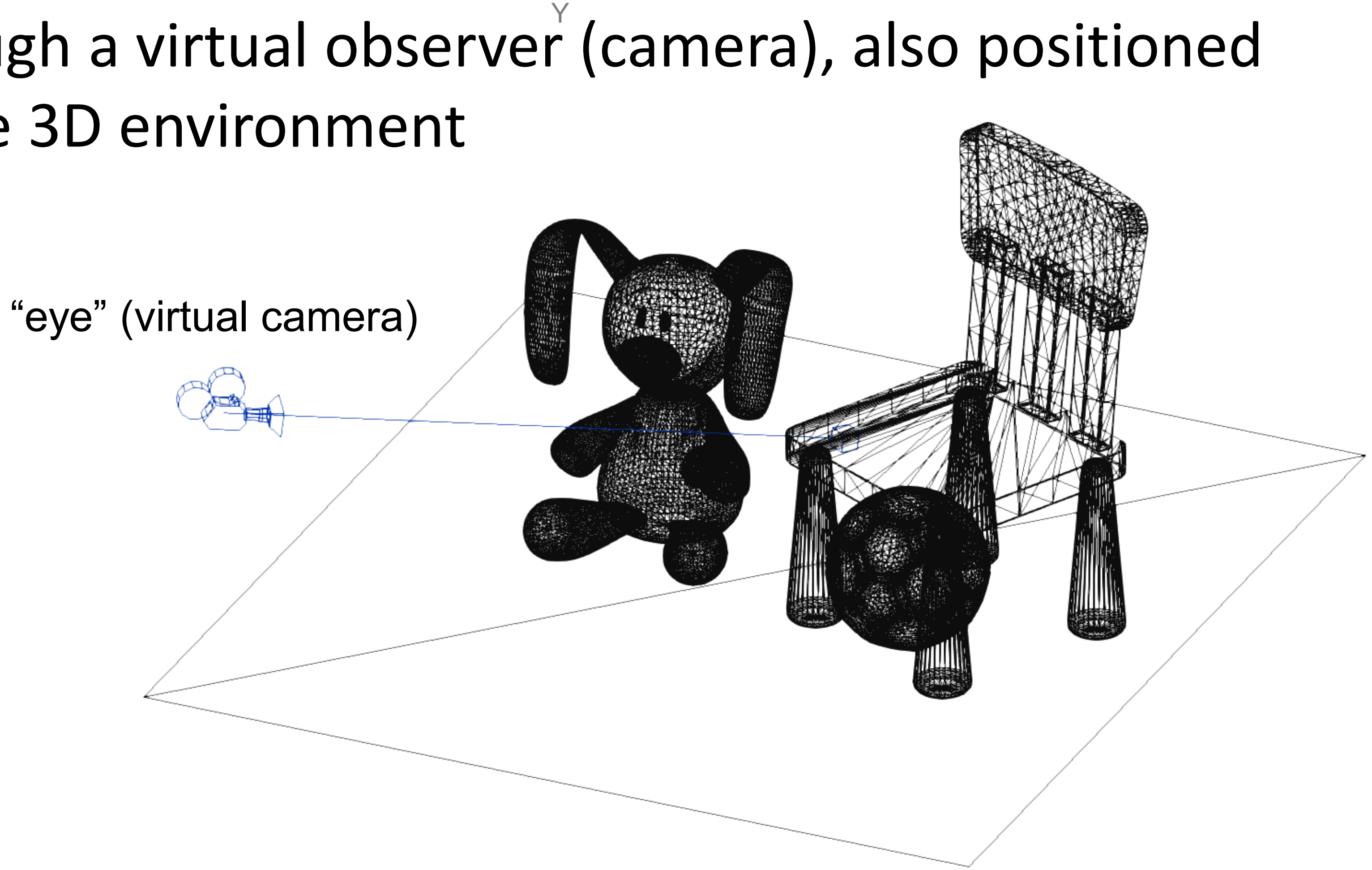
Viewing and Projections

Georgios Papaioannou - 2014

VIEWING TRANSFORMATION

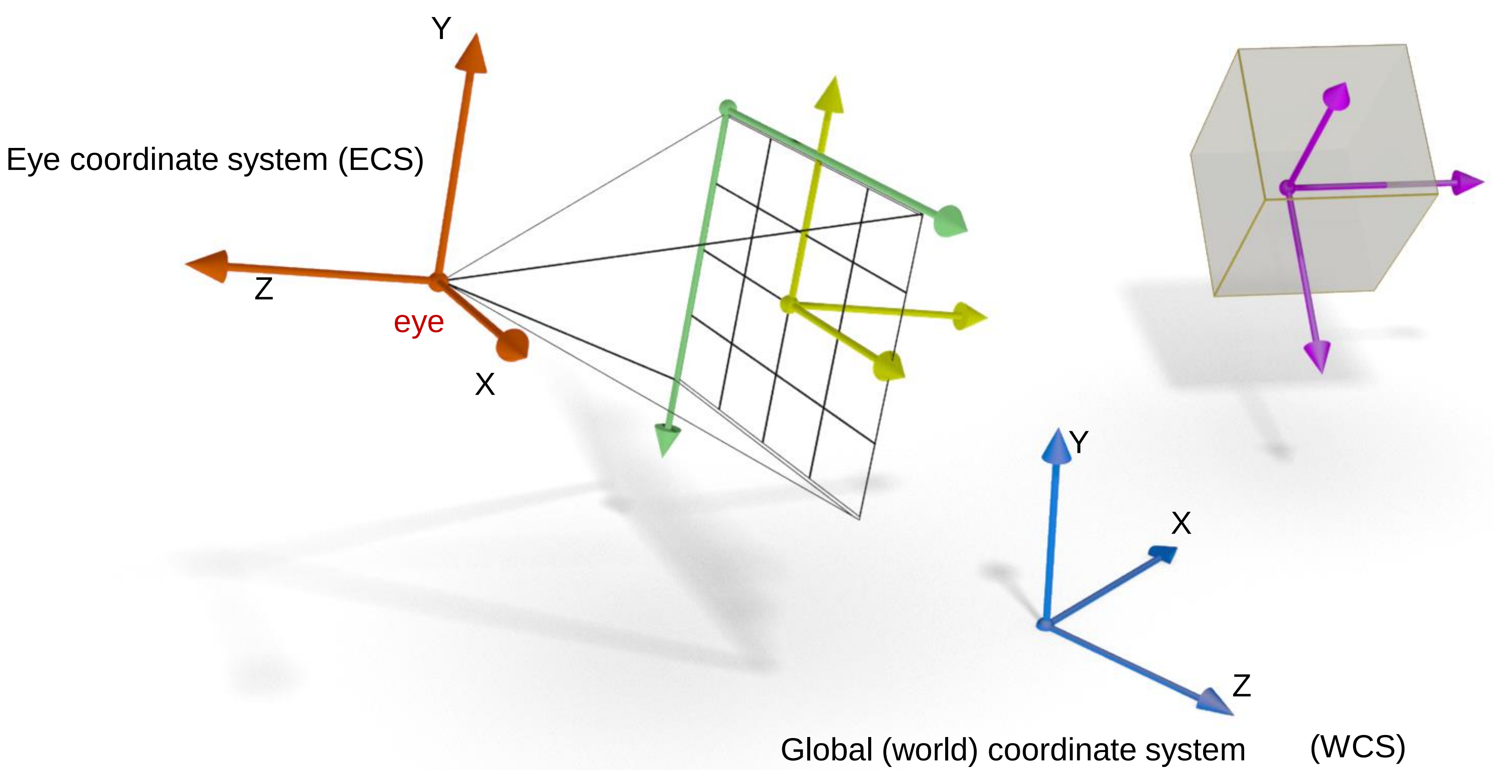
The Virtual Camera

- All graphics pipelines perceive the virtual world through a virtual observer (camera), also positioned in the 3D environment



Eye Coordinate System (1)

- The virtual camera or “eye” also has its own coordinate system, the eye coordinate system

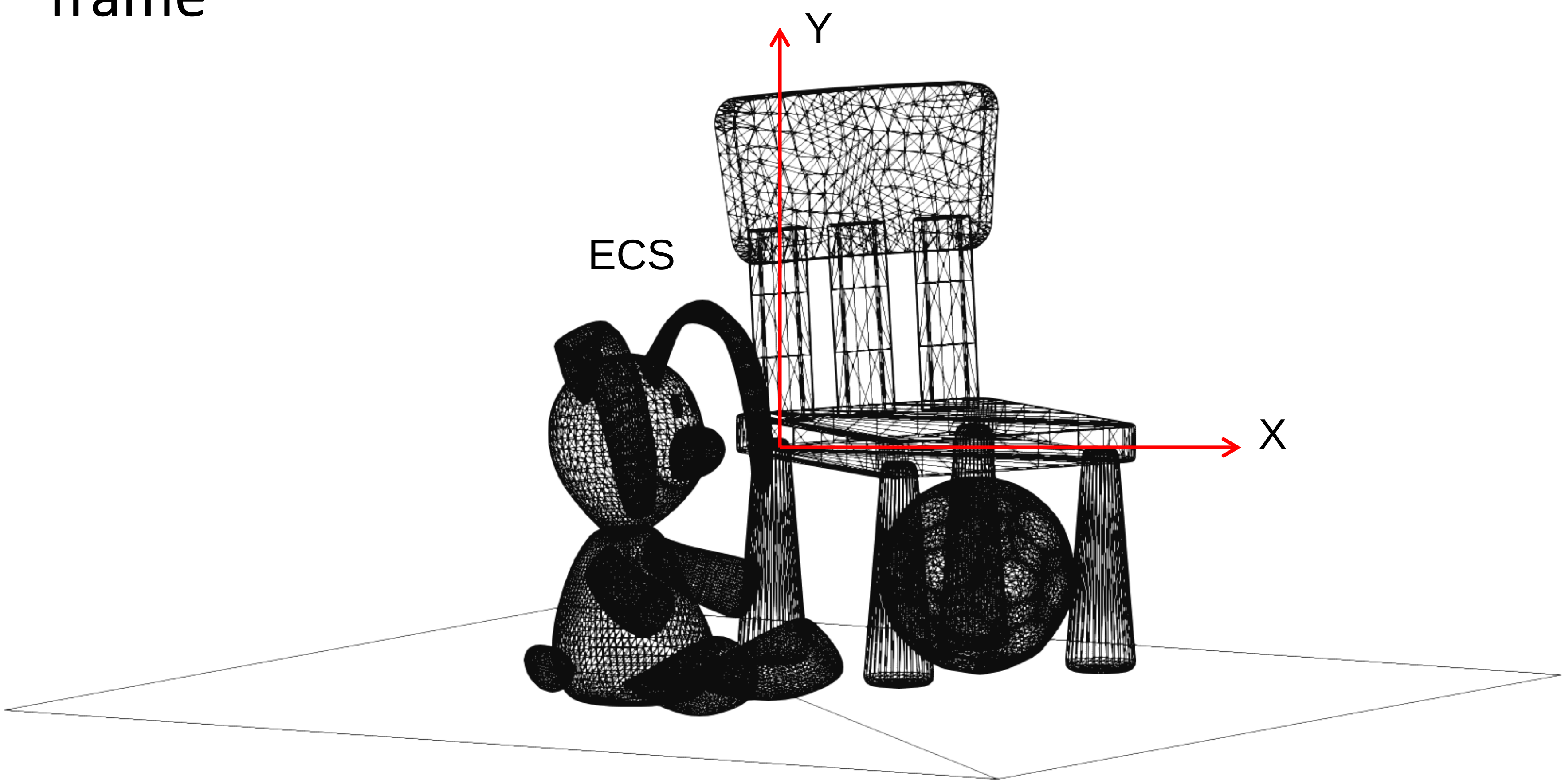


Eye Coordinate System (2)

- Expressing the scene's geometry in the ECS is a natural "egocentric" representation of the world:
  - It is how we perceive the user's relationship with the environment
  - It is usually a more convenient space to perform certain rendering tasks, since it is related to the ordering of the geometry in the final image

Eye Coordinate System (3)

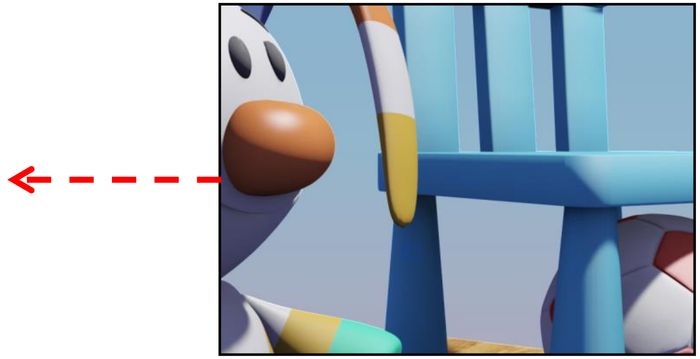
- Coordinates as “seen” from the camera reference frame



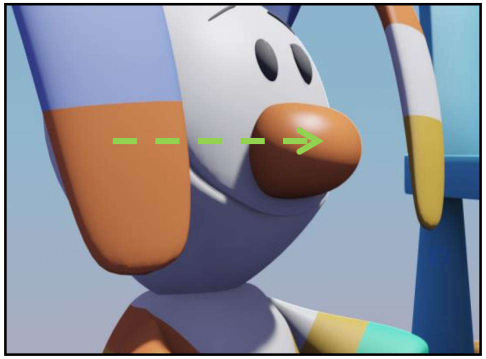
Eye Coordinate System (4)

- What "egocentric" means in the context of transformations?
  - Whatever transformation produced the camera system → its inverse transformation expresses the world w.r.t. the camera
- Example: If I move the camera "left", objects appear to move "right" in the camera frame:

WCS camera motion



Eye-space object motion



Moving to Eye Coordinates

- Moving to ECS is a change of coordinates transformation
- The WCS→ECS transformation expresses the 3D environment in the camera coordinate system
- We can define the ECS transformation in two ways:
  - A) Invert the transformations we applied to place the camera in a particular pose
  - B) Explicitly define the coordinate system by placing the camera at a specific location and setting up the camera vectors

WCS→ECS: Version A (1)

- Let us assume that we have an initial camera at the origin of the WCS
- Then, we can move and rotate the "eye" to any pose (rigid transformations only: No sense in scaling a camera):

$$\{\mathbf{o}_c, \vec{\mathbf{u}}, \vec{\mathbf{v}}, \vec{\mathbf{w}}\} = \overbrace{\mathbf{R}_1\mathbf{R}_2\mathbf{T}_1\mathbf{R}_2 \ldots . \mathbf{T}_n\mathbf{R}_m}^{\mathbf{M}_c}\{\mathbf{o}, \hat{\mathbf{e}}_1, \hat{\mathbf{e}}_2, \hat{\mathbf{e}}_3\}$$

- The eye space coordinates of shapes, given their WCS coordinates can be simply obtained by:

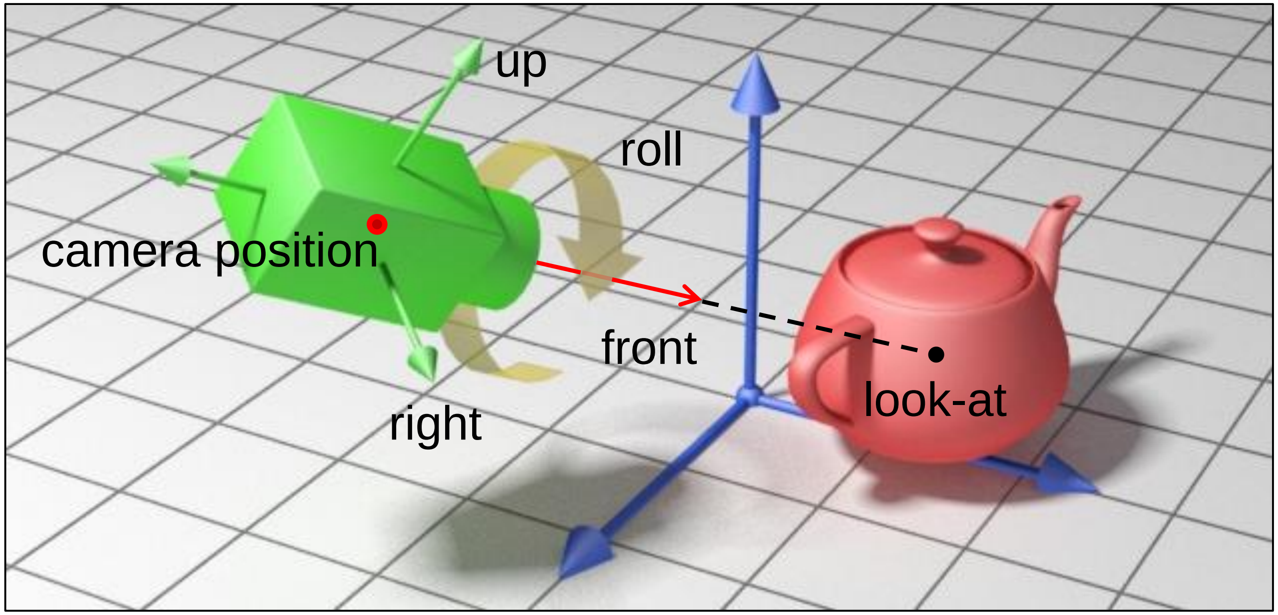
$$\mathbf{v}_{ECS} = \mathbf{M}_c^{-1}\mathbf{v}_{WCS}$$

WCS→ECS: Version A (2)

- This version of the WCS→ECS transformation computation is useful in cases where:
  - The camera system is dependent on (attached to) some moving geometry (e.g. a driver inside a car)
  - The camera motion is well-defined by a simple trajectory (e.g. an orbit around an object being inspected)

WCS→ECS: Version B ("Look At") (1)

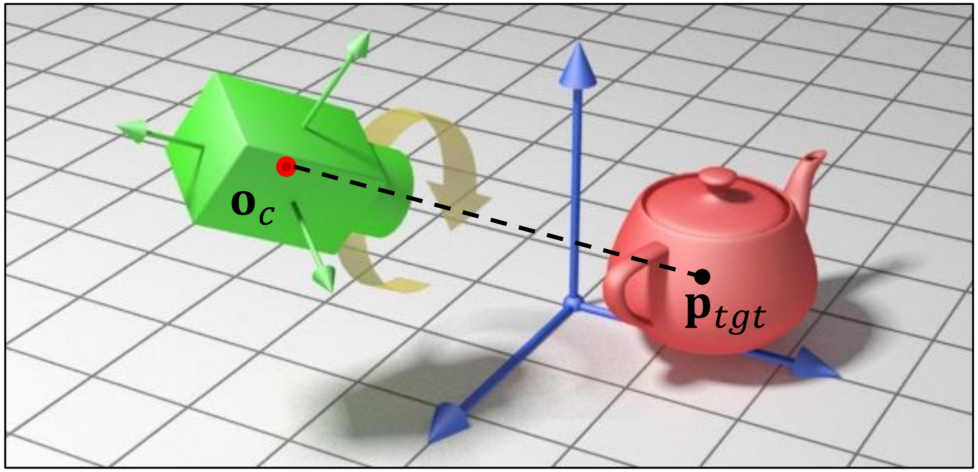
- Let us directly define a camera system by specifying where the camera is, where does it point to and what is its roll (or usually, its "up" or "right" vector)



WCS→ECS: Version B ("Look At") (2)

- The camera coordinate system offset is the eye (camera) position $\mathbf{o}_c$
- Given the look-at position (the camera target) $\mathbf{p}_{tgt}$ and $\mathbf{o}_c$, we can determine the "front" direction:

$$\vec{\mathbf{d}}_{front} = \mathbf{p}_{tgt} - \mathbf{o}_c \text{ (normalized)}$$



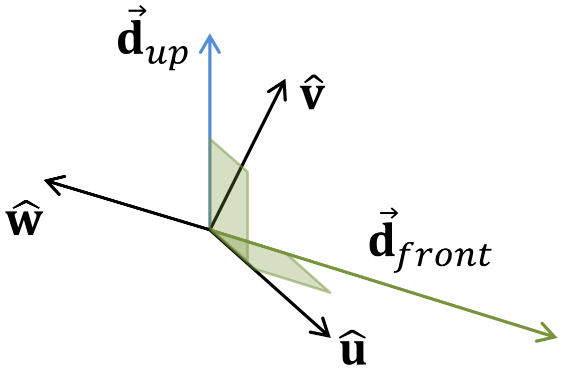
WCS→ECS: Version B ("Look At") (3)

- The "up" or "right" vector need not be given precisely, as we can infer the coordinate system indirectly
- Let us provide an "upright" up vector: $\vec{\mathbf{d}}_{up}$ =(0,1,0)
- Provided that $\vec{\mathbf{d}}_{up}$ is not parallel to $\vec{\mathbf{d}}_{front}$:

$$\hat{\mathbf{w}} = -\vec{\mathbf{d}}_{front}/\left\|\vec{\mathbf{d}}_{front}\right\|$$

$$\vec{\mathbf{u}} = \vec{\mathbf{d}}_{front} \times \vec{\mathbf{d}}_{up}, \qquad \hat{\mathbf{u}} = \vec{\mathbf{u}}/\|\vec{\mathbf{u}}\|$$

$$\hat{\mathbf{v}} = \hat{\mathbf{w}} \times \hat{\mathbf{u}}$$



WCS→ECS: Version B ("Look At") (4)

- We can use the derived local camera coordinate system to define the change of coordinates transformation (see 3D Transformations):

$$\mathbf{p}_{ECS} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \mathbf{T}_{-\mathbf{o}_c} \cdot \mathbf{p}_{WCS}$$

WCS→ECS: Version B ("Look At") (5)

- This version of the WCS→ECS transformation computation is useful in cases where:
  - There is a free roaming camera
  - The camera follows (observes) a certain target in space
  - The position (and target) are explicitly defined

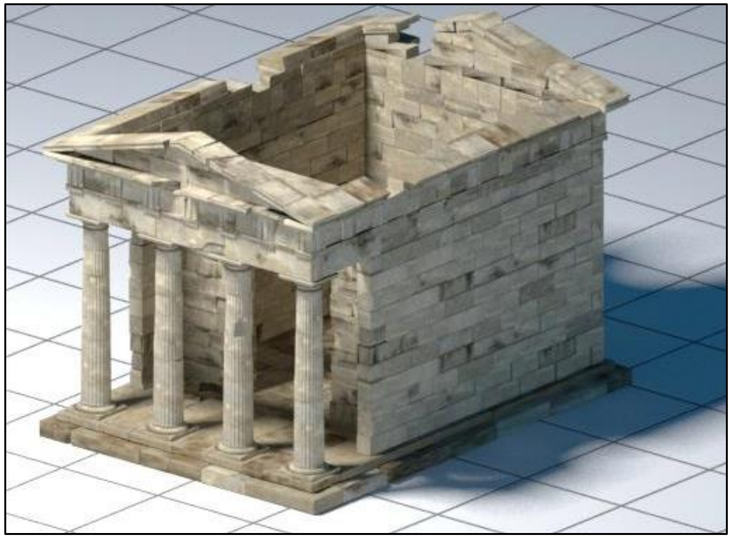
PROJECTIONS

Projection

- Is the process of transforming 3D coordinates of shapes to points on the viewing plane
- Viewing plane is the 2D flat surface that represents an embedding of an image into the 3D space
  - We can define viewing systems where the “projection” surface is not planar (e.g. fish-eye lenses etc.)
- (Planar) projections are define by a projection (viewing) plane and a center of projection (eye)

Taxonomy

- Two main categories:
  - Parallel projections: infinite distance between CoP and viewing plane



  - Perspective projections: Finite distance between CoP and viewing plane

Where do We Perform the Projections?

- Since in projections we “collapse” a 3D shape onto a 2D surface, we essentially want to loose one coordinate (say the depth z)
- Therefore, it is convenient to perform the projection when shapes are expressed in the ECS

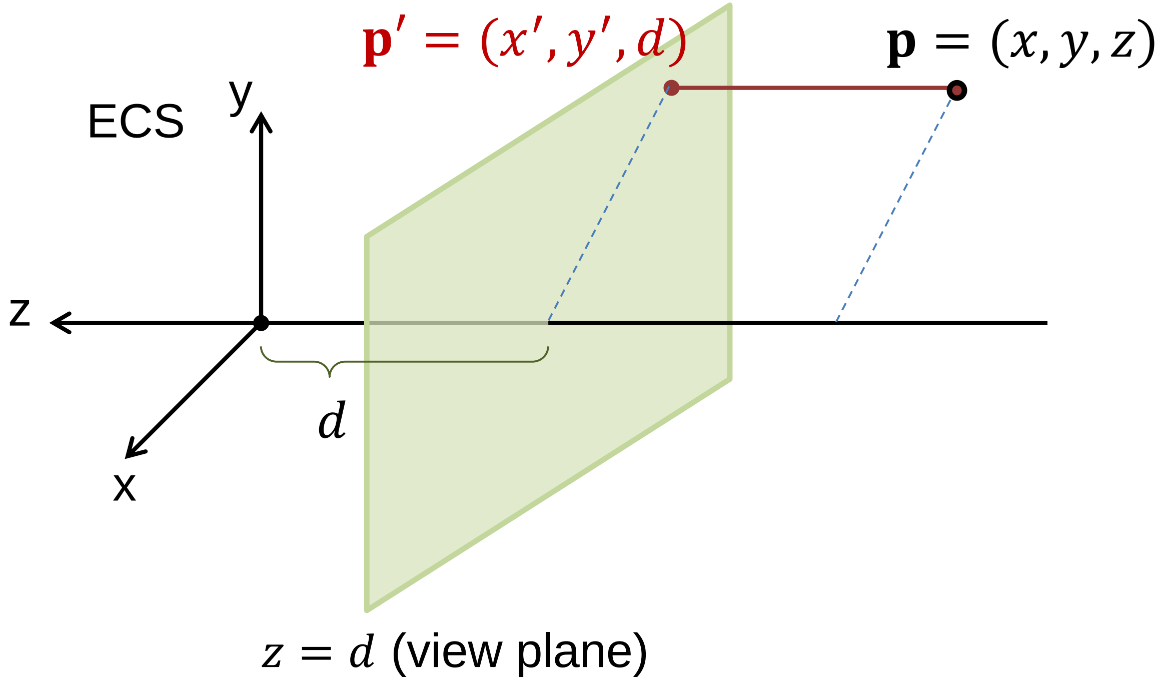
Orthographic Projection (1)

- The simplest projection:
- Collapse the coordinates on plane parallel to xy at z=d (usually 0)

$$y' = y$$

$$x' = x$$

$$z' = d$$



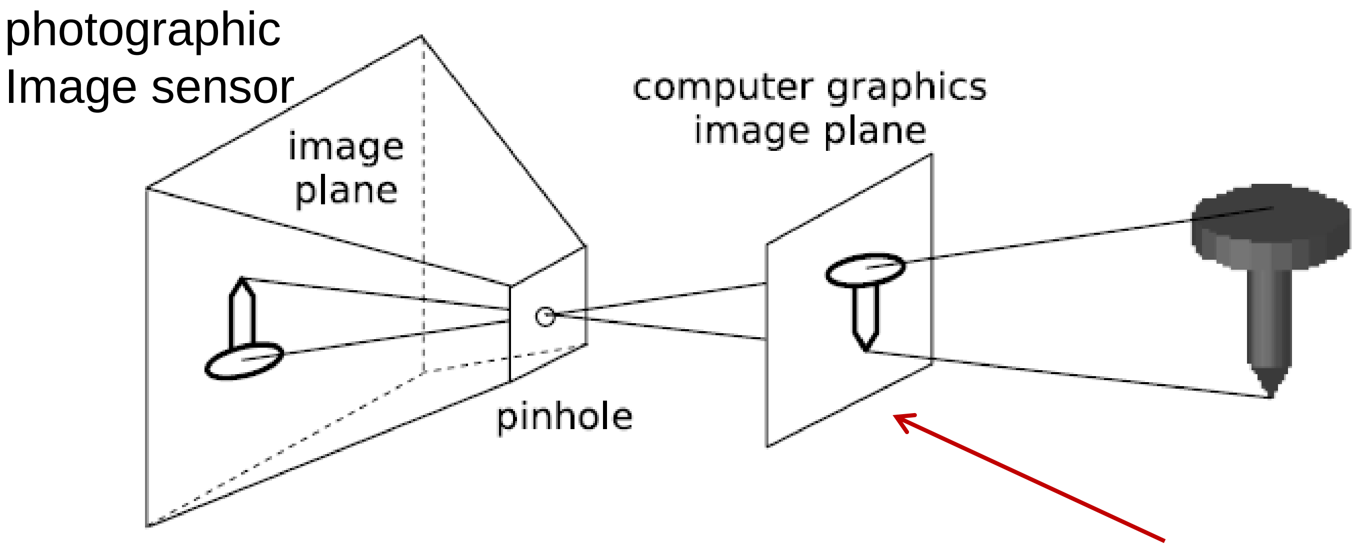
Orthographic Projection (2)

- Very simple matrix representation
- Note that the rank of the matrix is less than its dimension: This not a reversible transformation!
  - This is also intuitively justified since we “loose” all information about depth

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

The Pinhole Camera Model

- It is an ideal camera (i.e. cannot exist in practice)
- It is the simplest modeling of a camera:



For simplicity, graphics use a "front" symmetrical projection plane

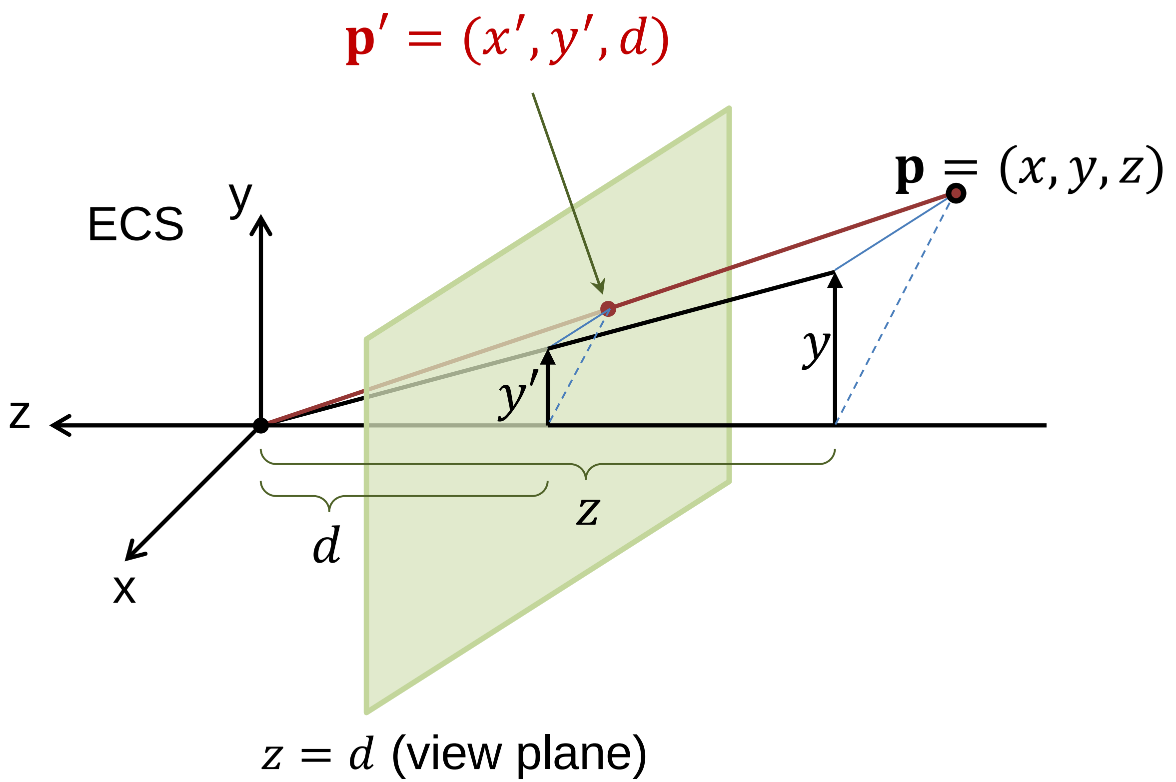
The Perspective Projection

- From similar triangles, we have:

$$y' = \frac{d \cdot y}{z}$$

$$x' = \frac{d \cdot x}{z}$$

$$z' = d$$



Matrix Form of Perspective Projection

- The perspective projection is not a linear operation (division by z) →
- It cannot be completely represented by a linear operator such as a matrix multiplication

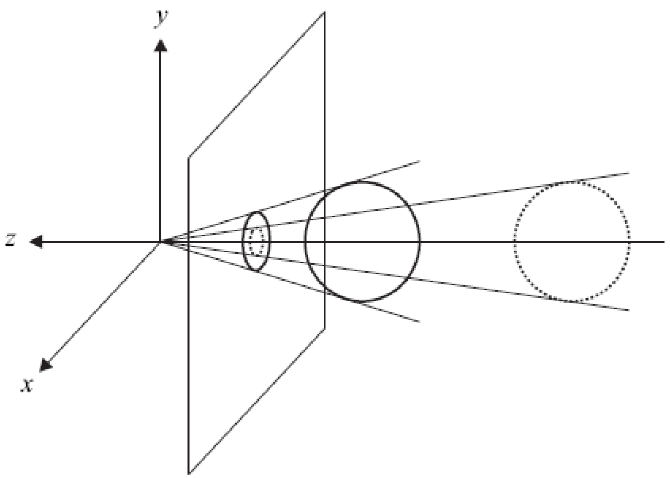
$$\mathbf{P}_{PER} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Requires a division by the w coordinate to rectify the homogeneous coordinates

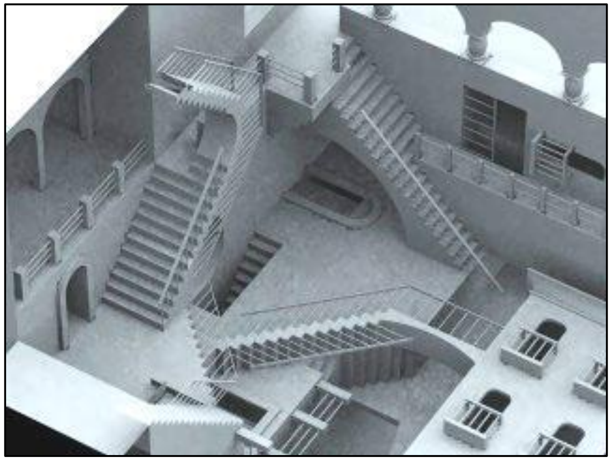
$$\mathbf{P}_{PER} \cdot \mathbf{p}_{WCS} = \begin{bmatrix} x \cdot d \\ y \cdot d \\ z \cdot d \\ z \end{bmatrix} \rightarrow \begin{bmatrix} x \cdot d \\ y \cdot d \\ z \cdot d \\ z \end{bmatrix} / z = \begin{bmatrix} x \cdot d/z \\ y \cdot d/z \\ d \\ 1 \end{bmatrix}$$

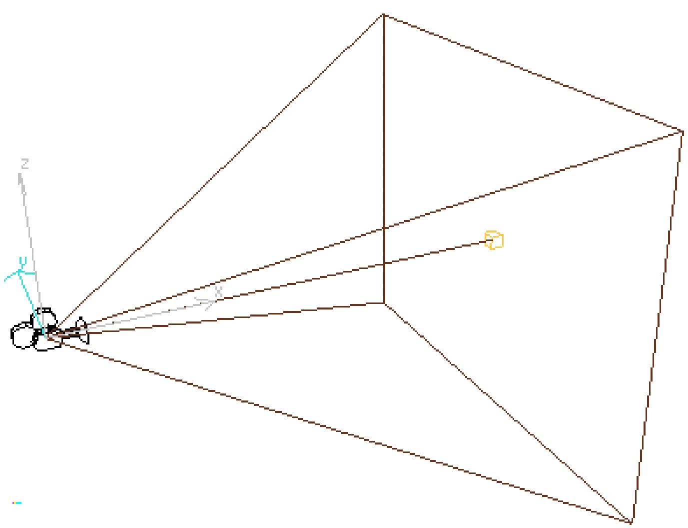
Properties of the Perspective Projection

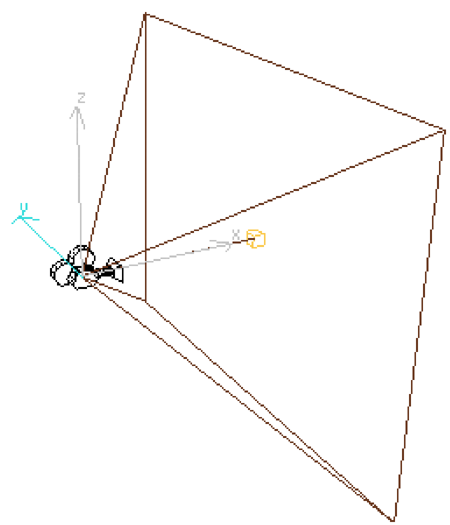
- Lines are projected to lines
- Distances are not preserved
- Angles between lines are not preserved unless lines are parallel to the view plane
- Perspective foreshortening: The size of the projected shape is inversely proportional to the distance to the plane



The Impact of Focal Distance *d*

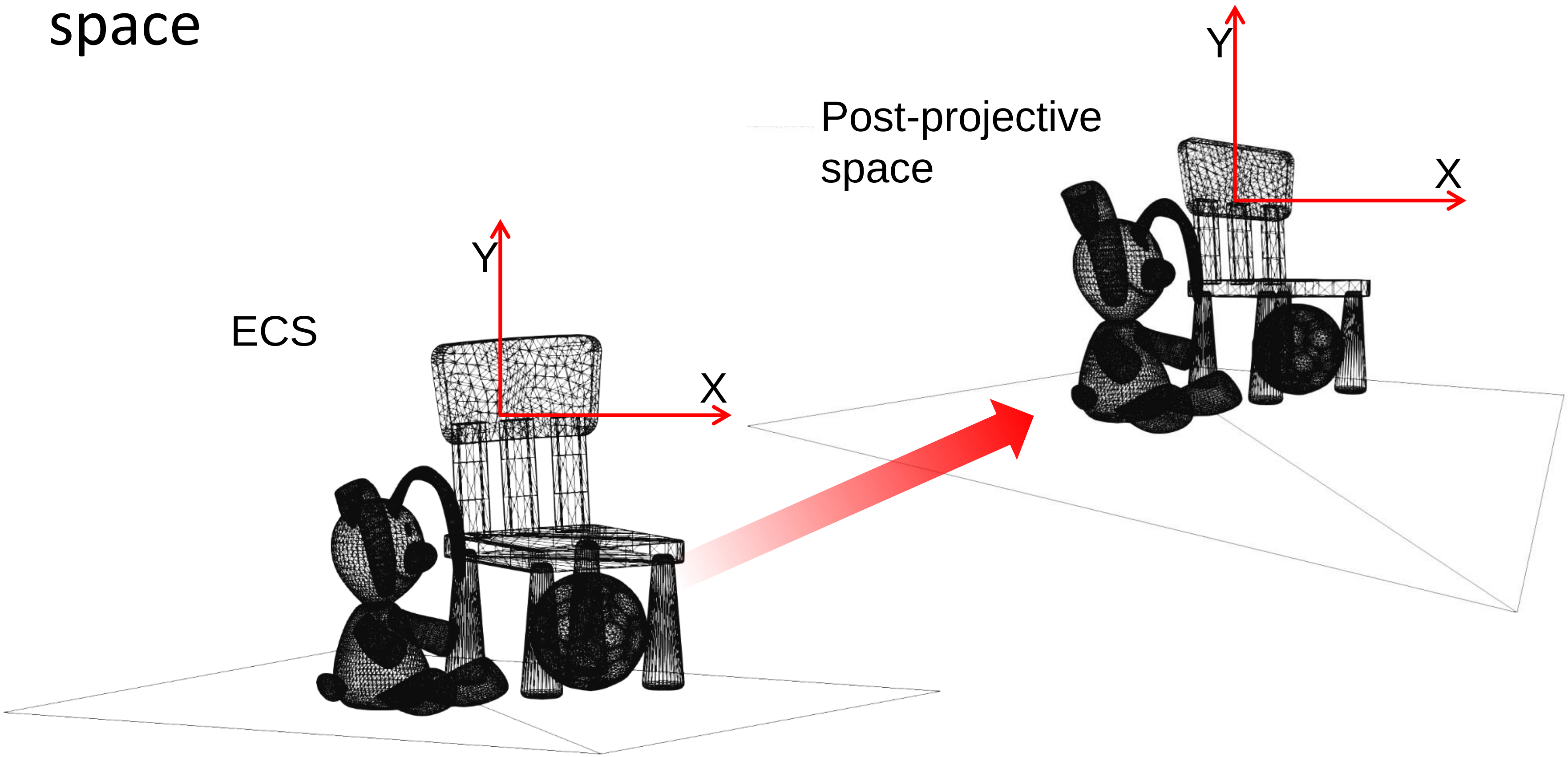






What Happens After Projection? (1)

- Coordinates are transformed to a “post-projective” space



What Happens After Projection? (2)

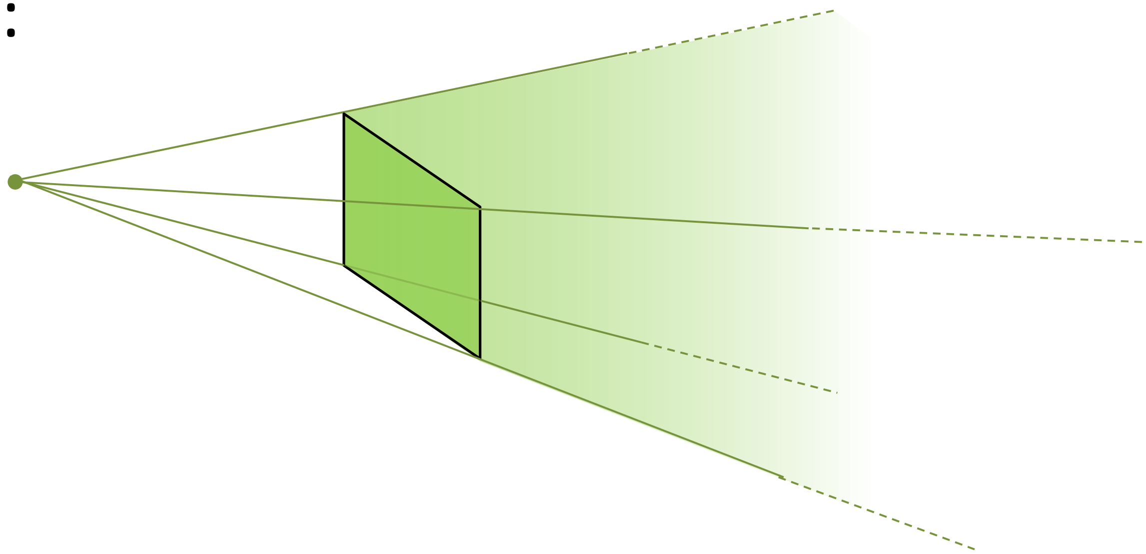
- Remember also that "depth" is for now collapsed to the focal distance
- How then are we going to use the projected coordinates to perform "depth" sorting in order to remove hidden surfaces?
- Also, how do we define the extents of what we can see?

Preserving the Depth

- Regardless of what the projection is, we also retain the transformed z values
- For numerical stability, representation accuracy and plausibility of displayed image, we limit the z-range
- $n \leq z \leq f$,
  - n=near clipping value,
  - f=far clipping value,

The View Frustum

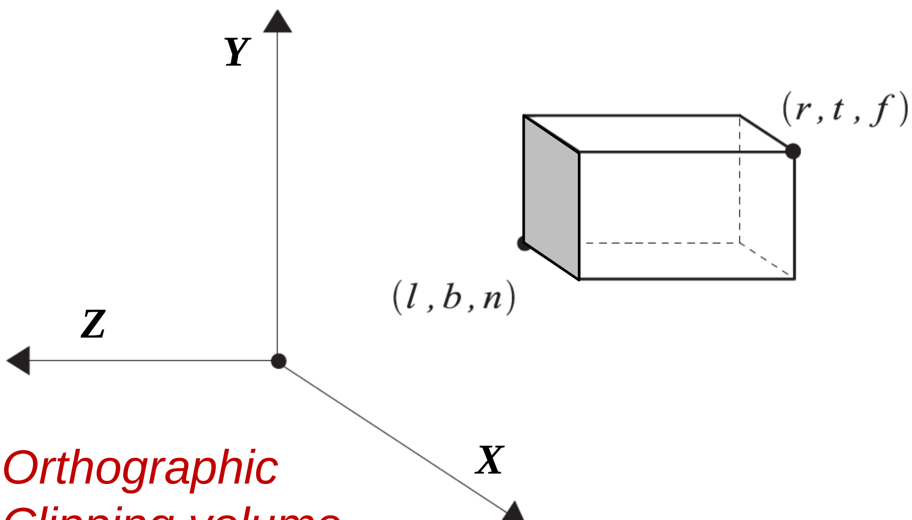
- The boundaries (line segments) of the image, form planes in space:



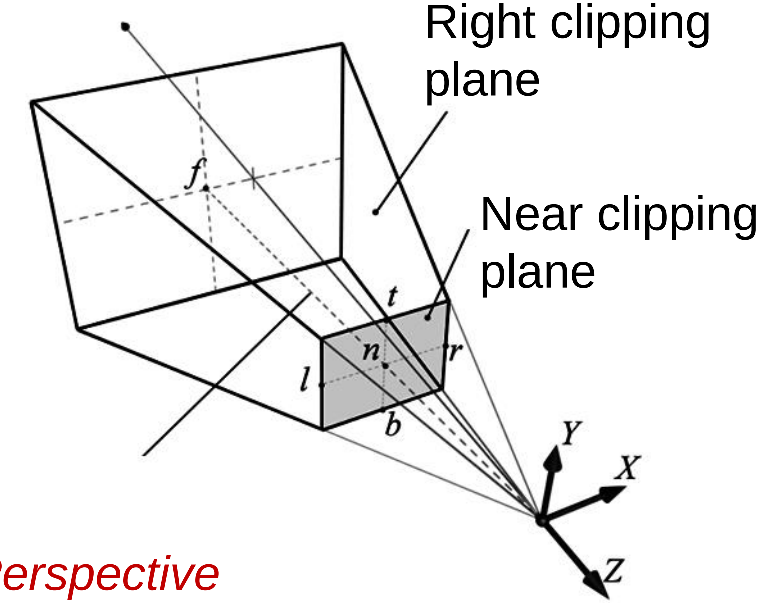
- The intersection of the visible subspaces, defines what we can see inside a view frustum

The Clipping Volume (1)

- The viewing frustum, forms a clipping volume
- It defines which parts of the 3D world are discarded, i.e. do not contribute to the final rendering of the image
- For many rendering architectures, this is a closed volume (capped by the far plane)



Orthographic Clipping volume



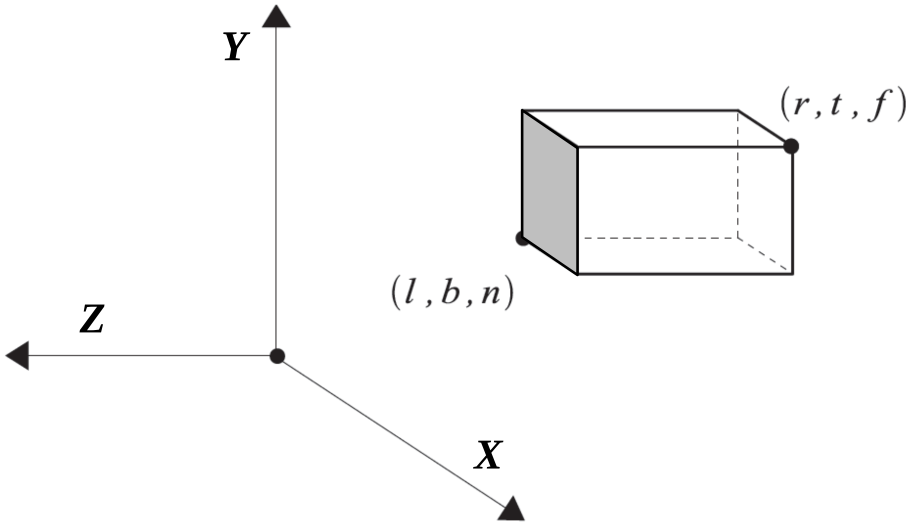
Perspective Clipping volume

The Clipping Volume (2)

- After projection, the contents of the clipping volume are warped to match a rectangular paralepiped
- This post-projective volume is usually considered normalized and its local coordinate system is called Canonical Screen Space (CSS)
- The respective device coordinates are also called Normalized Device Coordinates (NDC)

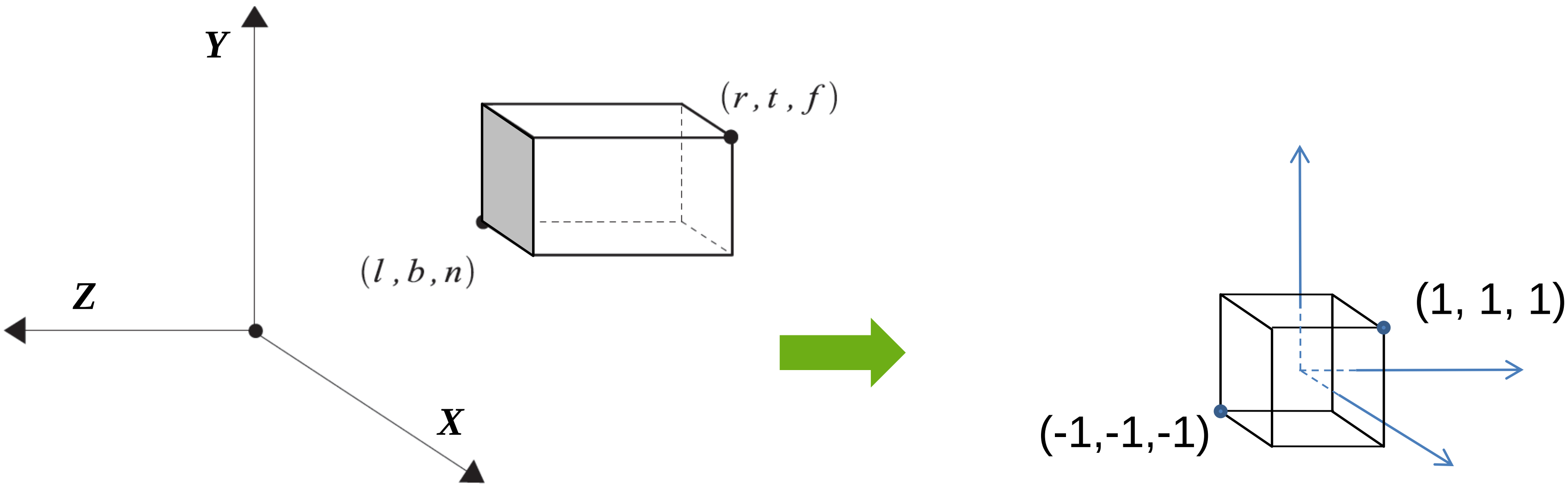
Orthographic Projection Revisited (1)

- Let us now create an orthographic projection that transforms a specific clipping box volume (left, right, bottom, top, near, far) to CSS:



- $x_e = l$, 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.)

Orthographic Projection Revisited (2)



Notice the change of handedness here:
(-1 corresponds to “near”, while “far” is 1)

- A simple translation → scaling transformation can warp the clipping volume into NDC

Orthographic Projection Revisited (3)

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

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

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

Perspective Projection Revisited (1)

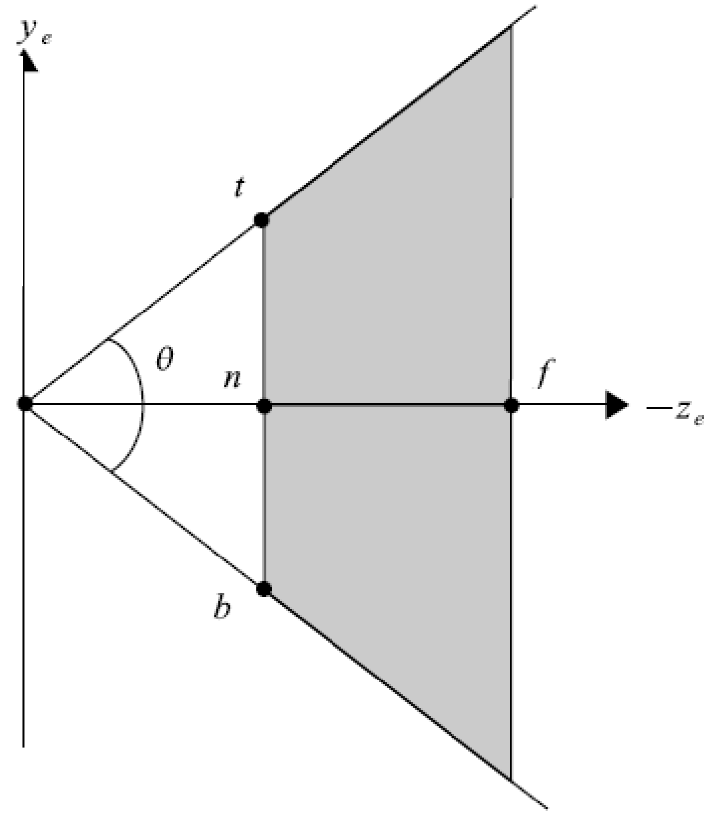
- We want a similar transformation to warp the contents of the perspective frustum into a normalized cube space (CSS)
- Let us now see what happens to geometry when the Cartesian coordinates are perspectively projected (warped) after the transformation:

Perspective Projection Revisited (2)

- In perspective projection, the clipping space is a capped pyramid (frustum)

- $z_e = n$, the near clipping plane;
- $z_e = f$, the far clipping plane ($f < n$).

$$t = |n| \cdot \tan\left(\frac{\theta}{2}\right),$$
$$b = -t,$$
$$r = t \cdot \text{aspect},$$
$$l = -r.$$



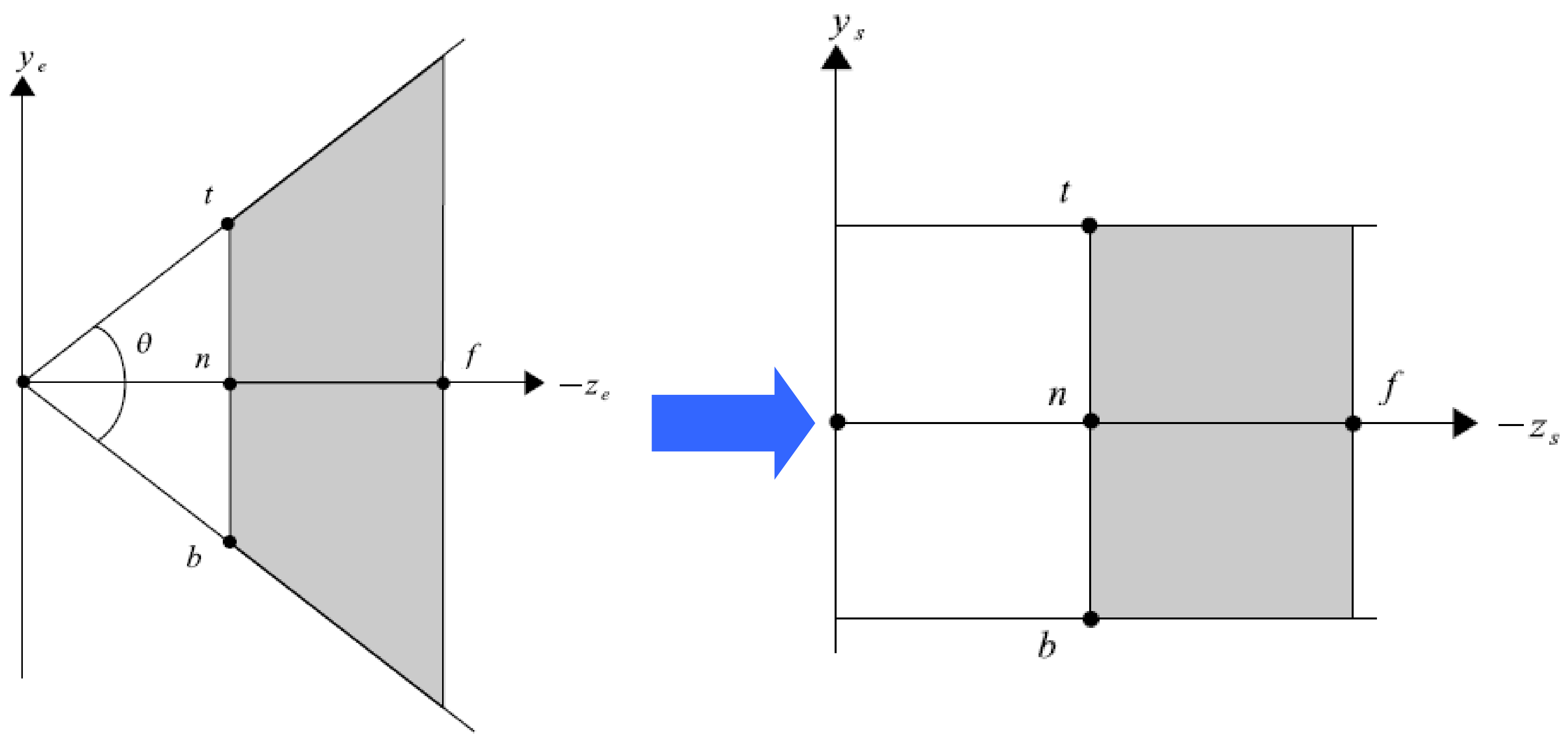
Perspective Projection Revisited (3)

- We still need to perform the perspective division
- We also need to retain the depth information
- Depth must obey the same transformation (division by z) → retain straight lines
- So it must be of the general form: $z_s=A+B/z_e$
- Solving A and B for the boundary conditions:
  $f=A+B/f$ and $n=A+B/n$:
- $A=n+f$
- $B=-nf$ →
- $z_s=n+f-nf/z_e$

Perspective Projection Revisited (4)

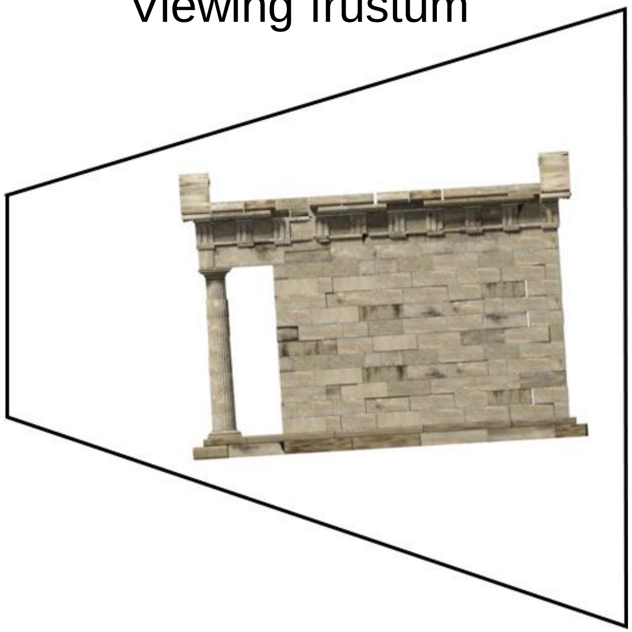
- $z_s = n + f - nf / z_e$

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

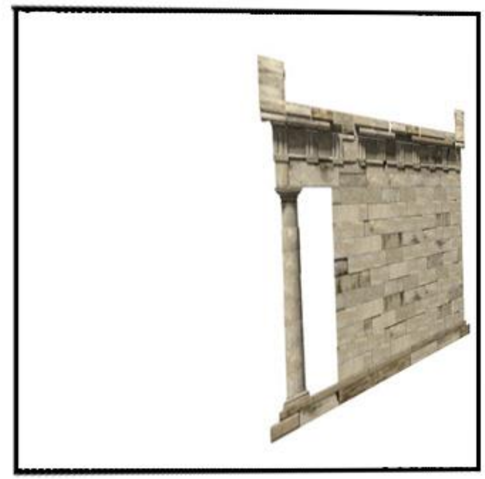


Perspective Projection Revisited (5)

Viewing frustum



Post-projective (NDC) space



Perspective Projection Revisited (6)

- Next, we must normalize the result to bring it to the CSS coordinates:

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

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

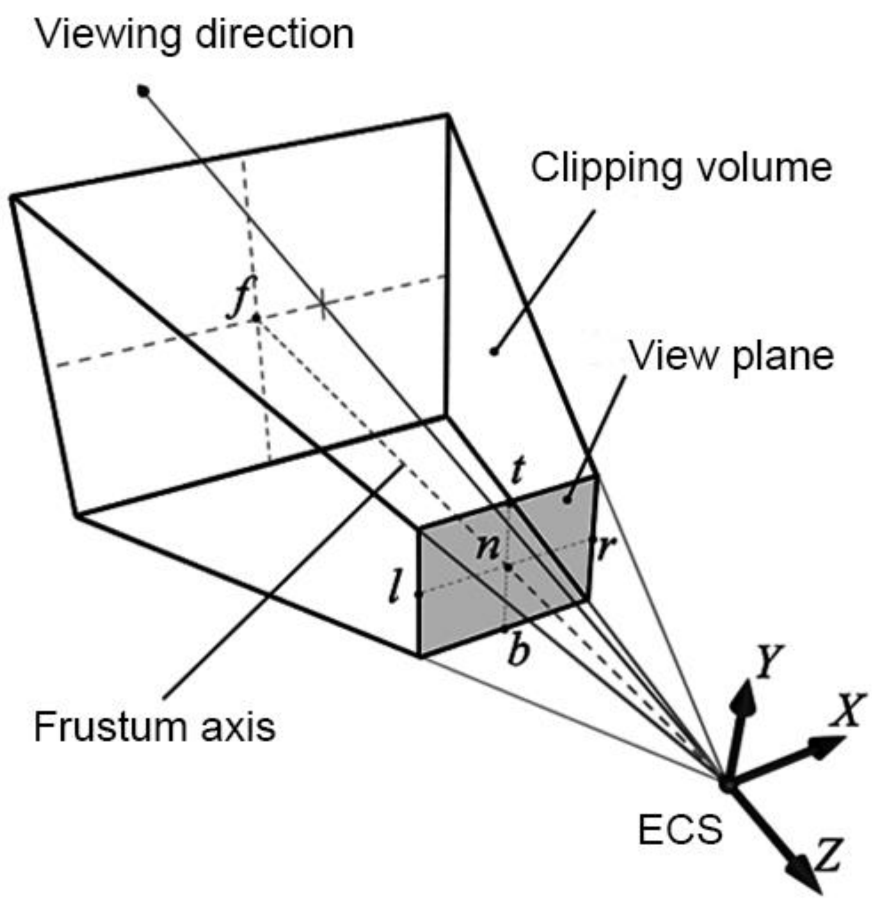
$$= \begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ 0 & 0 & \frac{n+f}{f-n} & -\frac{2nf}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Perspective Projection Revisited (7)

- Of course, we still need to divide with the w coordinate after the matrix multiplication

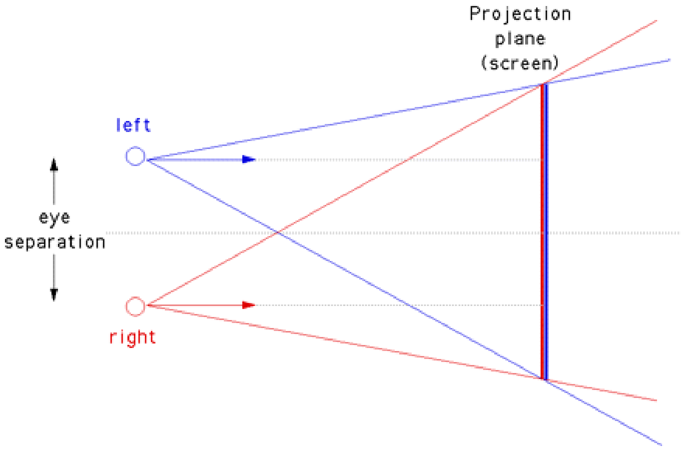
Extended Perspective Projection (1)

- In general, the frustum axis is not aligned with the viewing direction
- To bring this frustum to the CSS normalized volume, we must first skew it

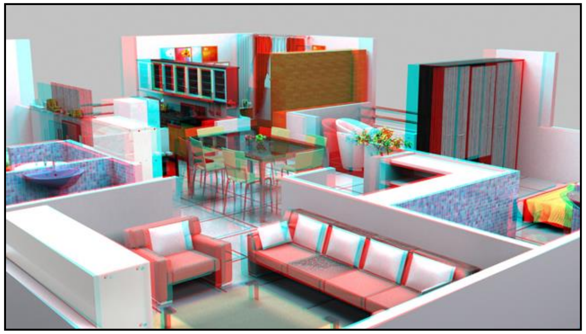


Extended Perspective Projection (2)

- Why do we need an off-axis projection?

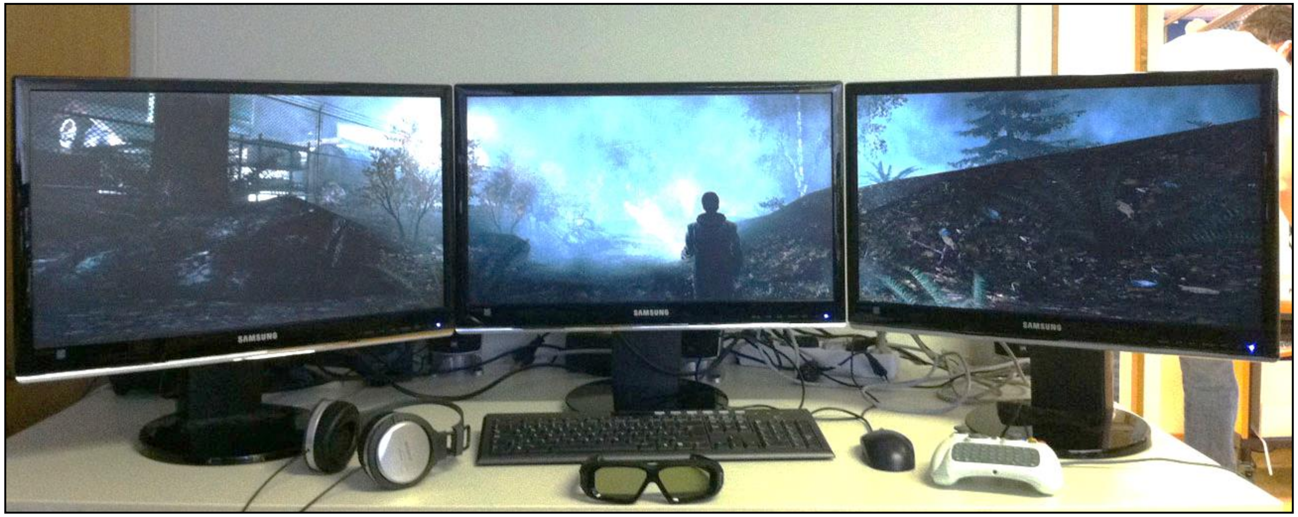


Stereo



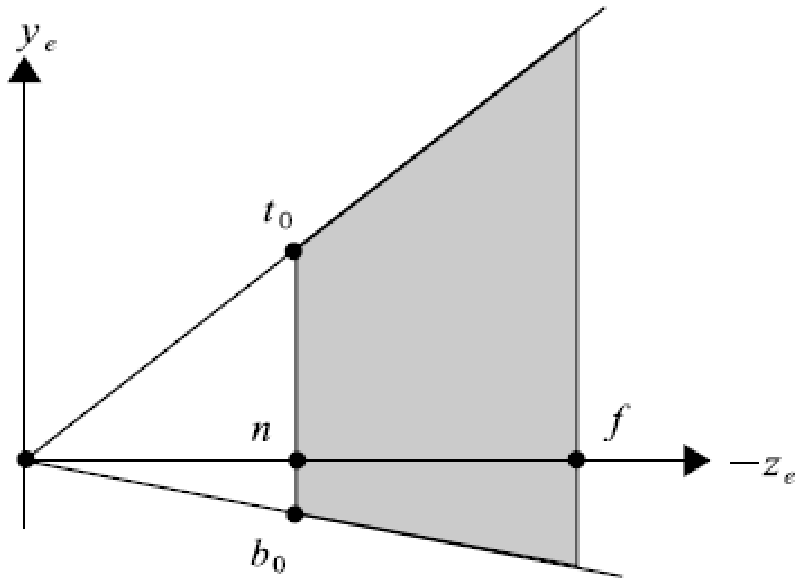
Planar reflections

Multi-view rendering



Extended Perspective Projection (3)

- The center of the near and far cap must coincide with the z axis
- Therefore, using the z-based shear transformation:



$$\mathbf{SH_{xy}} = \begin{bmatrix} 1 & 0 & A & 0 \\ 0 & 1 & B & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- We require: $\frac{l_0 + ro}{2} + An_o = 0 \qquad \frac{b_0 + to}{2} + Bn_o = 0$

Perspective: Putting Everything Together (1)

- The final extended perspective transformation matrix:

$$\mathbf{M}_{\mathbf{ECS}\rightarrow\mathbf{CSS}}^{\mathbf{PERSP-NON-SYM}} = \mathbf{M}_{\mathbf{ECS}\rightarrow\mathbf{CSS}}^{\mathbf{PERSP}} \cdot \mathbf{SH}_{\mathbf{NON-SYM}}$$

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

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

Contributors

- Georgios Papaioannou

- Sources:
  - T. Theoharis, G. Papaioannou, N. Platis, N. M. Patrikalakis, Graphics & Visualization: Principles and Algorithms, CRC Press