{\mathbf{OP}}$ are collinear so $\overrightarrow{\mathbf{OP}'} = a \cdot \overrightarrow{\mathbf{OP}}$ for some scalar a
- In the projection equations:

$$x' = ax \ , \ y' = ay \ , \ z' = az \quad (1)$$

scalar a must be determined.



Projection Example 2 (2)

- Vector $\overrightarrow{\mathbf{R_0P'}}$ is on the plane of projection, so :

$$\vec{N}\cdot\overrightarrow{\mathbf{R_0P'}} = 0$$

$$or\ n_x(x'-x_0)+n_y(y'-y_0)+n_z(z'-z_0)=0$$

$$or\ n_x x'+n_y y'+n_z z' = n_x x_0+n_y y_0+n_z z_0$$

- Substitute the values of x, y, z from (1), set $c = n_x x_0 + n_y y_0 + n_z z_0$ and solve for a: $$a = \frac{c}{n_x x + n_y y + n_z z}$$

- The projection equations include a division by a combination of x,y,z
- Put in matrix form by altering the homogeneous coordinate:

$$\mathbf{P}_{\mathbf{PER},\Pi} = \begin{bmatrix} c & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & c & 0 \\ n_x & n_y & n_z & 0 \end{bmatrix}$$

Projection Example 3

EXAMPLE 3: **Oblique projection with Azimuth & Elevation Angles**

Sometimes (particularly in architectural design), oblique projections are specified in terms of the azimuth and elevation angles φ and θ that define the relation of the direction to the plane of projection. Determine the projection matrix in this case.

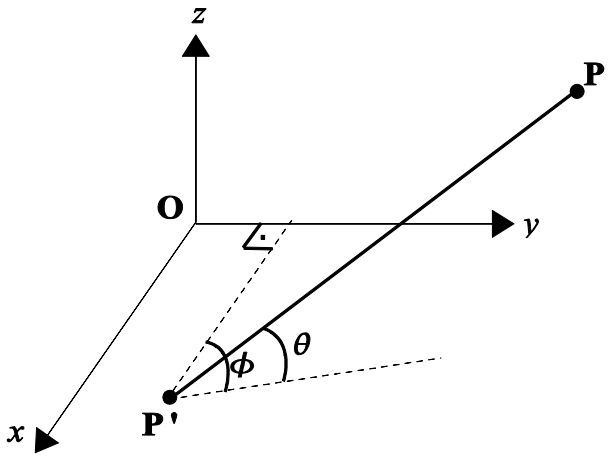
SOLUTION

- Let *xy* be the plane of projection
- The direction of the projection vector is:

$$\overrightarrow{\mathbf{DOP}} = [\cos\theta\cos\phi, \cos\theta\sin\phi, \sin\theta]^T$$

- So:

$$\mathbf{P}_{\mathbf{OBLIQUE}}(\phi,\theta) = \begin{bmatrix} 1 & 0 & -\frac{\cos\phi}{\tan\theta} & 0 \\ 0 & 1 & -\frac{\sin\phi}{\tan\theta} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Projection Example 4

EXAMPLE 4: **Oblique projection onto an Arbitrary Plane**

Determine the oblique projection mapping onto an arbitrary plane $\mathbf{\Pi}$ that is specified by a point $\mathbf{R_0}=[x_0,y_0,z_0]^T$ and a normal vector $\vec{\mathbf{N}} = [n_x, n_y, n_z]^T$. The direction of projection is given by the vector $\overrightarrow{\mathbf{DOP}} = [DOP_x, DOP_y, DOP_z]^T$

SOLUTION

Transform plane $\mathbf{\Pi}$ to coincide with the *xy*-plane, use the oblique projection matrix, undo the first transform:

- Step 1: Translate $\mathbf{R_0}$ to the origin, $\mathbf{T}(-\overrightarrow{\mathbf{R_0}})$
- Step 2: Align $\vec{\mathbf{N}}$ with the positive z-axis (by using $\mathbf{A}(\vec{\mathbf{N}})$ from [Ex 3.12])
- Step 3: Use the oblique projection matrix with the direction of projection transformed according to previous steps:

$$\overrightarrow{\mathbf{DOP}}' = \mathbf{A}(\vec{\mathbf{N}})\cdot\mathbf{T}(-\overrightarrow{\mathbf{R_0}})\cdot\overrightarrow{\mathbf{DOP}}$$

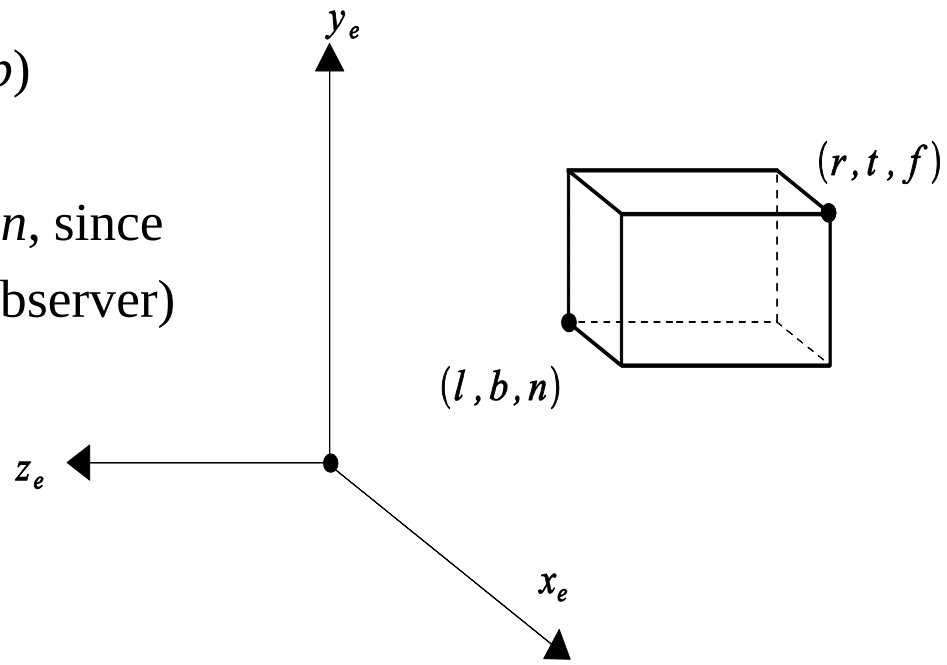
Projection Example 4 (2)

- Step 4: Undo the alignment $\mathbf{A}(\vec{\mathbf{N}})^{-1}$
- Step 5: undo the translation $\mathbf{T}(\overrightarrow{\mathbf{R}_0})$
- Finally:

$$\mathbf{P}_{\mathbf{OBLIQUE},\Pi}(\overrightarrow{\mathbf{DOP}}) = \mathbf{T}(\overrightarrow{\mathbf{R}_0})\cdot\mathbf{A}(\vec{\mathbf{N}})^{-1}\cdot\mathbf{P}_{\mathbf{OBLIQUE}}(\overrightarrow{\mathbf{DOP}}')\cdot\mathbf{A}(\vec{\mathbf{N}})\cdot\mathbf{T}(-\overrightarrow{\mathbf{R}_0})$$

ECS to CSS: Orthographic projection

- Orthographic projection onto the *xy*-plane:
  - Select a region of space (**view volume**) that will be mapped to CSS
    - rectangular parallelepiped
    - can be defined by 2 opposite vertices
  - $x_e = l$, the left clip plane
  - $x_e = r$, the right clip plane, ($r > l$)
  - $y_e = b$, the bottom clip plane
  - $y_e = t$, the top clip plane, ($t > b$)
  - $z_e = n$, the near clip plane
  - $z_e = f$, the far clip plane, ($f < n$, since the z_e axis points toward the observer)



ECS to CSS: Orthographic projection(2)

- To maintain the *z*-coordinate use the ortho. proj. $\mathbf{M_{ORTHO} = ID}$
- Converting the view volume into CSS:
  - Translation & Scaling
  - Map the (*l*, *b*, *n*) values to -1 and (*r*, *t*, *f*) values to 1

$$\mathbf{M}_{\mathbf{ECS\rightarrow CSS}}^{\mathbf{ORTHO}} = \mathbf{S}\left(\frac{2}{r-l}, \frac{2}{t-b}, \frac{2}{f-n}\right)\cdot\mathbf{T}\left(-\frac{r+l}{2}, -\frac{t+b}{2}, -\frac{n+f}{2}\right)\cdot\mathbf{ID} =$$

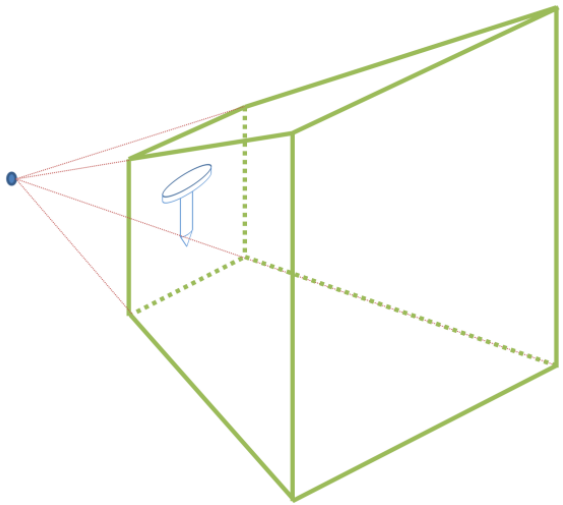
$$\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{f-n} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -\frac{r+l}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -\frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{2}{r-l}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

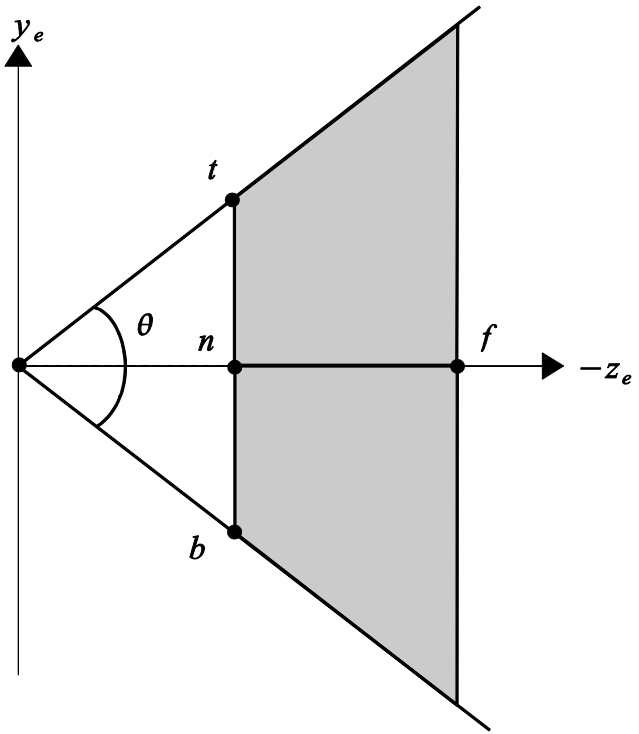
- Thus a WCS point $\mathbf{X_w} = [x_w, y_w, z_w]^T$ is mapped into CSS by:

$$\mathbf{X_s} = \mathbf{M}_{\mathbf{ECS\rightarrow CSS}}^{\mathbf{ORTHO}} \cdot \mathbf{M_{WCS\rightarrow ECS}} \cdot \mathbf{X_w}$$

ECS to CSS: Perspective projection

- View volume is a truncated pyramid, symmetrical about the $-z_e$ axis
- This view volume is specified by:
  - θ , the angle of the field of view in the *y*-direction
  - *aspect*, the width to height ratio of a cross section of the pyramid
  - $z_e = n$, the near clipping plane
  - $z_e = f$, the far clipping plane ($f < n$)





ECS to CSS: Perspective projection (2)

- Projection takes place onto the near clipping plane $z_e = n$
- Compute the other clipping boundaries:
  - → top: $t = |n| \cdot \tan(\frac{\theta}{2})$
  - → bottom: $b = -t$
  - → right: $r = t \cdot aspect$
  - → left: $l = -r$
- Use a modified version of the perspective projection matrix:

$$\mathbf{P}_{\mathbf{PER}} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

- *z*-coordinate :
  - Preserve it for hidden surface & other computations in screen space
  - Simply keeping z_e will deform objects
  - Use a mapping that preserves lines & planes

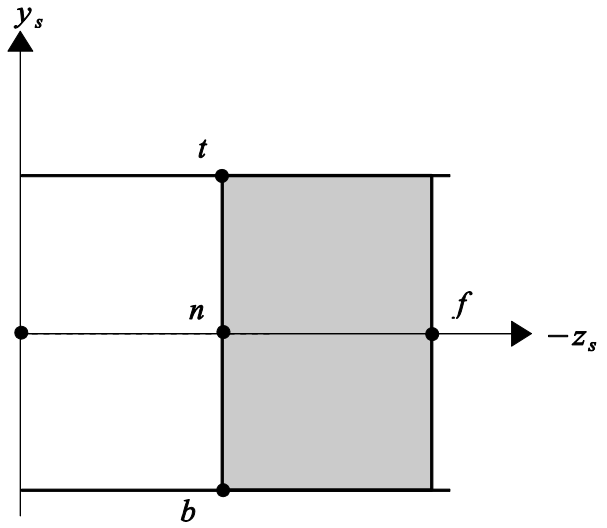
    $z_s = A + B / z_e$, where A,B are constants

ECS to CSS: Perspective projection (3)

- Require that $(z_e = n) \rightarrow (z_s = n)$ and $(z_e = f) \rightarrow (z_s = f)$ so
  $A = (n + f)$ and $B = -nf$
- This mapping will not alter the boundaries $z_e = n$, $z_e = f$ but this will not be true for z_e values between them
- Thus the perspective projection matrix is:

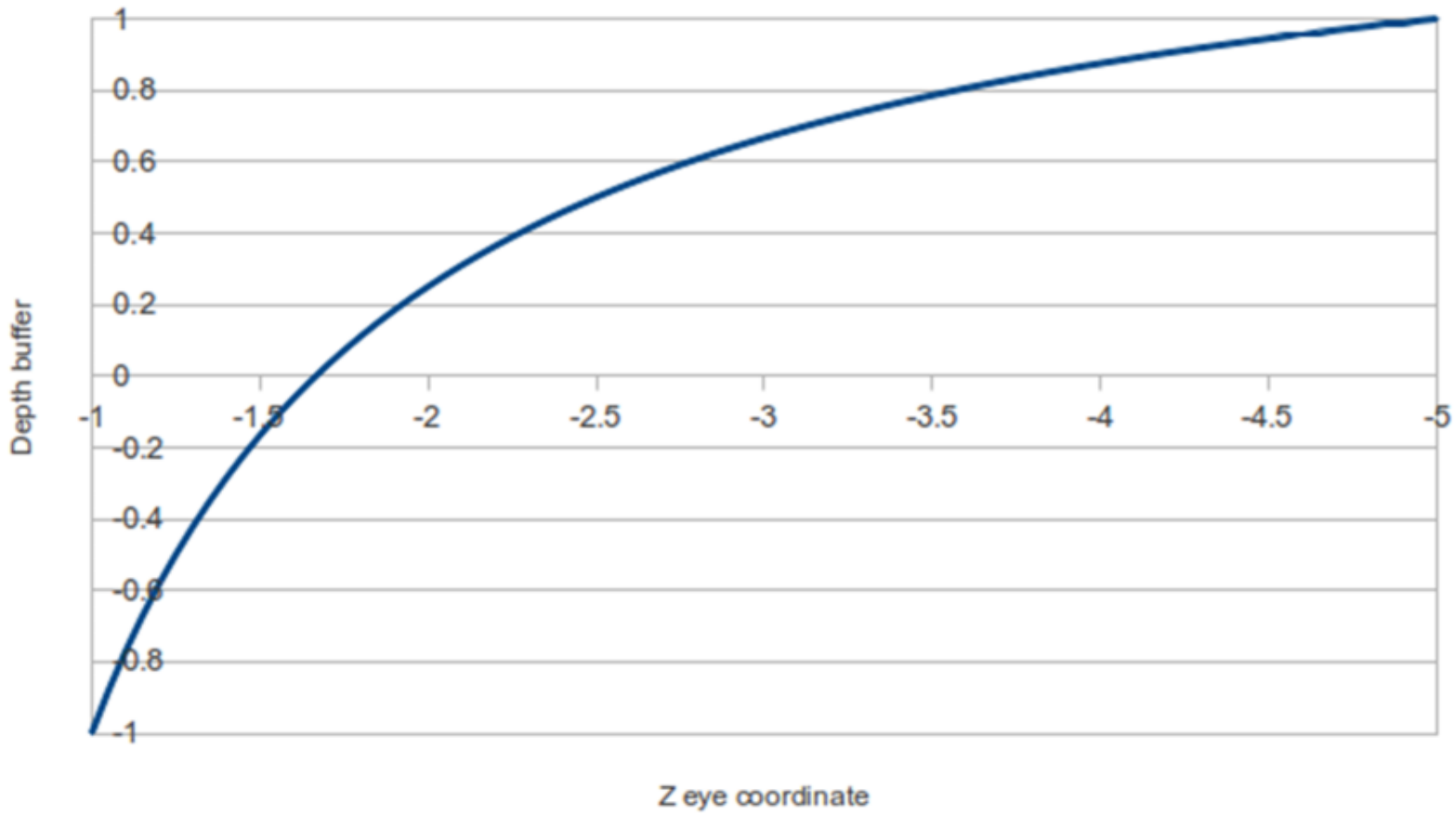
$$\mathbf{P}_{\mathbf{VT}} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -nf \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

  - Makes the w-coordinate equal to z_e
  - Must be followed by a division by z_e (**perspective division**)
  - Clipping boundaries are not affected
  - Transforms the truncated pyramid into a rectangular parallelepiped :



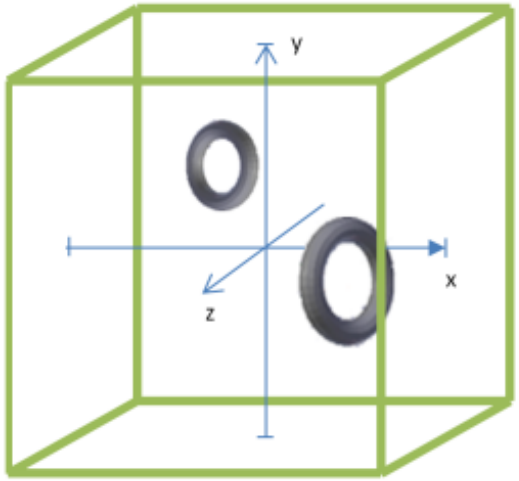
ECS to CSS: Perspective projection (4)

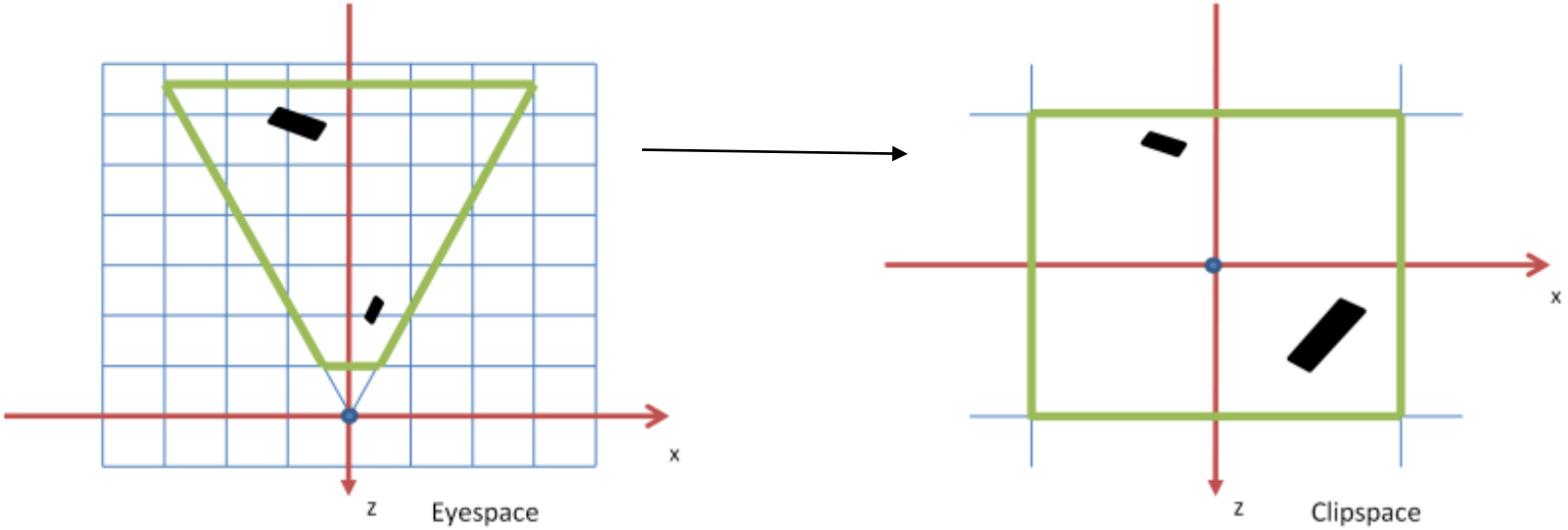
- Numerical effect of z transformation



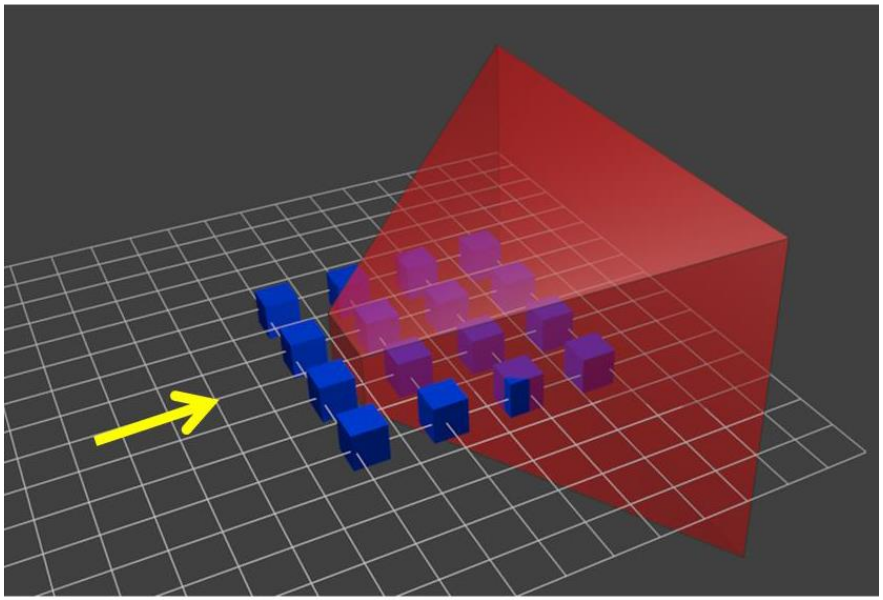
ECS to CSS: Perspective projection (5)

Clipspace

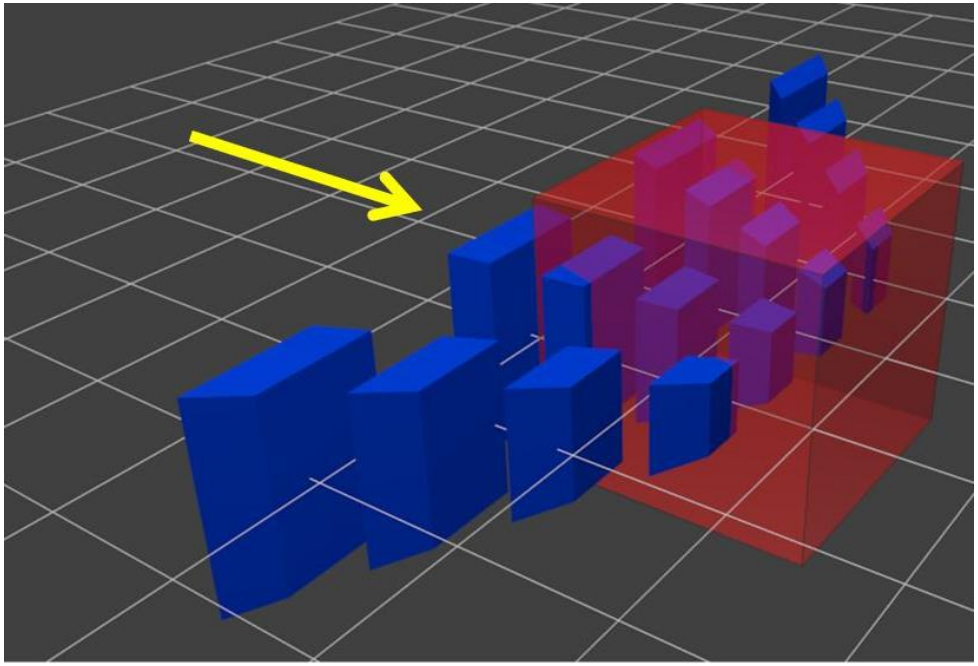




ECS to CSS: Perspective projection (6)



Before



After

ECS to CSS: Perspective projection (7)

- Situation similar to the setting before the orthographic projection
- This view volume is already symmetrical about the $-z_e$ axis
- ECS-to-CSS conversion steps:
  1. $\mathbf{P_{VT}}$
  2. translation along z_e
  3. scaling

$$\mathbf{M}_{\mathbf{ECS}\rightarrow\mathbf{CSS}}^{\mathbf{PERSP}} = \mathbf{S}\left(\frac{2}{r-l}, \frac{2}{t-b}, \frac{2}{f-n}\right)\cdot\mathbf{T}\left(0, 0, -\frac{n+f}{2}\right)\cdot\mathbf{P}_{\mathbf{VT}} =$$

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

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

ECS to CSS: Perspective projection (8)

- Thus a WCS point $\mathbf{X_w} = [x_w, y_w, z_w]^T$ is converted into CSS by:

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \mathbf{M}^{\mathbf{PERSP}}_{\mathbf{ECS}\rightarrow\mathbf{CSS}} \cdot \mathbf{M}_{\mathbf{WCS}\rightarrow\mathbf{ECS}} \cdot \mathbf{X_w}$$

followed by the perspective division by $w\ (= z_e)$ $\quad \mathbf{x_s} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} / w$

- Frustum culling is performed before the perspective division ensuring that the coordinates of every point on every object are within the clipping bounds:

$$-w \leq x, y, z \leq w$$

- The coordinates of every point are now in the range [-1,1]

ECS to CSS: Perspective projection (9)

Example:

- Take the boundary points with ECS coordinates
  $[l, b, n, 1]^T$ and $[0, 0, f, 1]^T$
- Apply the $\mathbf{P_{VT}}$ matrix:

$$\mathbf{P_{VT}} \cdot \begin{bmatrix} l \\ b \\ n \\ 1 \end{bmatrix} = \begin{bmatrix} ln \\ bn \\ n^2 \\ n \end{bmatrix} \qquad \mathbf{P_{VT}} \cdot \begin{bmatrix} 0 \\ 0 \\ f \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ f^2 \\ f \end{bmatrix}$$

- The homogeneous coordinate is no longer 1
- Apply the scaling & translation matrices:

$$\mathbf{S} \cdot \mathbf{T} \cdot \begin{bmatrix} ln \\ bn \\ n^2 \\ n \end{bmatrix} = \begin{bmatrix} -n \\ -n \\ -n \\ n \end{bmatrix} \qquad \mathbf{S} \cdot \mathbf{T} \cdot \begin{bmatrix} 0 \\ 0 \\ f^2 \\ f \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ f \\ f \end{bmatrix}$$

ECS to CSS: Perspective projection (10)

Example(continued):

- Due to the symmetry of the truncated pyramid about $-z_e$:

  $r = -l, t = -b$ so $r - l = -2l, t - b = -2b$

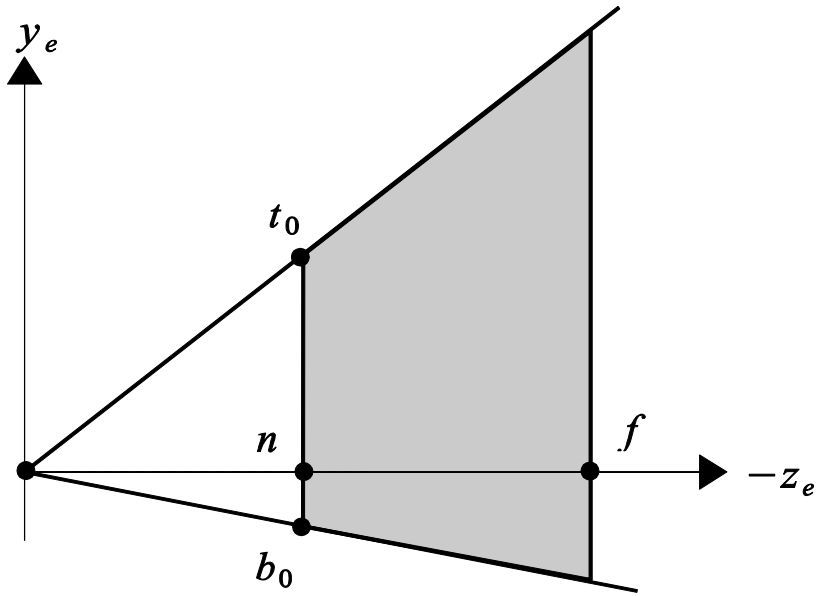
- Perspective division:

$$\begin{bmatrix} -n \\ -n \\ -n \\ n \end{bmatrix} / n = \begin{bmatrix} -1 \\ -1 \\ -1 \\ 1 \end{bmatrix} \qquad \begin{bmatrix} 0 \\ 0 \\ f \\ f \end{bmatrix} / f = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

which are the CSS values of the points

Extended Viewing Transformation

A. Truncated Pyramid Not Symmetrical about z_e-Axis



e.g. stereo viewing where two viewpoints are slightly offset on the x_e –axis

Extended Viewing Transformation(2)

A. Truncated Pyramid Not Symmetrical about z_e-Axis

- Parameters of the clipping planes:
  - $z_e = n_0$, the near clipping plane (as before)
  - $z_e = f_0$, the far clipping plane, $f_0 < n_0$ (as before)
  - $y_e = b_0$, the y_e-coordinate of the bottom clipping plane at its intersection with the near clipping plane
  - $y_e = t_0$, the y_e-coordinate of the top clipping plane at its intersection with the near clipping plane
  - $x_e = l_0$, the x_e-coordinate of the left clipping plane at its intersection with the near clipping plane
  - $x_e = r_0$, the x_e-coordinate of the right clipping plane at its intersection with the near clipping plane

Extended Viewing Transformation(3)

A. Truncated Pyramid Not Symmetrical about z_e-Axis (steps)

- Convert the non-symmetrical pyramid to symmetrical about z_e with a shear transformation on the *xy*-plane

$$\mathbf{SH}_{xy} = \begin{bmatrix} 1 & 0 & A & 0 \\ 0 & 1 & B & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (1)$$

- Determine the A, B parameters
  - Take the shear on the *y*-coordinate: map the midpoint of t_0b_0 to 0

$$\frac{b_0 + t_0}{2} + B \cdot n_0 = 0 \Rightarrow B = -\frac{b_0 + t_0}{2n_0}$$

  - Similarly for the x_e shear factor: $A = -\dfrac{l_0 + r_0}{2n_0}$
- So (1) becomes:

$$\mathbf{SH}_{\mathbf{NON-SYM}} = \begin{bmatrix} 1 & 0 & -\dfrac{l_0 + r_0}{2n_0} & 0 \\ 0 & 1 & -\dfrac{b_0 + t_0}{2n_0} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Extended Viewing Transformation(4)

A. Truncated Pyramid Not Symmetrical about z_e-Axis (steps)

- Change the clipping boundaries, to reflect the symmetrical shape of the new pyramid:

$$n = n_0 \qquad f = f_0$$

$$l = l_0 - \frac{l_0 + r_0}{2} \qquad r = r_0 - \frac{l_0 + r_0}{2}$$

$$b = b_0 - \frac{b_0 + t_0}{2} \qquad t = t_0 - \frac{b_0 + t_0}{2}$$

- Substitute the above into $\mathbf{M}^{\mathbf{PERSP}}_{\mathbf{ECS}\rightarrow\mathbf{CSS}}$

$$\mathbf{M}^{\mathbf{PERSP}}_{\mathbf{ECS}\rightarrow\mathbf{CSS}} = \begin{bmatrix} \frac{2n_0}{r_0 - l_0} & 0 & 0 & 0 \\ 0 & \frac{2n_0}{t_0 - b_0} & 0 & 0 \\ 0 & 0 & \frac{n_0 + f_0}{f_0 - n_0} & -\frac{2n_0 f_0}{f_0 - n_0} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

which is equivalent to the original $\mathbf{M}^{\mathbf{PERSP}}_{\mathbf{ECS}\rightarrow\mathbf{CSS}}$ with the clipping bounds replaced by the initial clipping bounds.

Extended Viewing Transformation(5)

A. Truncated Pyramid Not Symmetrical about z_e-Axis

- Thus it is not necessary to have initial clipping bounds; can name them *n*, *f*, *l*, *r*, *b*, *t* from the start.
- The ECS→CSS mapping in the case of non-symmetrical perspective projection is thus:

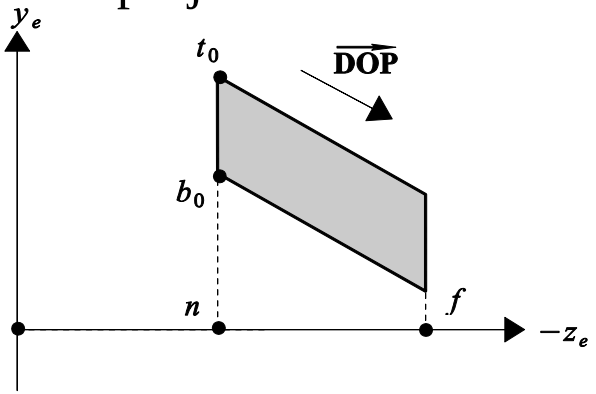
$$\mathbf{M}_{\mathbf{ECS}\rightarrow\mathbf{CSS}}^{\mathbf{PERSP-NON-SYM}} = \mathbf{M}_{\mathbf{ECS}\rightarrow\mathbf{CSS}}^{\mathbf{PERSP}} \cdot \mathbf{SH}_{\mathbf{NON-SYM}} = \begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ 0 & 0 & \frac{n+f}{f-n} & -\frac{2nf}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -\frac{l+r}{2n} & 0 \\ 0 & 1 & -\frac{b+t}{2n} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

Extended Viewing Transformation(6)

B. Oblique Projection

- Useful, e.g., in the computation of oblique views for 3D displays:
  - $\mathbf{M}^{\mathbf{ORTHO}}_{\mathbf{ECS}\rightarrow\mathbf{CSS}}$ is not sufficient
  - Direction of projection must be taken into account
- View volume: a six-sided parallelepiped specified by:
  - The 6 parameters used for the non-symmetrical pyramid (n_0, f_0, l_0, r_0, b_0, t_0)
  - The direction of projection vector $\overrightarrow{\mathbf{DOP}}$



- Steps:
  1. Translate the view volume so that (l_0, b_0, n_0)-point moves to the ECS origin
  2. Perform a shear in the *xy*-plane

Extended Viewing Transformation(7)

B. Oblique Projection (steps)

- Take the point defined by the origin and $\overrightarrow{\mathbf{DOP}} = [DOP_x, DOP_y, DOP_z]^T$
  - Shear the (DOP_y) coordinate to 0: $DOP_y + B \cdot DOP_z = 0 \Rightarrow B = -\dfrac{DOP_y}{DOP_z}$
  - Similarly for the x-shear factor: $A = -\dfrac{DOP_x}{DOP_z}$
- The required transformation is:

$$\mathbf{SH}_{\mathbf{PARALLEL}} \cdot \mathbf{T}_{\mathbf{PARALLEL}} = \begin{bmatrix} 1 & 0 & -\dfrac{DOP_x}{DOP_z} & 0 \\ 0 & 1 & -\dfrac{DOP_y}{DOP_z} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -l_0 \\ 0 & 1 & 0 & -b_0 \\ 0 & 0 & 1 & -n_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Alter the clipping bounds to reflect the new rectangular parallelepiped
  $n = 0, f = f_0 - n_0, l = 0, r = r_0 - l_0, \; b = 0, t = t_0 - b_0$
- ECS→CSS mapping for a general parallel projection is thus:

$$\mathbf{M}^{\mathbf{PARALLEL}}_{\mathbf{ECS}\rightarrow\mathbf{CSS}} = \mathbf{M}^{\mathbf{ORTHO}}_{\mathbf{ECS}\rightarrow\mathbf{CSS}} \cdot \mathbf{SH}_{\mathbf{PARALLEL}} \cdot \mathbf{T}_{\mathbf{PARALLEL}}$$

Frustum Culling & the Viewing Transformation

- **Frustum culling**:
  - Eliminates primitives (parts or whole) that lie outside the view volume(frustum)
  - Is implemented by 3D clipping algorithms
- The viewing transformation defines the 3D clipping boundaries.
- Clipping takes place in CSS, after the application of $\mathbf{M}_{\mathbf{ECS}\rightarrow\mathbf{CSS}}^{\mathbf{PERSP}}$ or $\mathbf{M}_{\mathbf{ECS}\rightarrow\mathbf{CSS}}^{\mathbf{ORTHO}}$ but before the division by w.
- Clipping boundaries for: perspective projection: $-w \leq x, y, z \leq w$

  orthographic or parallel projection: $-1 \leq x, y, z \leq 1$
- Why clip in 3D and not in 2D, after throwing away *z*?
  - In perspective projection, objects behind the center of projection ***E*** would appear upside-down (no sufficient information in clipping)
  - Avoids possible perspective division by 0
  - The near and far clipping planes limit the depth range & enable the optimal allocation of the bits of the depth buffer

The Viewport Transformation

- **Viewport**: rectangular part of the screen where the contents of the view volume are displayed.
- Defined by its bottom-left and top-right corners
  - $[x_{min}, y_{min}]^T$ and $[x_{max}, y_{max}]^T$ in pixel coordinates
  - $[x_{min}, y_{min}, z_{min}]^T$ and $[x_{max}, y_{max}, z_{max}]^T$ to maintain the z-coordinate
- **Viewport transformation** converts objects from CSS into the viewport coordinate system (VCS) with a scaling & a translation:

$$\mathbf{M}_{\text{CSS}\rightarrow\text{VCS}}^{\text{VIEWPORT}} = \begin{bmatrix} 1 & 0 & 0 & \frac{x_{min}+x_{max}}{2} \\ 0 & 1 & 0 & \frac{y_{min}+y_{max}}{2} \\ 0 & 0 & 1 & \frac{z_{min}+z_{max}}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{x_{max}-x_{min}}{2} & 0 & 0 & 0 \\ 0 & \frac{y_{max}-y_{min}}{2} & 0 & 0 \\ 0 & 0 & \frac{z_{max}-z_{min}}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{x_{max}-x_{min}}{2} & 0 & 0 & \frac{x_{min}+x_{max}}{2} \\ 0 & \frac{y_{max}-y_{min}}{2} & 0 & \frac{y_{min}+y_{max}}{2} \\ 0 & 0 & \frac{z_{max}-z_{min}}{2} & \frac{z_{min}+z_{max}}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- This is a generalization of the 2D window-to-viewport transformation [Ex. 3.8].
- The size of the viewport defines the final size of the objects on screen