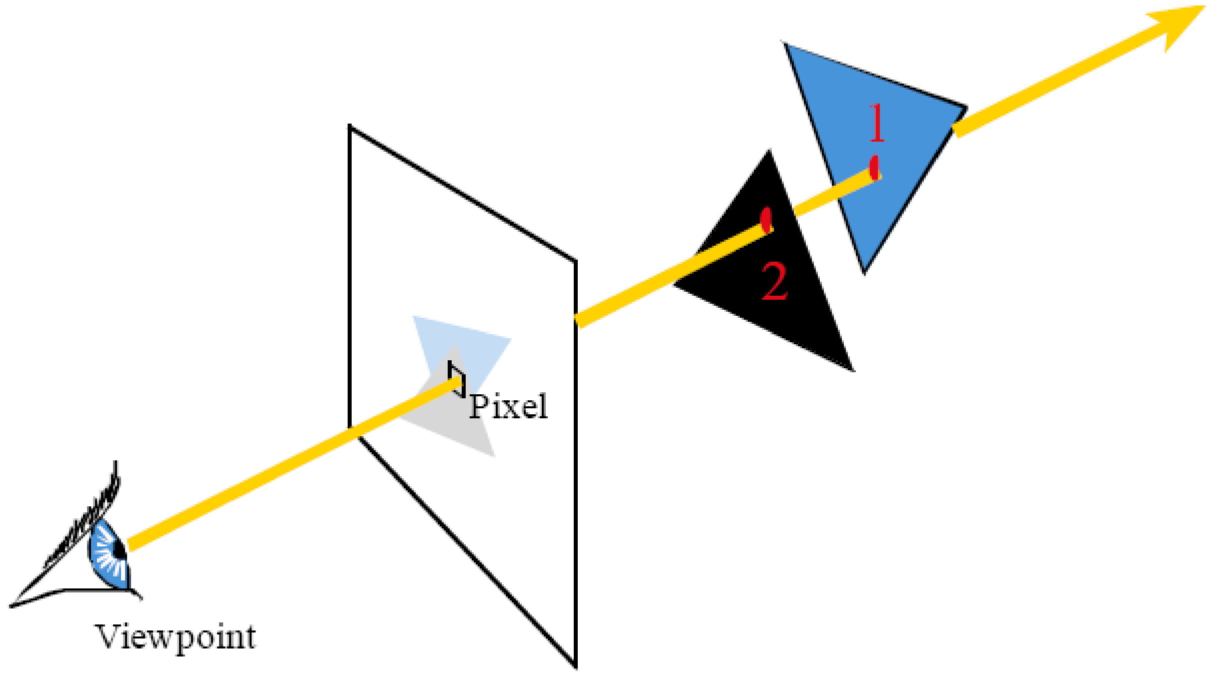
3D Viewing, Orthographic and Perspective Projection

Image order and Object order

- Image order: “backward” approach
  - Start from pixel and ask what part of scene projects to pixel
  - Explicitly construct the ray corresponding to the pixel
- Object order: “forward” approach
  - Start from a point in 3D and compute its projection into the image
- Matrix transformations critical for object order approach (important for both)
  - Combines seamlessly with coordinate transformations used to position camera and model
  - Ultimate goal: single matrix operation to map any 3D point to its correct screen location.

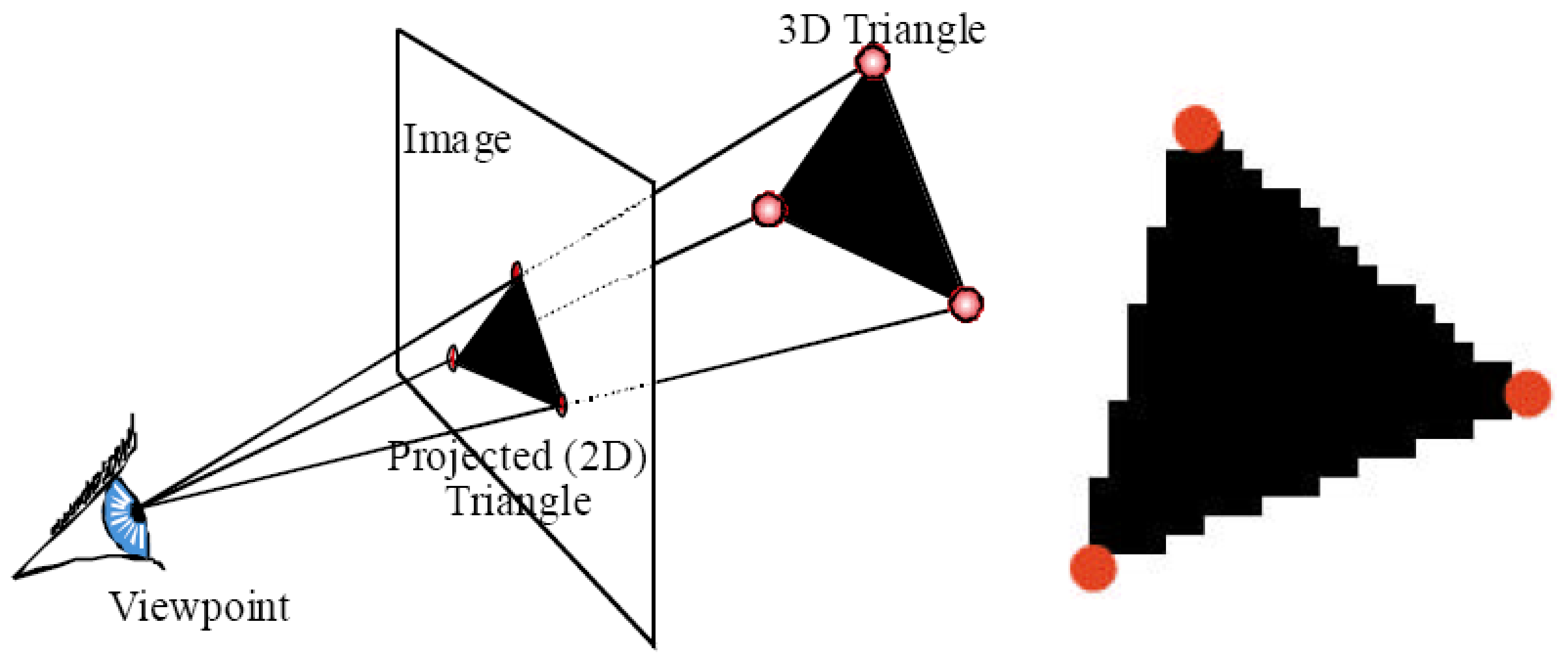
Image Order Approach to Viewing



[Durand]

- Ray generation produces rays, not points in scene
- Cast a ray at every pixel and find points of intersection
- This is called ray casting and ray tracing

Object Order Approach to Viewing



[Durand]

Projection (left) and rasterization (right) of a triangle.

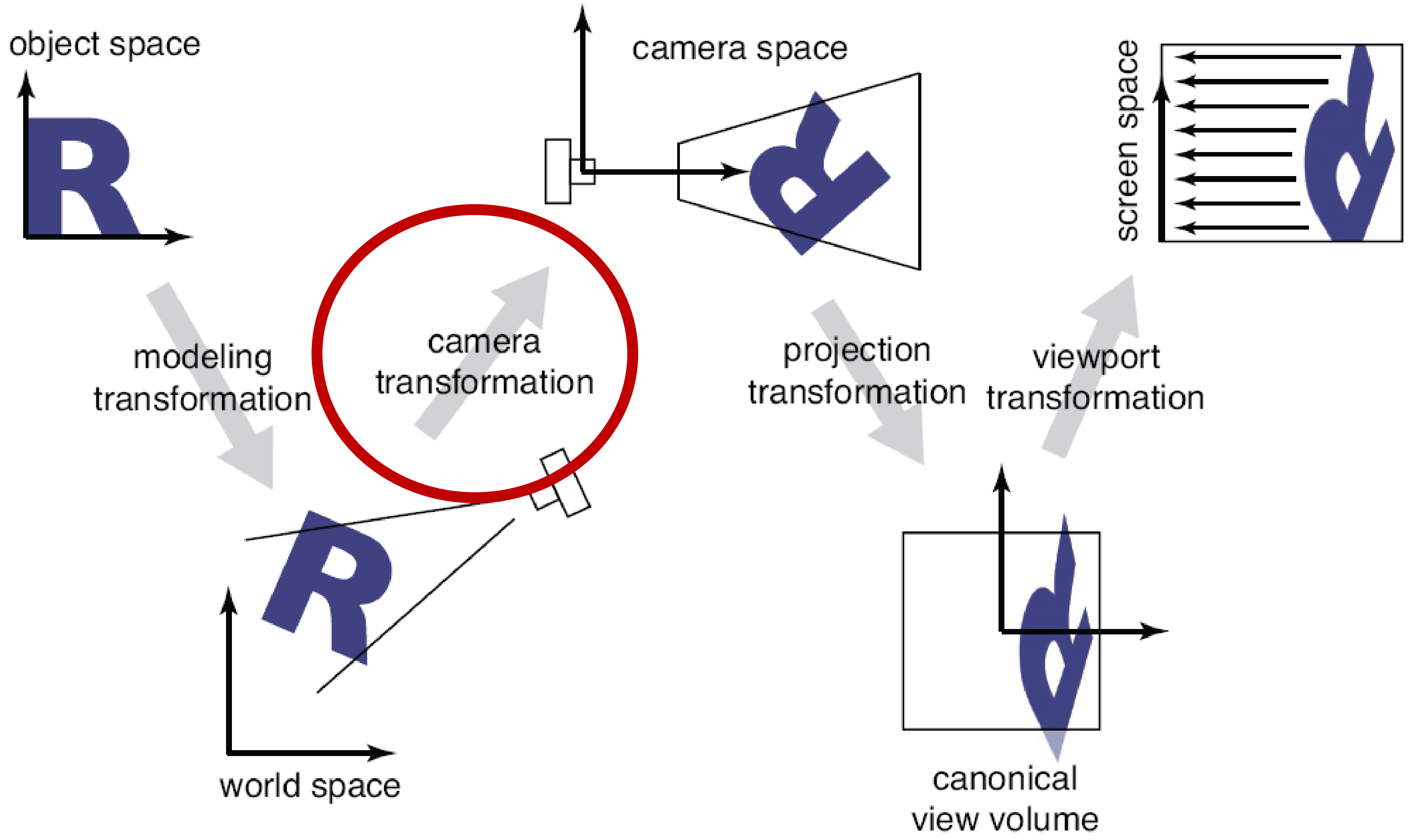
- Inverting the ray tracing process requires division for the perspective case
- Once triangle vertices are known in screen coordinates the triangle can be filled in (rasterization… more on this later)

Mathematics of projection

- Always work in eye coordinates
  - Assume eye point at the origin and plane perpendicular to z
- Orthographic case
  - A simple projection: just toss out z
- Perspective case: scale diminishes with z
  - And increases with d

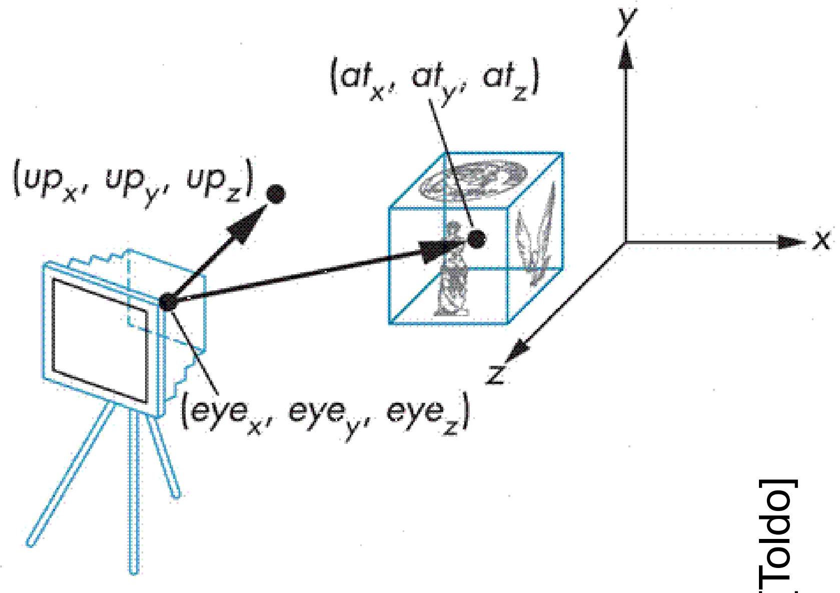
Pipeline of transformations

- Standard sequence of transforms



Camera / Eye Coordinates

- We have discussed object to world transformations
- Need world to camera transformation
  - In viewing, we typically know:
    - Where the camera is
    - What we want to look at
    - Which way is up
  - Need a rigid transformation (rotation and translation)
  - How many degrees of freedom?
  - How to compute it?
  - In OpenGL we can use gluLookat



[Toldo]

Based on slides by Steve Marschner

“Lookat” Transformation

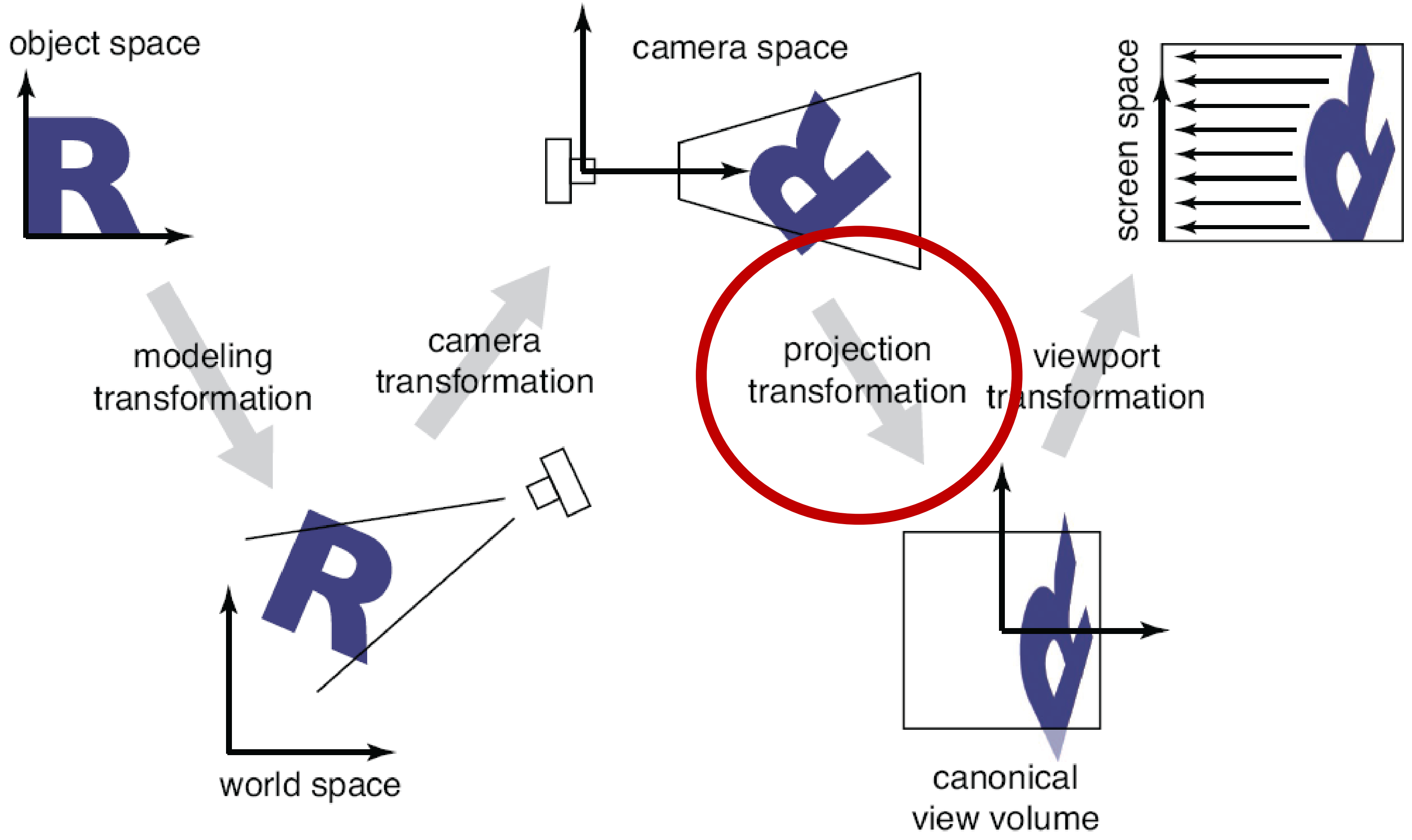
- uvn or uvw commonly used for xyz camera axes
- Compute transform from points *e, l,* and vector V_{up}
  - *e* is the eye point, or view reference point
  - *l* is the lookat point
- Separate the rotation *R* and translation *T*
  - What order to compose? What is easiest?
  - What is *T*?
  - What is *R*?
- We want

$$\begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1}$$

In OpenGL, see gluLookAt

Pipeline of transformations

- Standard sequence of transforms

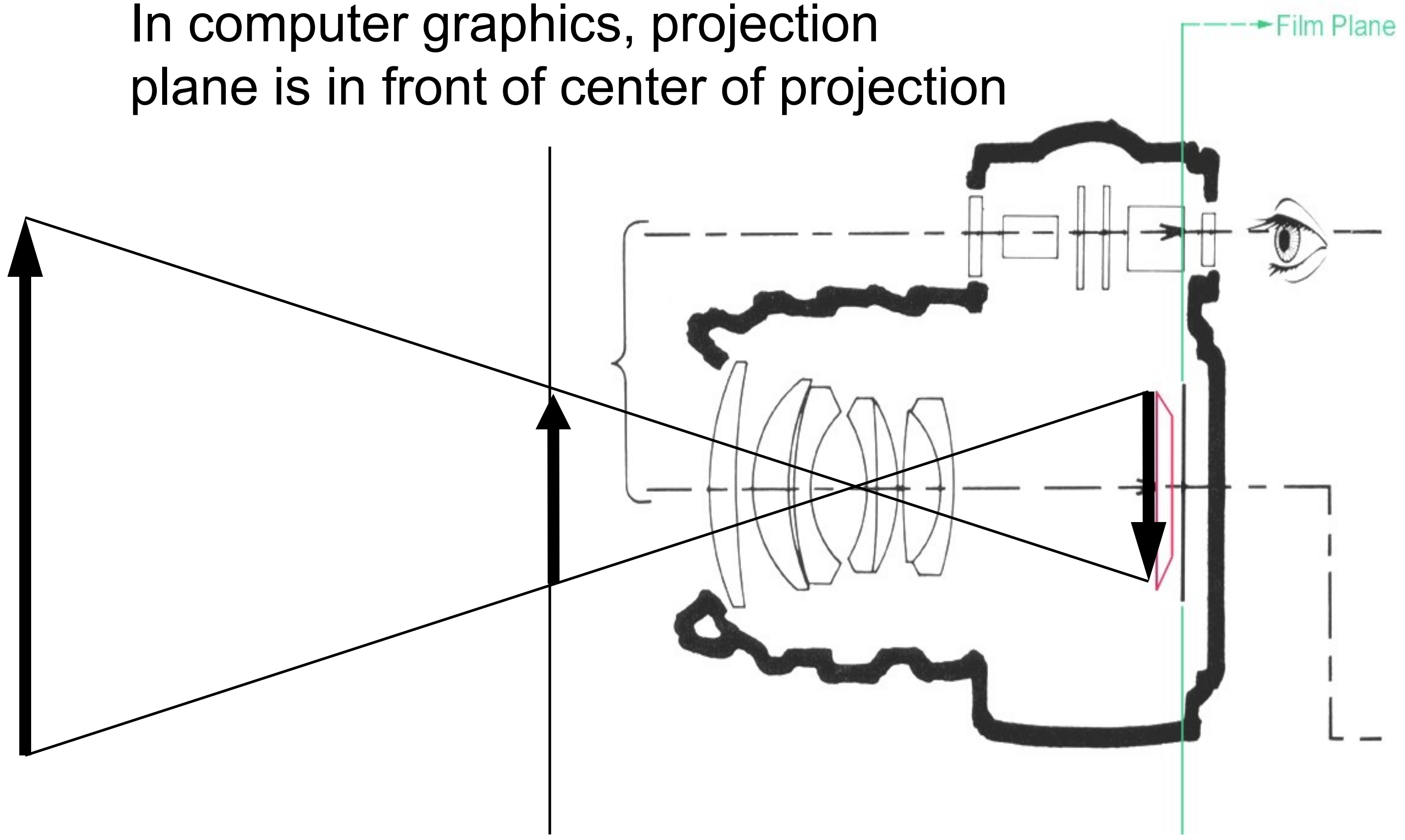


Based on slides by Steve Marschner

Projection

- How to form an image by planar perspective projection?

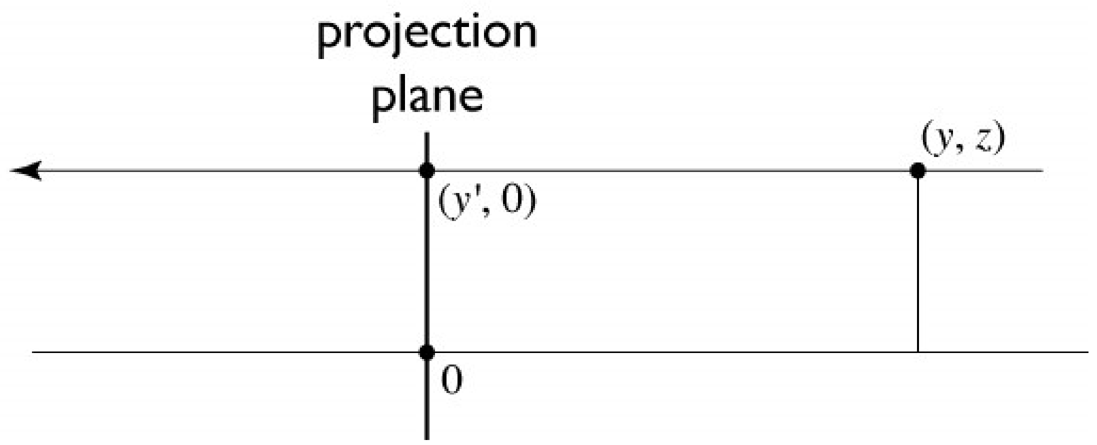
In computer graphics, projection plane is in front of center of projection



[Source unknown]

Based on slides by Steve Marschner

Parallel projection: orthographic

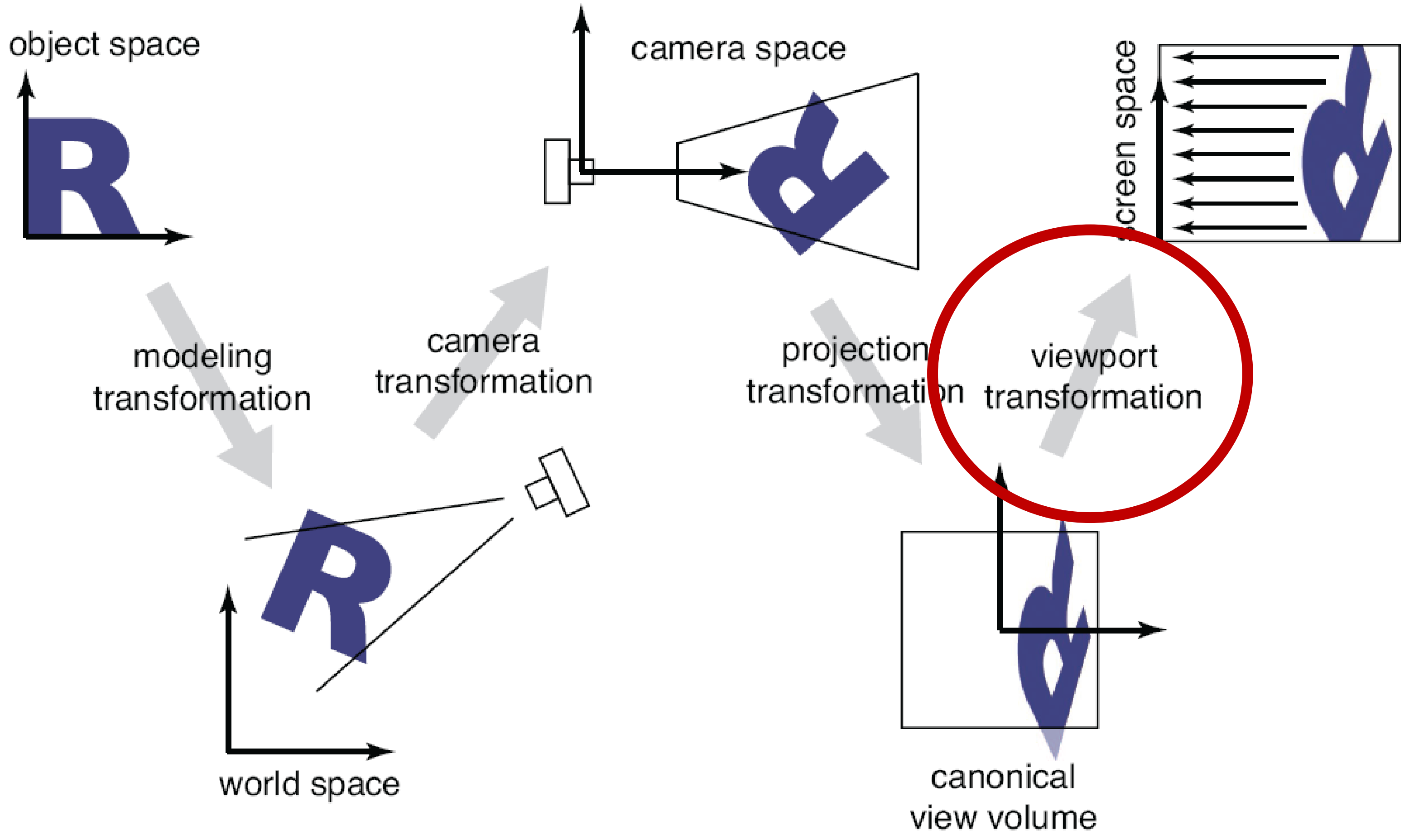


But let us start with something simpler… orthographic projection
to implement orthographic, just toss out *z*:

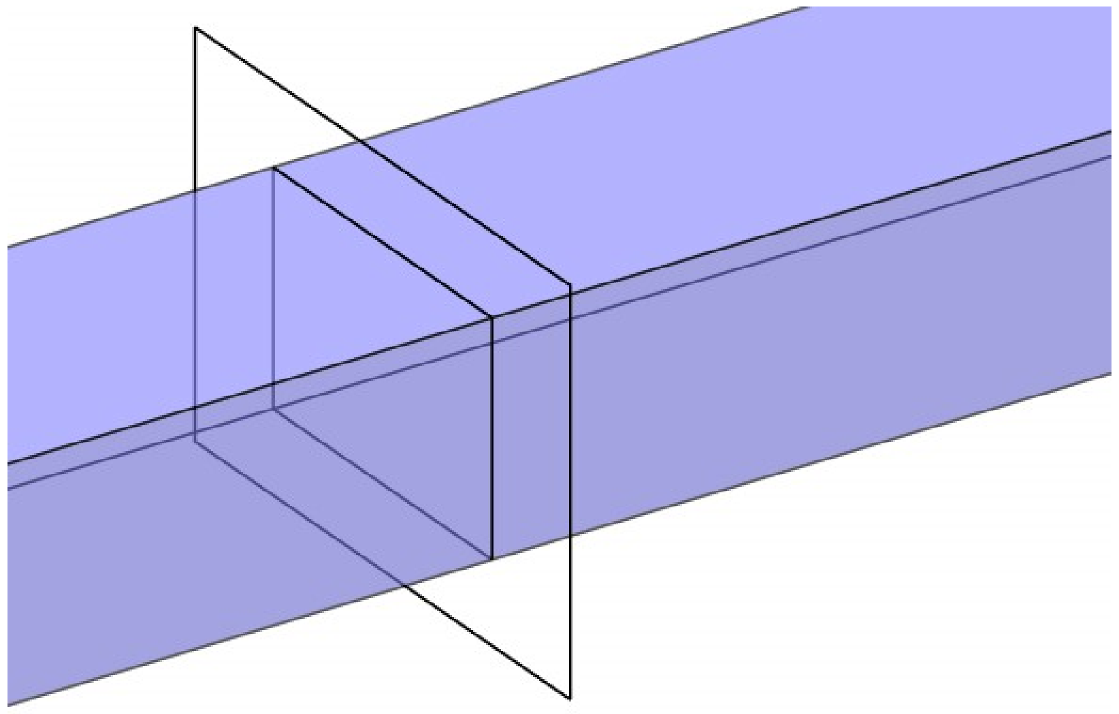
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Pipeline of transformations

- Standard sequence of transforms



View volume: orthographic



Based on slides by Steve Marschner

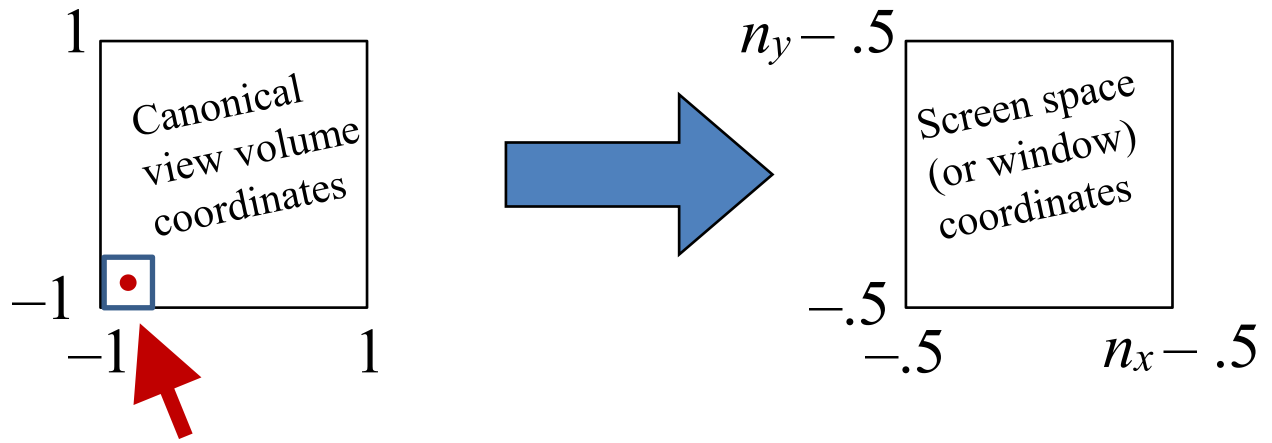
Viewing a cube of size 2

- Start by looking at a restricted case: the canonical view volume
- It is the cube $[-1,1]^3$, viewed from the z direction
- Matrix to project it into a square image in $[-1,1]^2$ is trivial

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Viewing a cube of size 2

- To draw in image, need coordinates in pixel units
- Suppose n_x pixels in x direction and n_y pixels in y direction

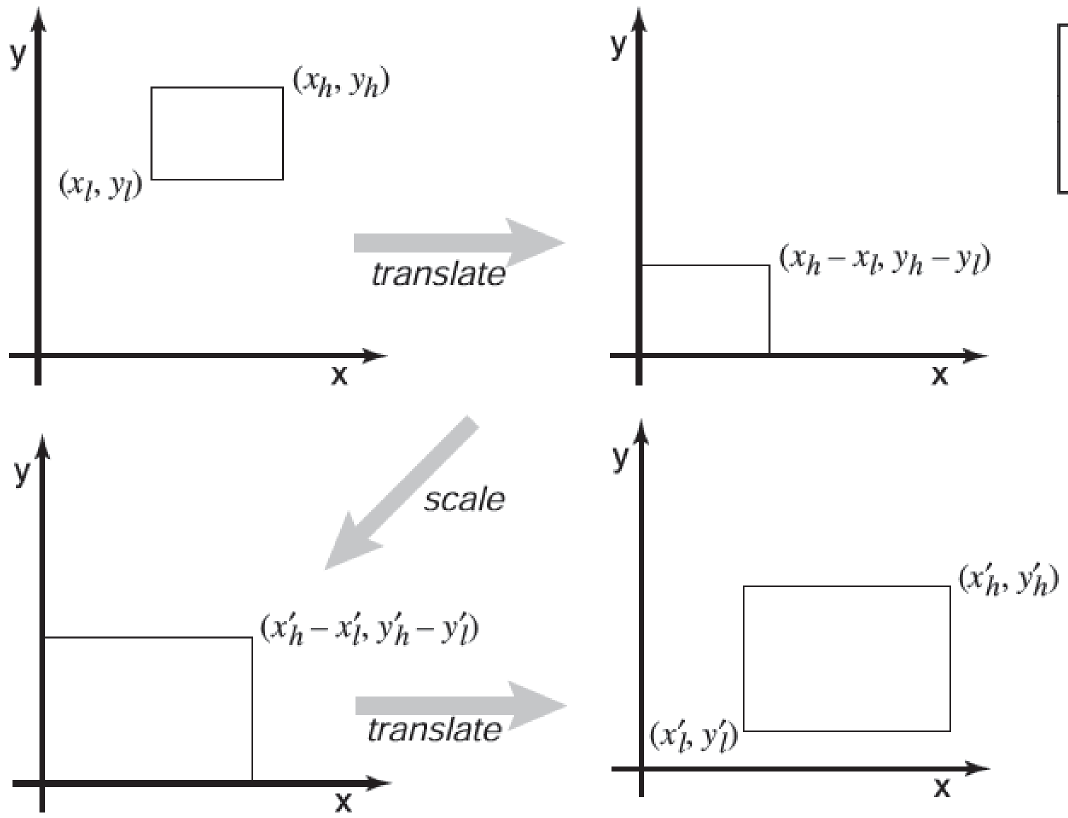


1
Canonical view volume coordinates
–1
–1
1
$n_y - .5$
Screen space (or window) coordinates
–.5
–.5
$n_x - .5$

Pixel size in canonical view volume is $2/n_x$ by $2/n_y$
Center is at ½ pixel width and height away from $(-1,-1)$
i.e., $(-1+1/n_x, -1+1/n_y)$ maps to $(0,0)$ integer pixel location
$(1-1/n_x, 1-1/n_y)$ maps to $(nx-1, ny-1)$ integer pixel location

Windowing transforms

- This transformation is worth generalizing



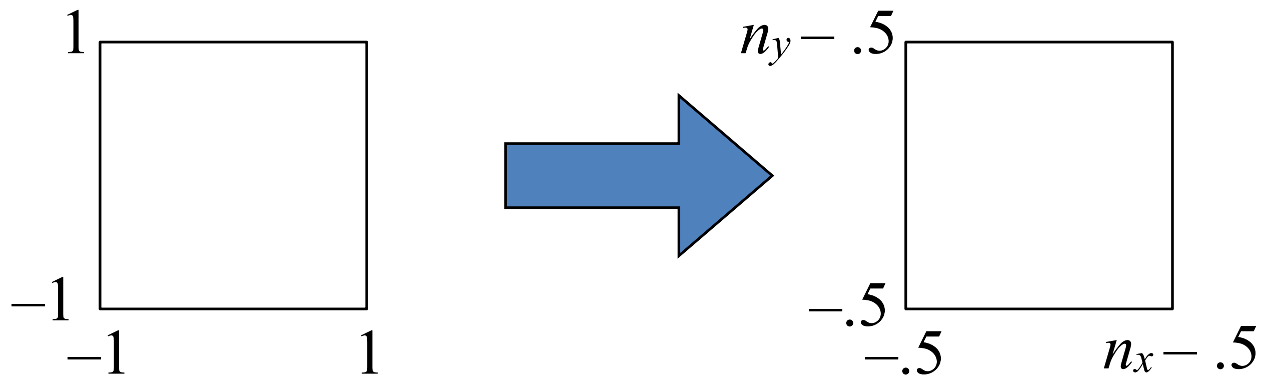
$$\begin{bmatrix} 1 & 0 & x'_l \\ 0 & 1 & y'_l \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & 0 \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_l \\ 0 & 1 & -y_l \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & \frac{x'_l x_h - x'_h x_l}{x_h - x_l} \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & \frac{y'_l y_h - y'_h y_l}{y_h - y_l} \\ 0 & 0 & 1 \end{bmatrix}$$

Take one axis-aligned rectangle or box to another, a useful transformation chain!

[Shirley3e f. 6-16; eq. 6-6]

Viewport transformation



$$\begin{bmatrix} x_{\text{screen}} \\ y_{\text{screen}} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{n_x}{2} & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & \frac{n_y-1}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\text{canonical}} \\ y_{\text{canonical}} \\ 1 \end{bmatrix}$$

$(-1+1/n_x, -1+1/n_y)$ maps to (0,0)

$(1-1/n_x, 1-1/n_y)$ maps to (n_x-1, n_y-1)

(In OpenGL 4.0, glViewport sets up this transform)

Viewport transformation

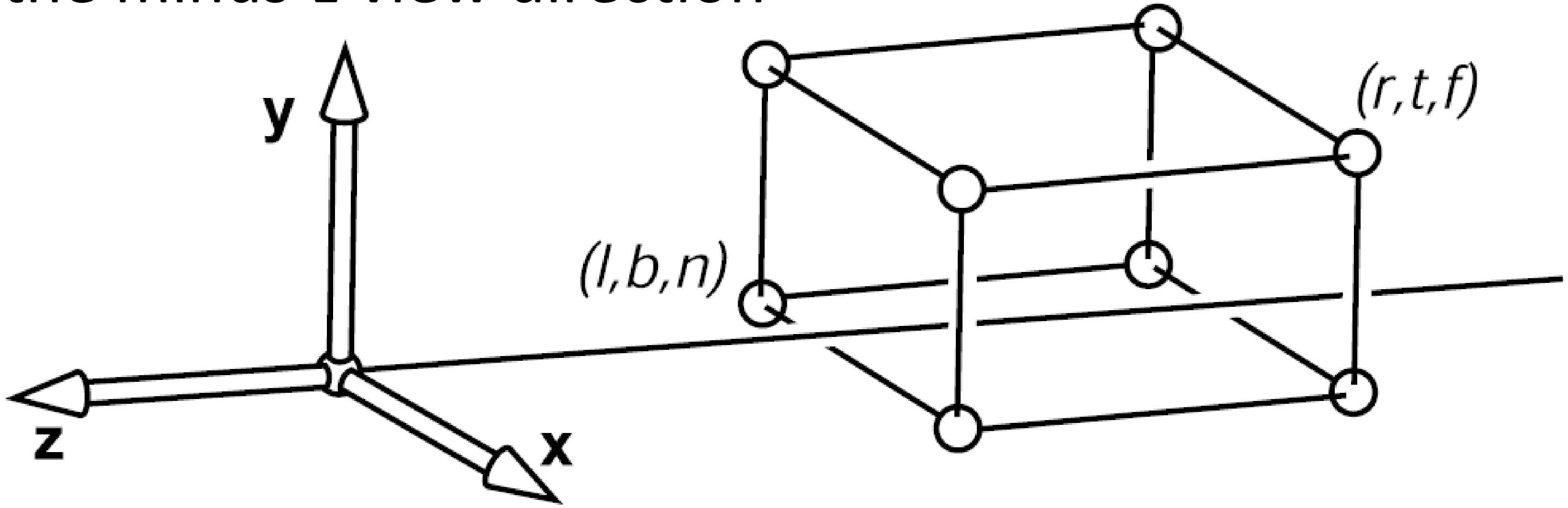
- In 3D, carry along *z* for the ride
  - one extra row and column

$$\mathbf{M}_{\text{vp}} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x - 1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y - 1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Based on slides by Steve Marschner

Orthographic projection

- First generalization: different view rectangle
  - – retain the minus-z view direction



  - – specify view by left, right, top, bottom (as in RT)
  - – also near, far

Clipping planes

- Recall…
  - Object-order rendering considers each object in turn, i.e., forward rendering, rasterization
  - Image-order rendering considers each pixel in turn, i.e., backward rendering, ray tracing
- In object-order systems we **always** use at least two *clipping planes* that further constrain the view volume
  - near plane: parallel to view plane; things between it and the viewpoint will not be rendered
  - far plane: also parallel; things behind it will not be rendered
- These planes are:
  - partly to *remove unnecessary stuff* (e.g., behind the camera)
  - but really to *constrain the range of depths* (we'll see why later)

Orthographic projection

- We can implement this by mapping the view volume to the canonical view volume.
- This is just a 3D windowing transformation!

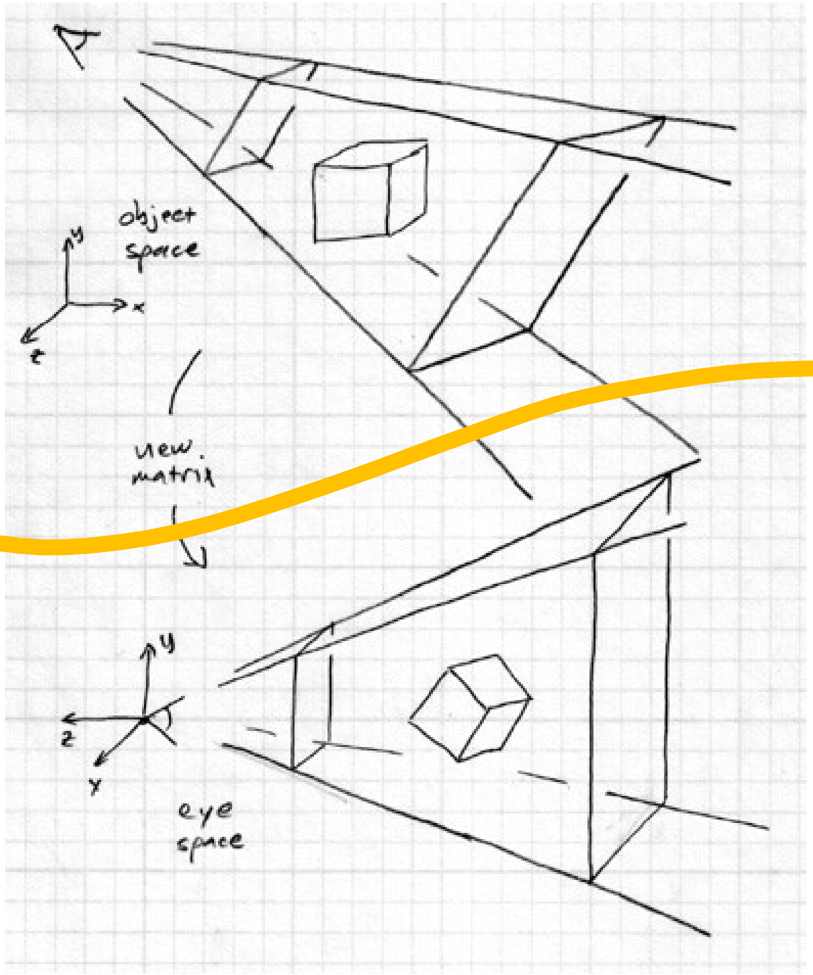
$$\begin{bmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & 0 & \frac{x'_l x_h - x'_h x_l}{x_h - x_l} \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & 0 & \frac{y'_l y_h - y'_h y_l}{y_h - y_l} \\ 0 & 0 & \frac{z'_h - z'_l}{z_h - z_l} & \frac{z'_l z_h - z'_h z_l}{z_h - z_l} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{M}_{\text{orth}} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Camera and modeling matrices

- We worked out all the preceding transforms starting from eye coordinates
  - before we do any of this, we must transform into that space
- Transform from world (canonical) to eye space is called the viewing matrix
  - Easy for us to compute the matrix F_c which takes us from eye space to canonical space, but here we use the inverse F_c^{-1}
- Remember that geometry would originally have been in the object's local coordinates;
  - Transform into world coordinates is called the modeling matrix, M_m
- Note some systems (e.g., OpenGL) combine the two into a modelview matrix and just skip world coordinates

Viewing transformation



the camera matrix rewrites all coordinates in eye space

Based on slides by Steve Marschner

Orthographic transformation chain

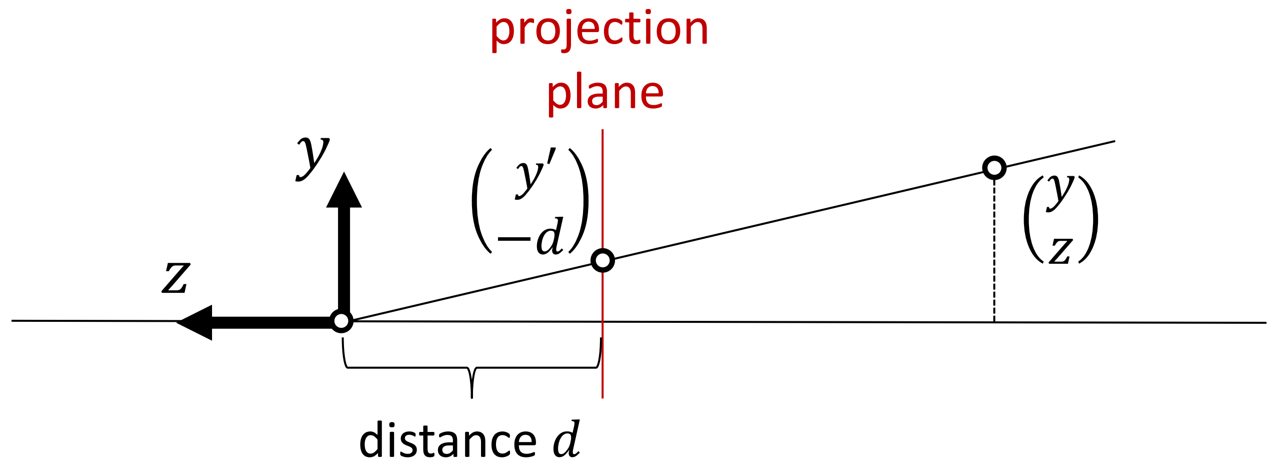
- Start with coordinates in object's local coordinates, p_o
  - Transform into world coordinates (modeling transform, M_m)
  - Transform into eye coordinates (camera transform, $M_{cam} = F_c^{-1}$)
  - Orthographic projection, M_{orth}
  - Viewport transform, M_{vp}

$$p_s = M_{vp}\, M_{orth}\, M_{cam}\, M_m\, p_o$$

$$\begin{bmatrix} x_s \\ y_s \\ z_c \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y-1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{OpenGL set with glViewport}} \underbrace{\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{Projection Matrix}} \underbrace{\begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \mathrm{M_m}}_{\text{Modeling Matrix}} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

Based on slides by Steve Marschner

Planar Perspective projection


projection plane
y
z
$\begin{pmatrix} y' \\ -d \end{pmatrix}$
$\begin{pmatrix} y \\ z \end{pmatrix}$
distance d

With the Projection plane at distance d, note the similar triangles:

$$\frac{y'}{d} = \frac{y}{-z}$$
$$y' = -dy/z$$

- y’ is the foreshortened version of y, that is, it is smaller by a factor $-d/z$
- We can think of dividing by negative z so that y’ has the same sign as y!

Homogeneous coordinates revisited

- Perspective requires division
  - that is not part of affine transformations
  - in affine, parallel lines stay parallel
    - therefore no vanishing point
    - therefore no rays converging on viewpoint
- "True" purpose of homogeneous coords: projection

Homogeneous coordinates revisited

- Introduced *w* = 1 coordinate as a placeholder

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \to \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

  – used as a convenience for unifying translation with linear

- Can also allow arbitrary non-zero *w*

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \sim \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix}$$

Implications of w

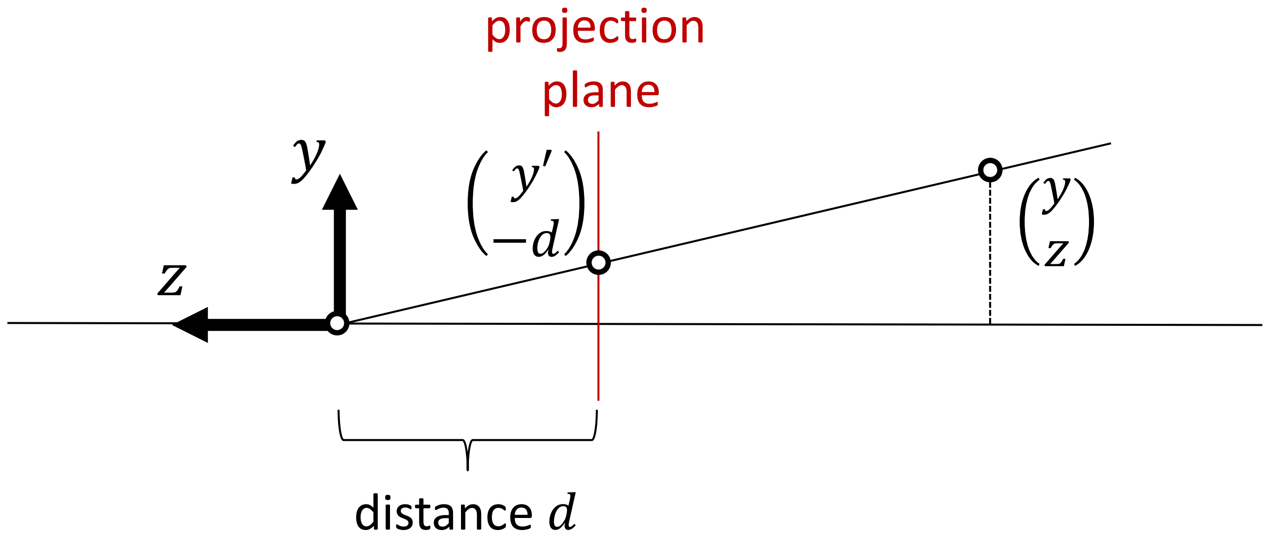
$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \sim \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix}$$

- All scalar multiples of a 4-vector are equivalent
- When w is not zero, can divide by w
  - Therefore these points represent "normal" affine points
- When w is zero, it's a point at infinity, i.e., a direction
  - Can think of this as the point where parallel lines intersect
  - Can also think of it as the vanishing point

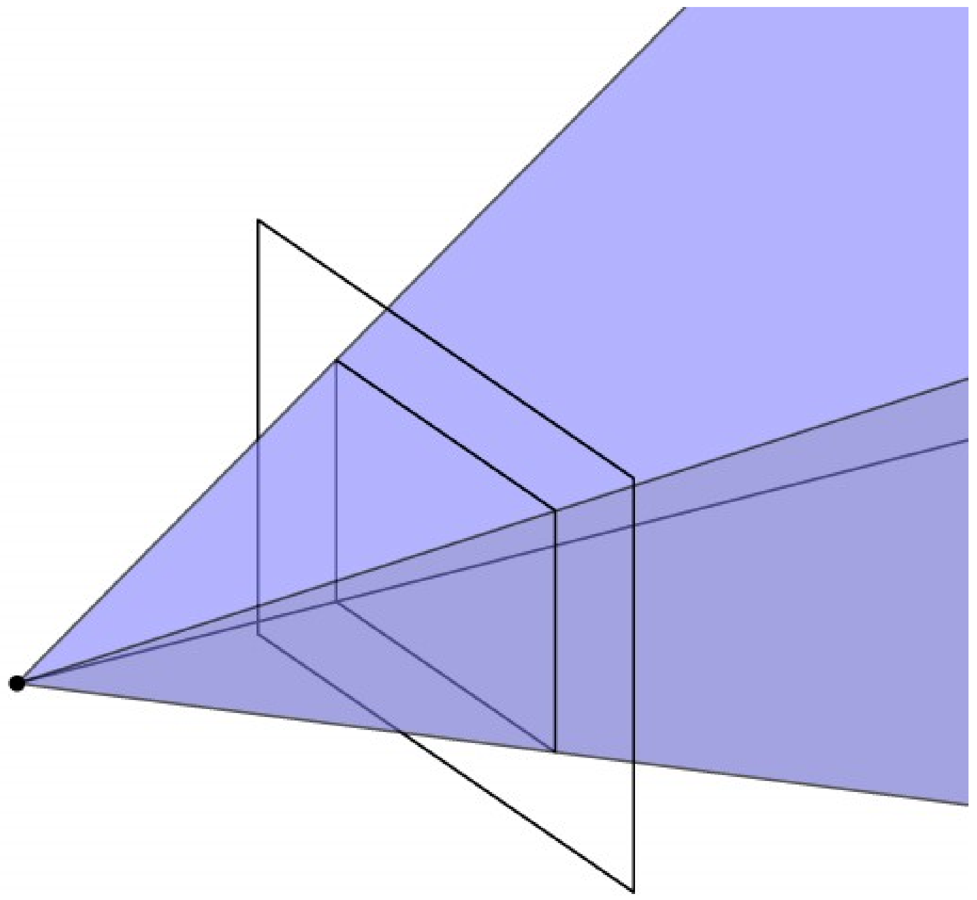
Perspective Projection

- To implement perspective, put $-z$ into w

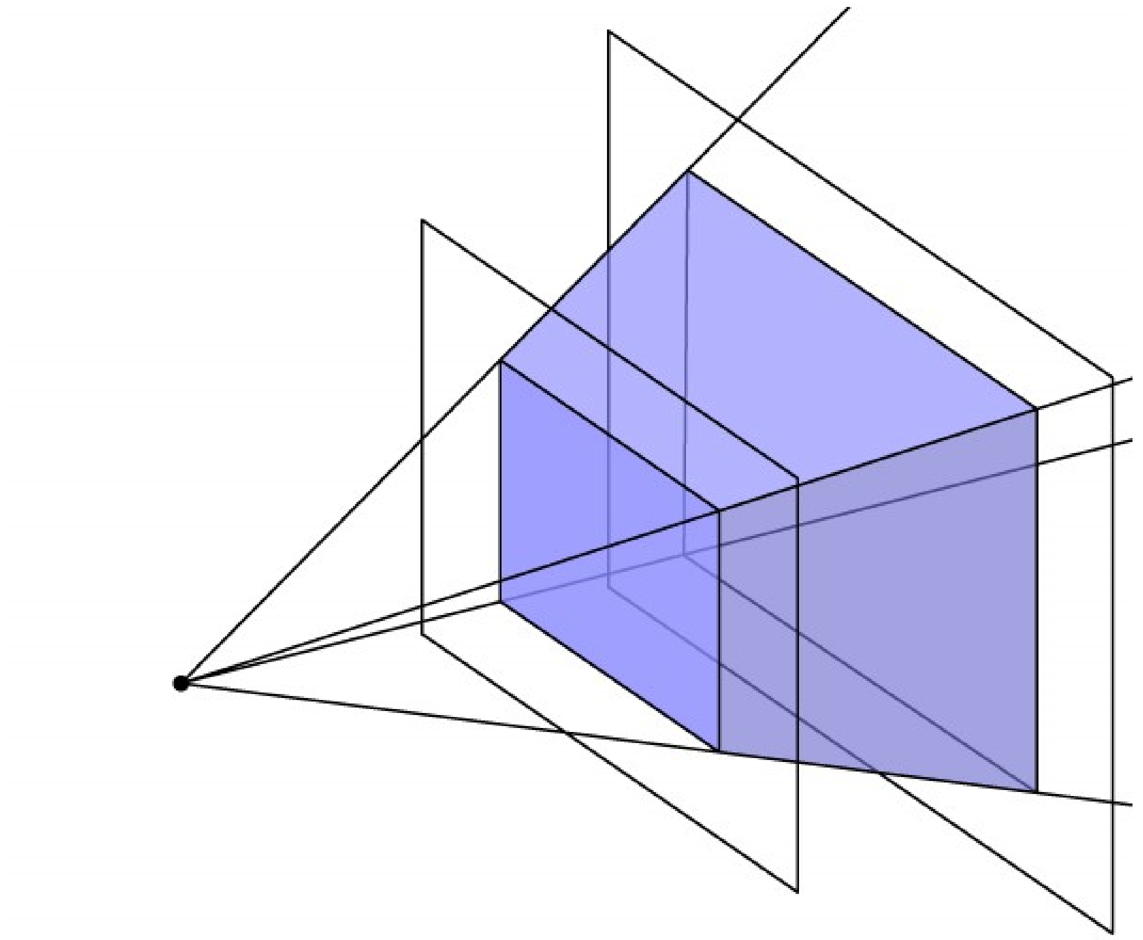
$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} -dx/z \\ -dy/z \\ 1 \end{pmatrix} \sim \begin{pmatrix} dx \\ dy \\ -z \end{pmatrix} = \begin{pmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$


projection plane
y
z
$\begin{pmatrix} y' \\ -d \end{pmatrix}$
$\begin{pmatrix} y \\ z \end{pmatrix}$
distance d

View volume: perspective



View volume: perspective (clipped)



Perspective Projection

Carrying depth through perspective

- Perspective has a varying denominator,
  - We can't preserve depth!
- As a compromise we can preserve depth order, and depth on near and far planes
  - Let n and f be the ***distance*** (+ve) to the near and far plane
  - Let the projection plane be at distance n (That is $d = n$)

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} \sim \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ -z \end{pmatrix} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

  - Choose a and b so that $z'(-n) = -n$ and $z'(-f) = -f$.

Carrying depth through perspective

$$z' = \frac{az + b}{-z}$$

$$(-n) = \frac{a(-n) + b}{-(-n)}$$ Plug in $-n$ for z

$$-n^2 = b - an$$ Let this be Equation 1

$$-f^2 = b - af$$ Have similar expression for f

$$-(n^2 - f^2) = -a(n - f)$$ Subtract previous two lines

$$(n + f) = a$$

$$-n^2 = b - (n + f)n$$ Replace a in Equation 1

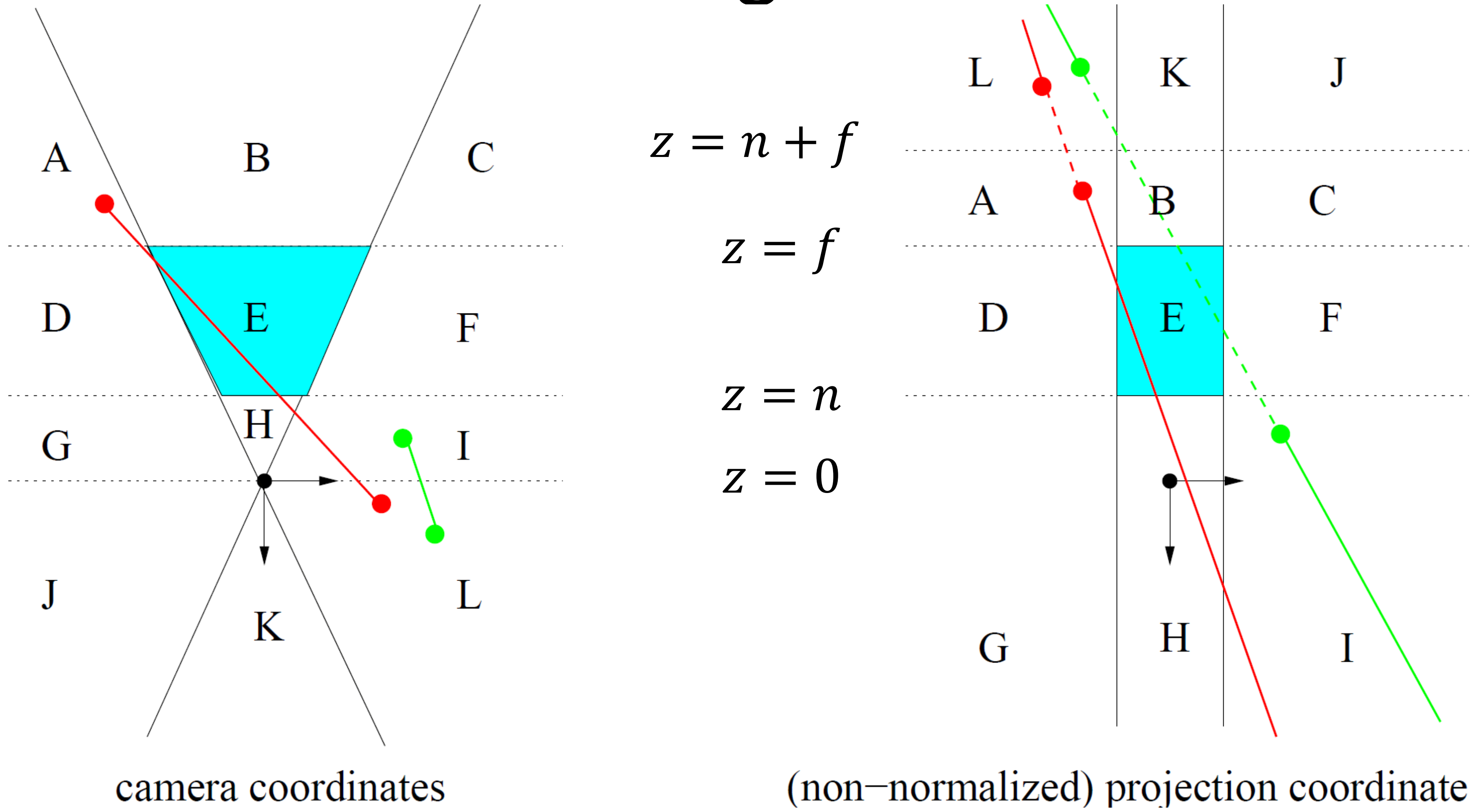
$$-n^2 = b - n^2 - nf$$

$$nf = b$$

Questions

- Questions to better understand the action of a 4x4 homogenous planar perspective transformation matrix
  - What happens to vectors $(x, y, z, 0)$?
    - They map to plane z=near+far
  - What happens to the origin $(0,0,0,w)$?
    - Maps to $(0,0,nfw,0)$
  - What happens to points on plane **with z normal** at center of projection ?
    - Map to points at infinity (all vectors come from the homogeneous representations of points on this plane at center of projection)
  - What happens to lines through the origin ?
    - They become parallel
  - What happens to points at $z = (-n + -f)/2$?
    - $(n^2 + f^2)/(n + f)$, that is, z is not preserved !! However, order is! (has implications for z depth precision)
  - What happens to points behind the camera ?
    - It goes in front of the camera

Aside: What goes where?



This has implications for clipping (i.e., discarding) geometry!

$[z_l, z_h]$ to $[z'_l, z'_h]$
$[-n, -f]$ to $[-1,1]$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{z'_h - z'_l}{z_h - z_l} & \dfrac{z'_l z_h - z'_h z_l}{z_h - z_l} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{1 - (-1)}{(-f) - (-n)} & \dfrac{(-1)(-f) - (1)(-n)}{(-f) - (-n)} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{2}{n - f} & \dfrac{f + n}{n - f} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Window transform for x and y too

- Need perspective projection to produce points in the canonical view volume, so combine with an orthographic projection matrix (windowing transform)
- Use left right top and bottom (on near plane)

$$M_{per} = M_{orth}P$$

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

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

Perspective transformation chain

– Transform into world coordinates (modeling transform, M_m)

– Transform into eye coords (camera transform, $M_{cam} = F_c^{-1}$)

– Perspective matrix, P

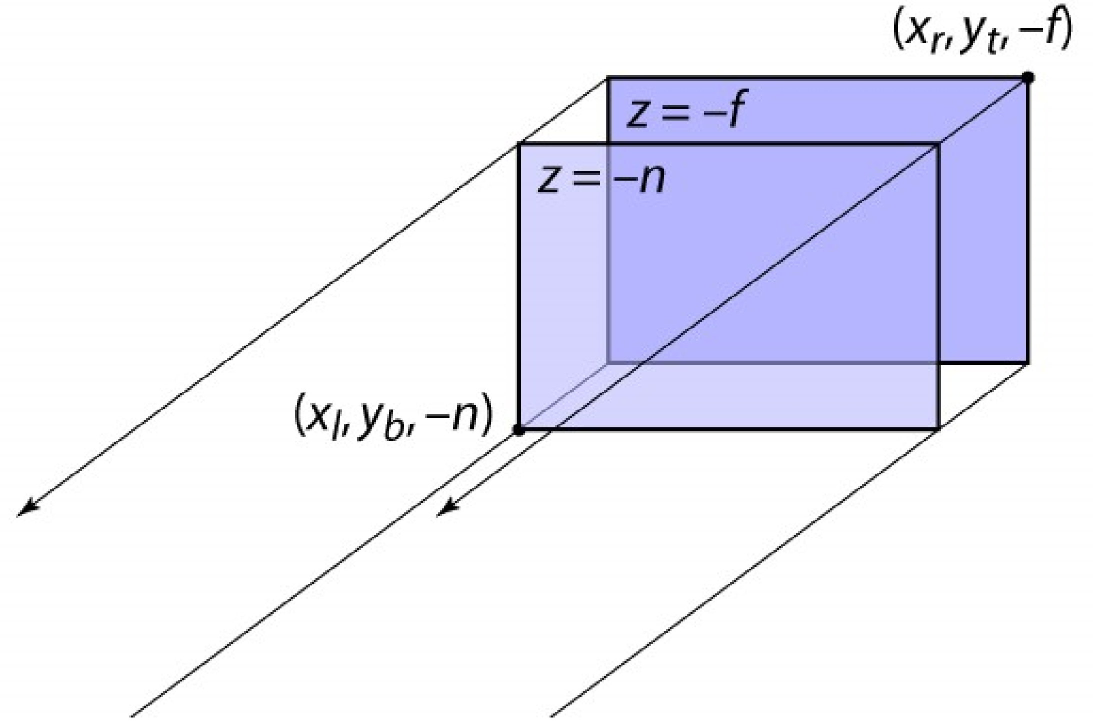
– Orthographic projection, M_{orth}

– Viewport transform, M_{vp}

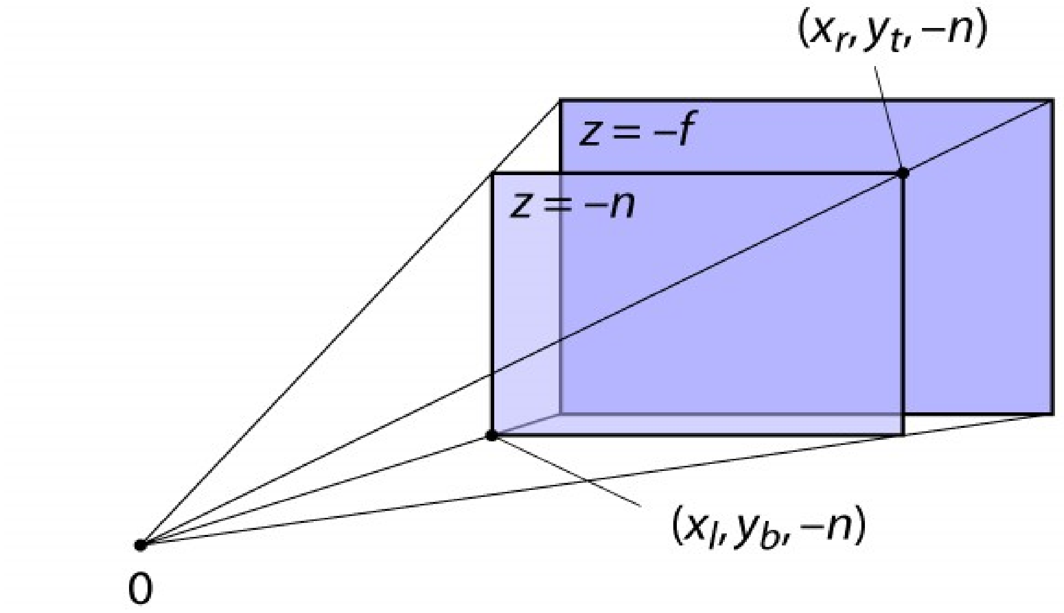
$$p_s = M_{vp}\, M_{orth}\, P\, M_{cam}\, M_m\, p_o$$

$$\begin{bmatrix} x_s \\ y_s \\ z_c \\ 1 \end{bmatrix} = \underbrace{\boxed{\begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y-1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}}_{\text{OpenGL set with glViewport}} \underbrace{\boxed{\begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & \frac{f+n}{n-f} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & nf \\ 0 & 0 & -1 & 0 \end{pmatrix}}}_{\text{Projection Matrix}} \underbrace{\boxed{\mathrm{M}_{\mathrm{cam}}\mathrm{M}_{\mathrm{m}}}}_{\text{Viewing Matrix and Modeling Matrix}} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

OpenGL view frustum: orthographic



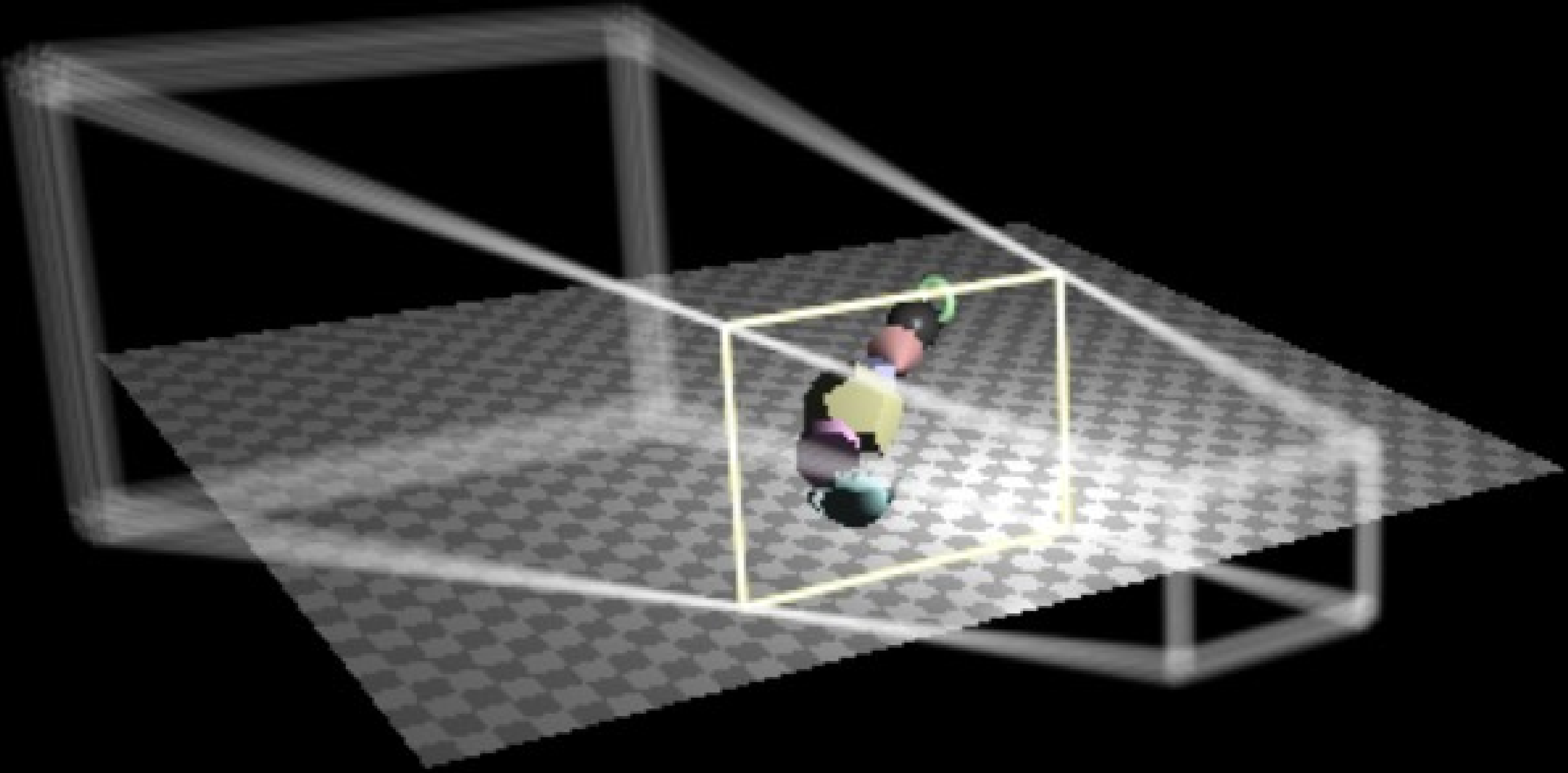
OpenGL view frustum: perspective



Note we often describe a frustum either based on field of view in the y direction and the aspect ration, or based on the left right bottom and top of the near plane rectangle. Note we often talk about distance to near and far, thus the negative z coordinates in the image above.

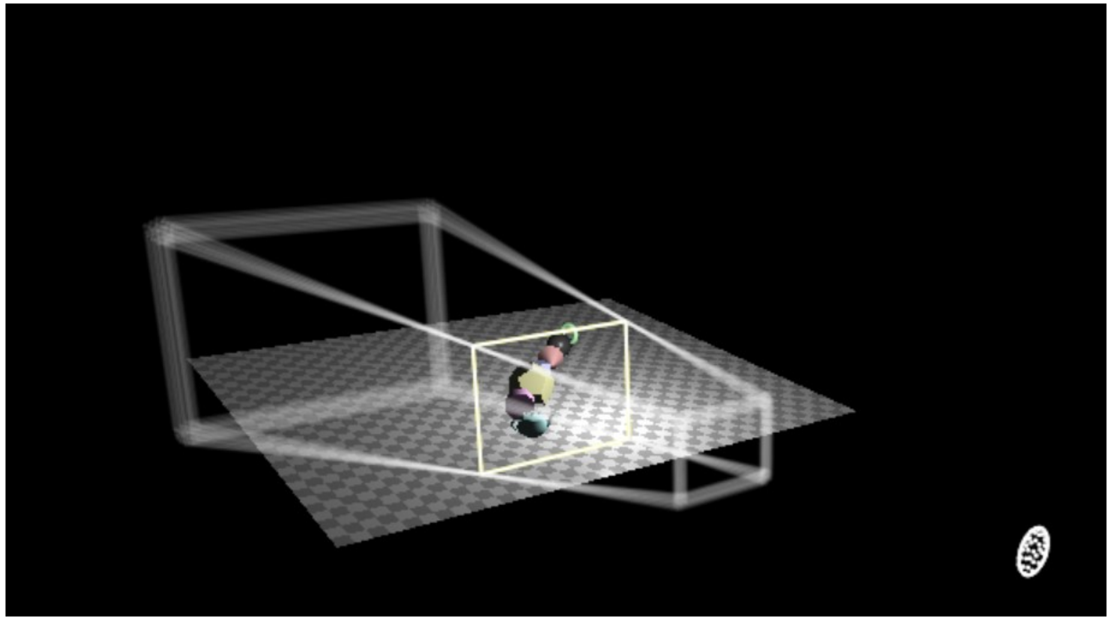
Frustum applications

- Shifted perspective
- Tiled rendering (e.g., render very high resolutions)
- 3D viewing (i.e., left eye right eye)
- Depth of field (i.e., accumulating multiple render passes)

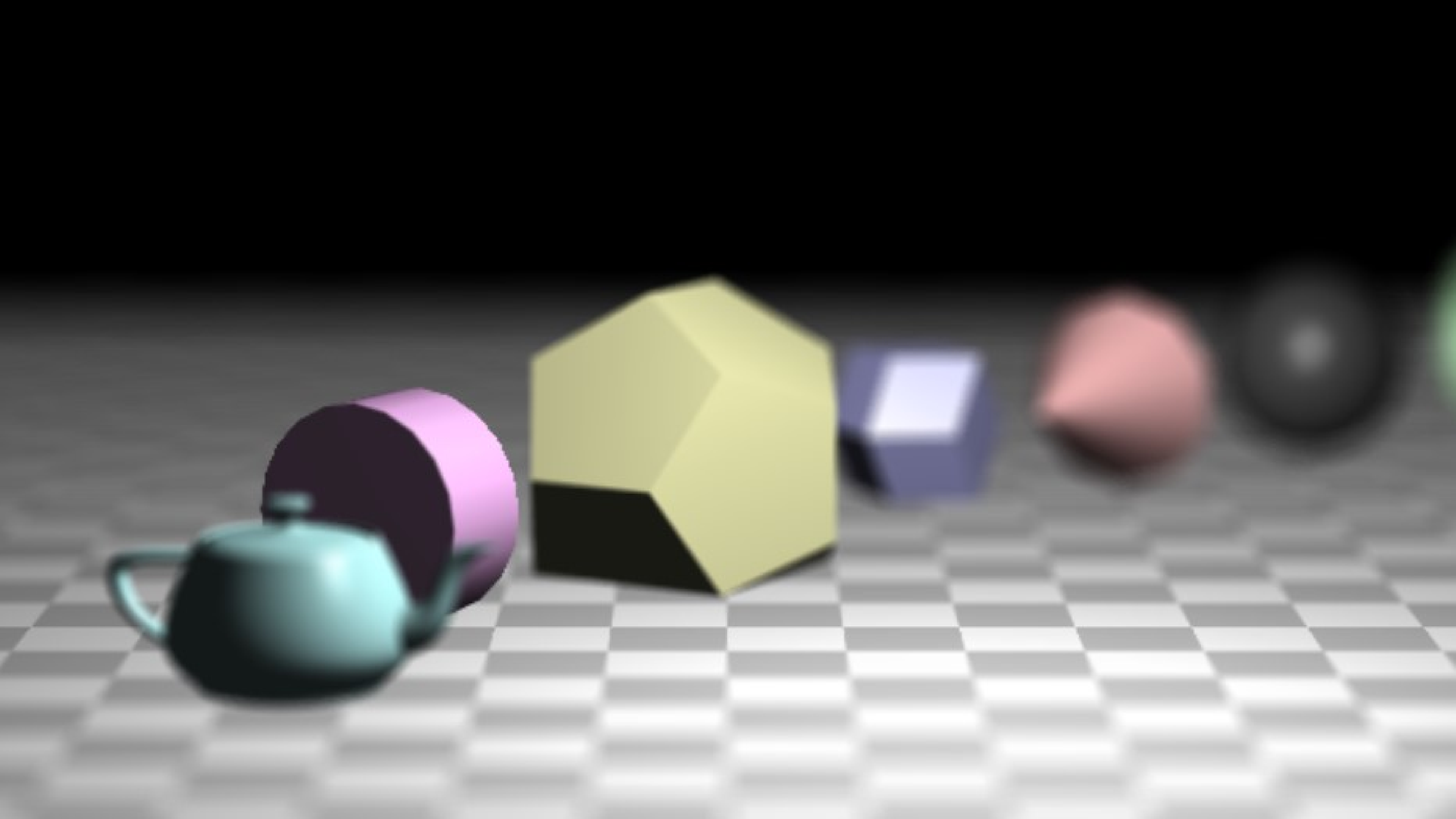


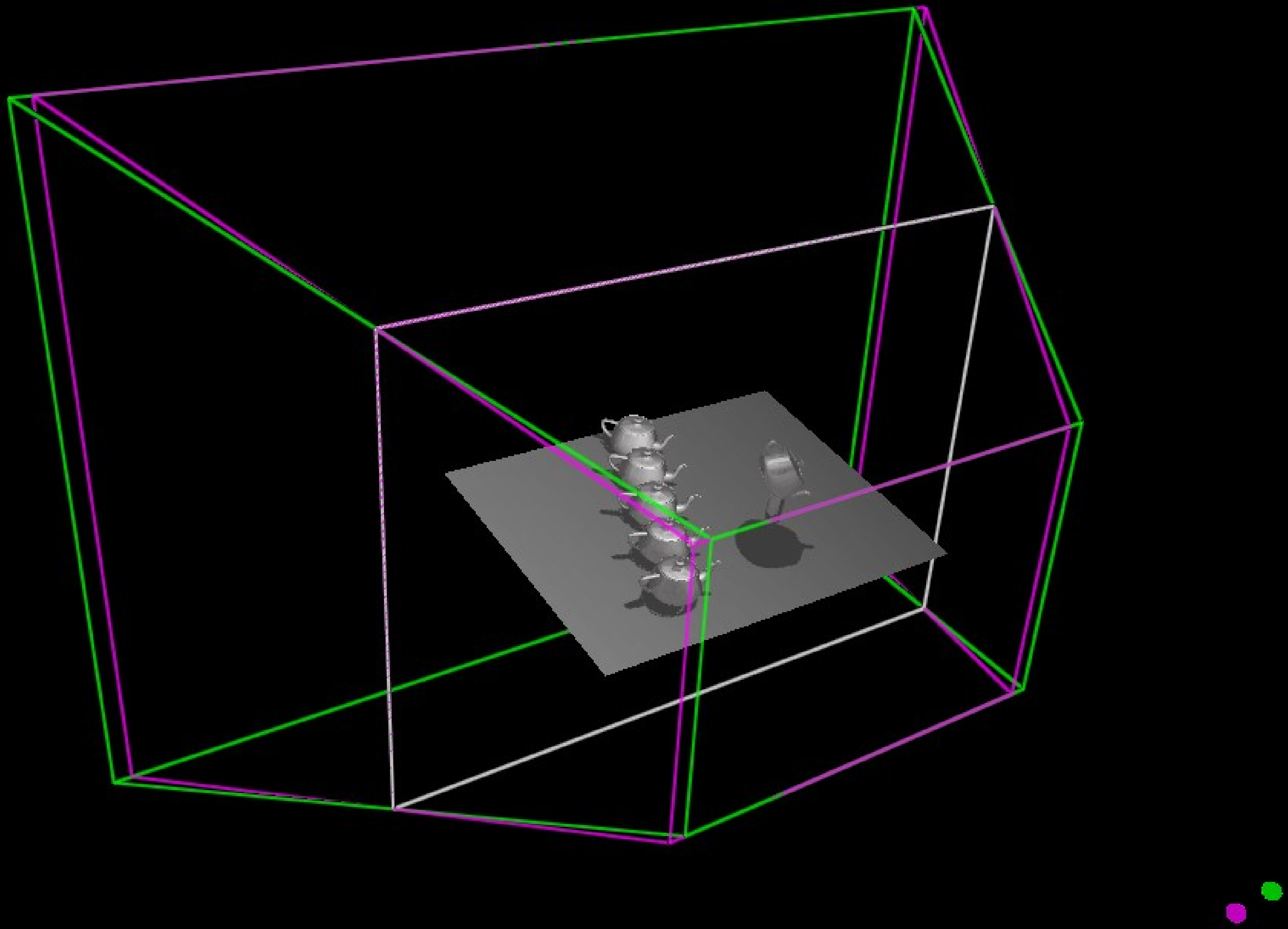
Drawing Camera Frame and Frustum

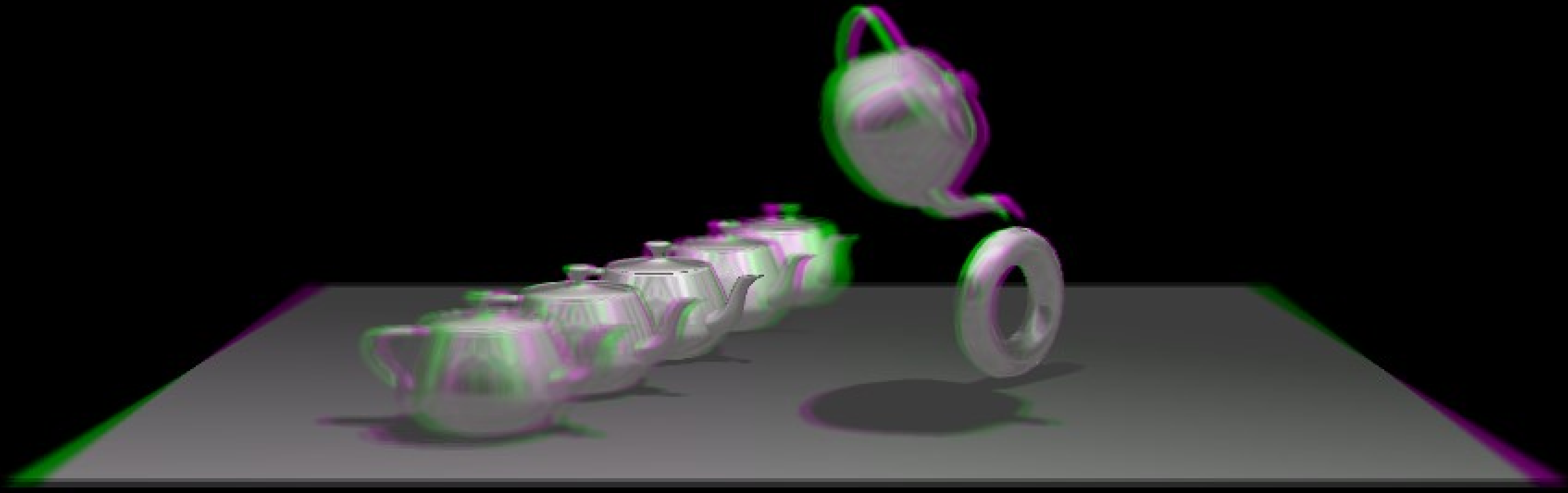
- How is this image made?
  - How many “cameras” are involved in making this image?
  - Is the last question a trick question?
  - How can you draw a camera frustum in OpenGL?
  - How can you draw the coordinate frame of the camera in OpenGL?

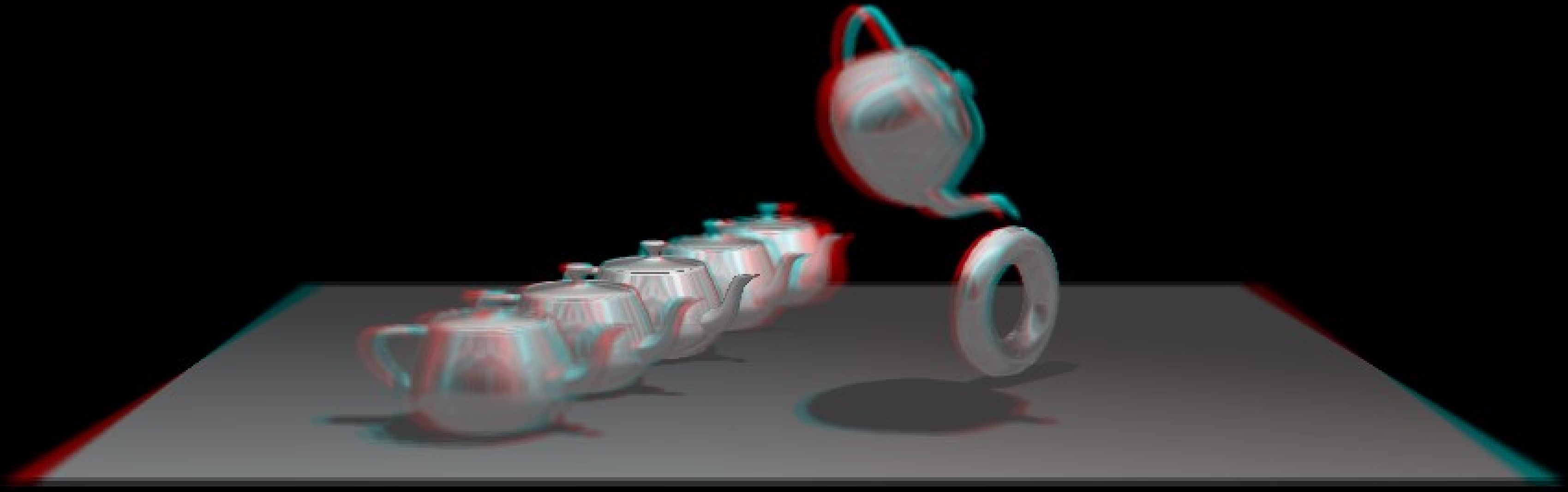


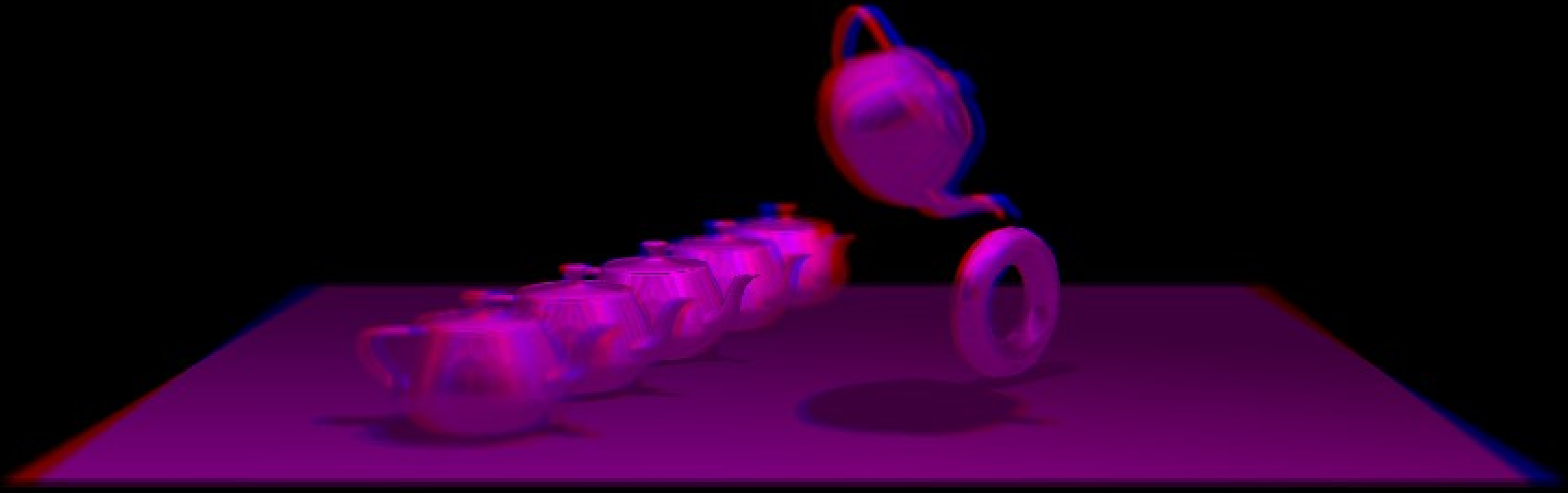
Based on slides by Steve Marschner







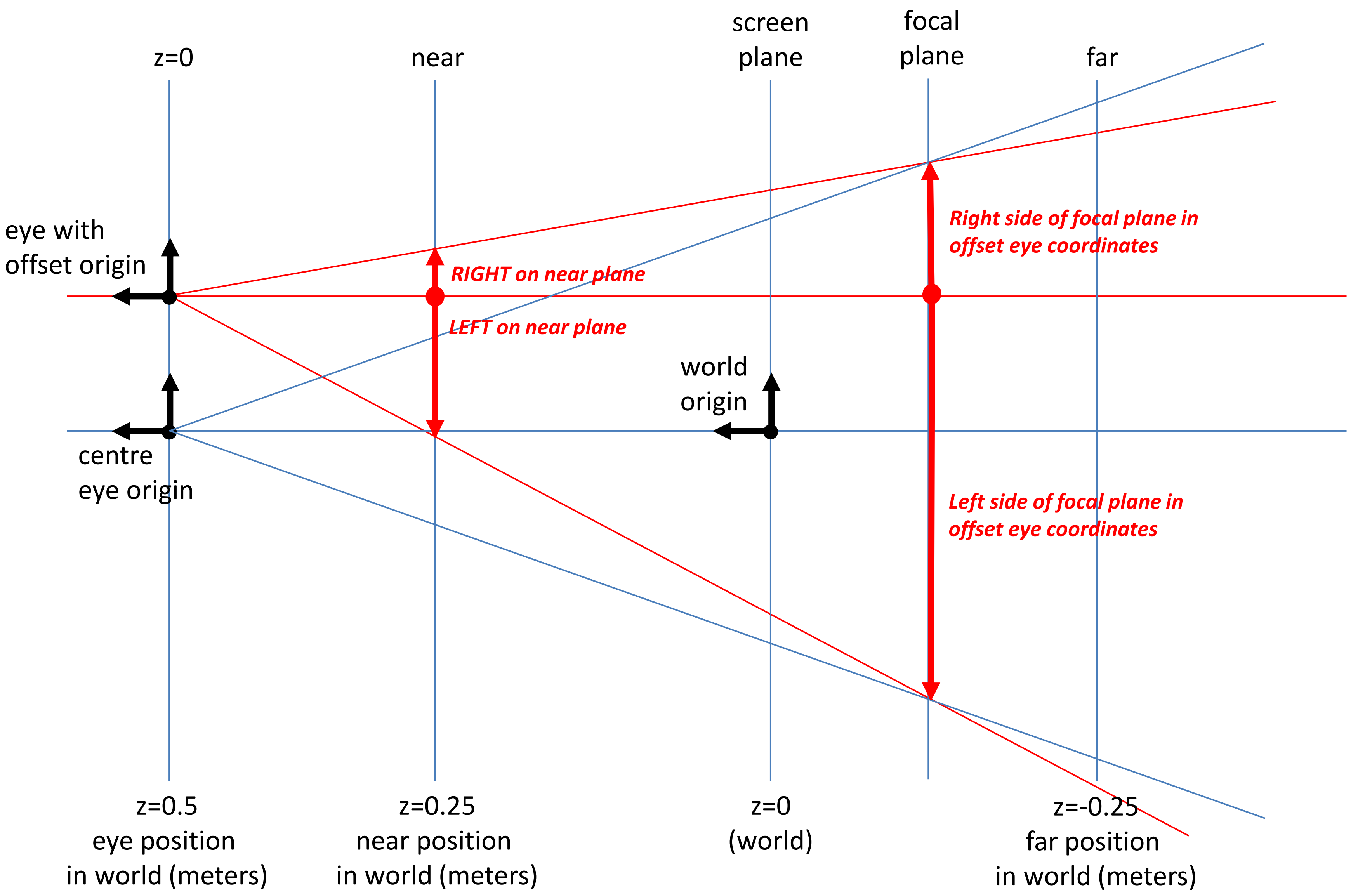




Frustums, Depth of Field, Anaglyphs

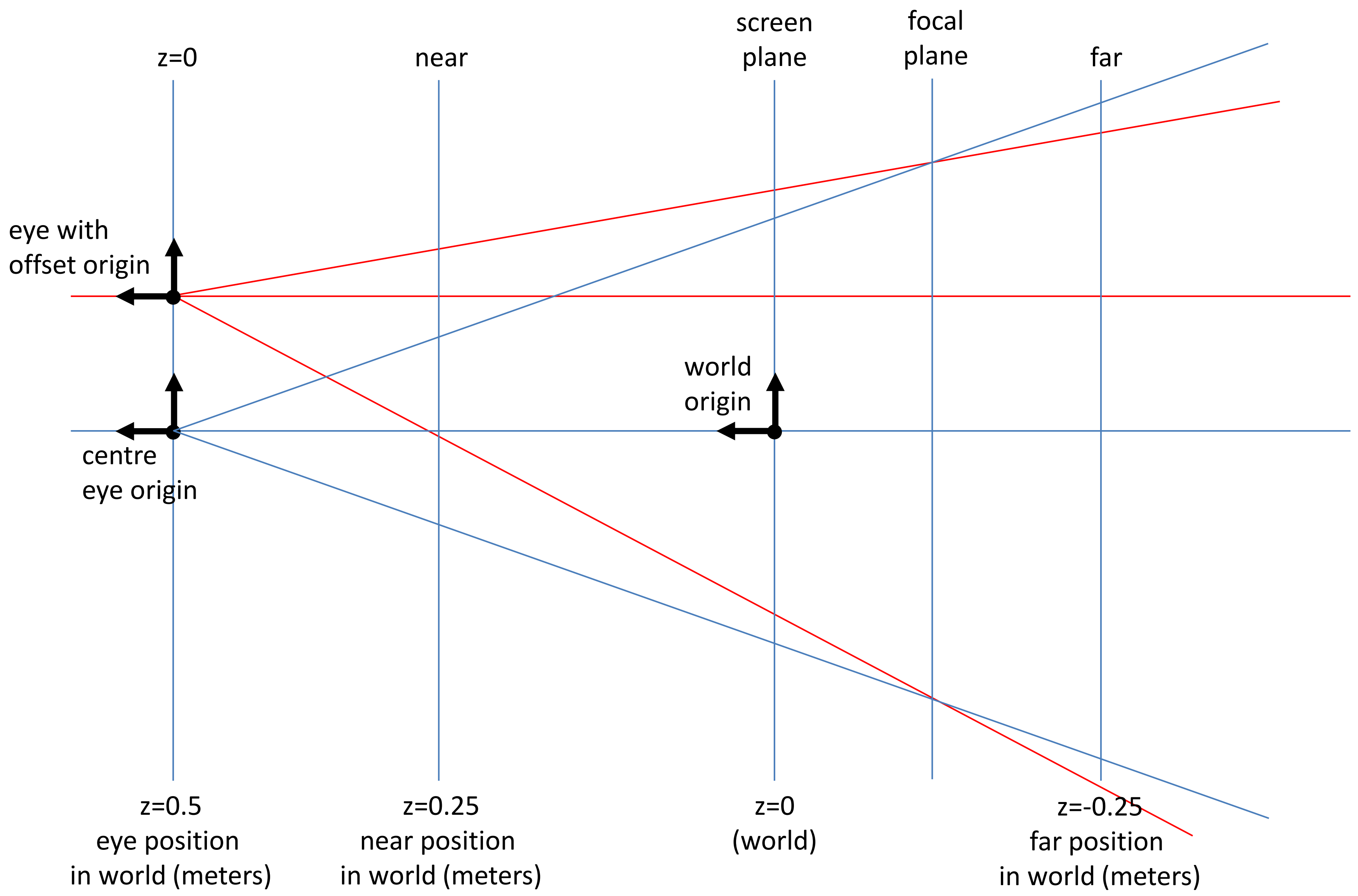
- Everything computed with similar triangles!
  - In each case, the left right top bottom on near plane are given by the left right top bottom of a well defined rectangle on some other plane.
- Need to understand which coordinate system you are using for any given quantity!

Example depth of field frustum setup…
bottom view! x axis up !



z=0
near
screen plane
focal plane
far
eye with offset origin
RIGHT on near plane
LEFT on near plane
Right side of focal plane in offset eye coordinates
world origin
centre eye origin
Left side of focal plane in offset eye coordinates
z=0.5 eye position in world (meters)
z=0.25 near position in world (meters)
z=0 (world)
z=-0.25 far position in world (meters)

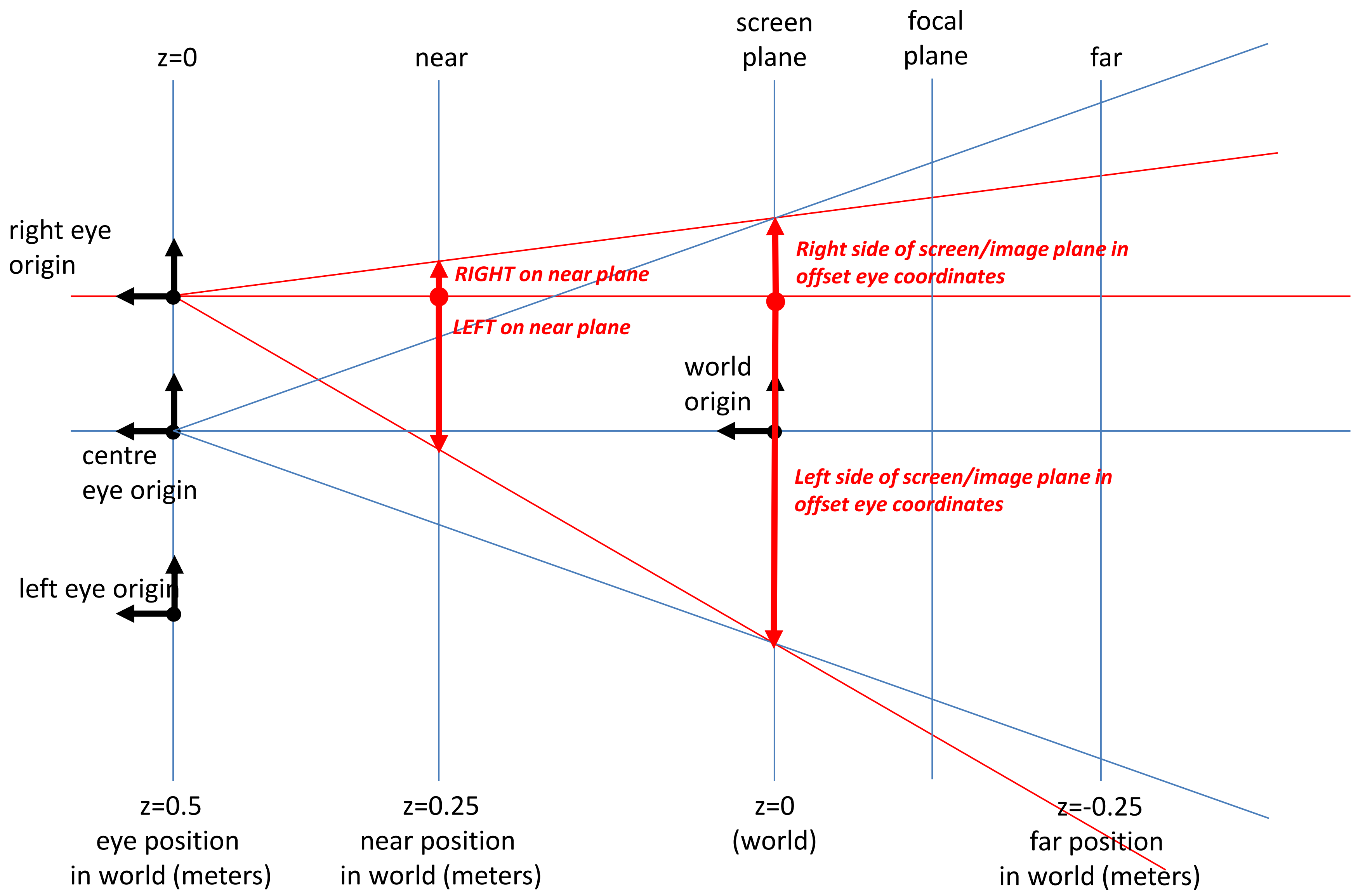
Example depth of field frustum setup…
bottom view! x axis up !


screen plane
focal plane
z=0
near
far
eye with offset origin
world origin
centre eye origin
z=0.5 eye position in world (meters)
z=0.25 near position in world (meters)
z=0 (world)
z=-0.25 far position in world (meters)

Based on slides by Steve Marschner

Example anaglyph frustum setup…

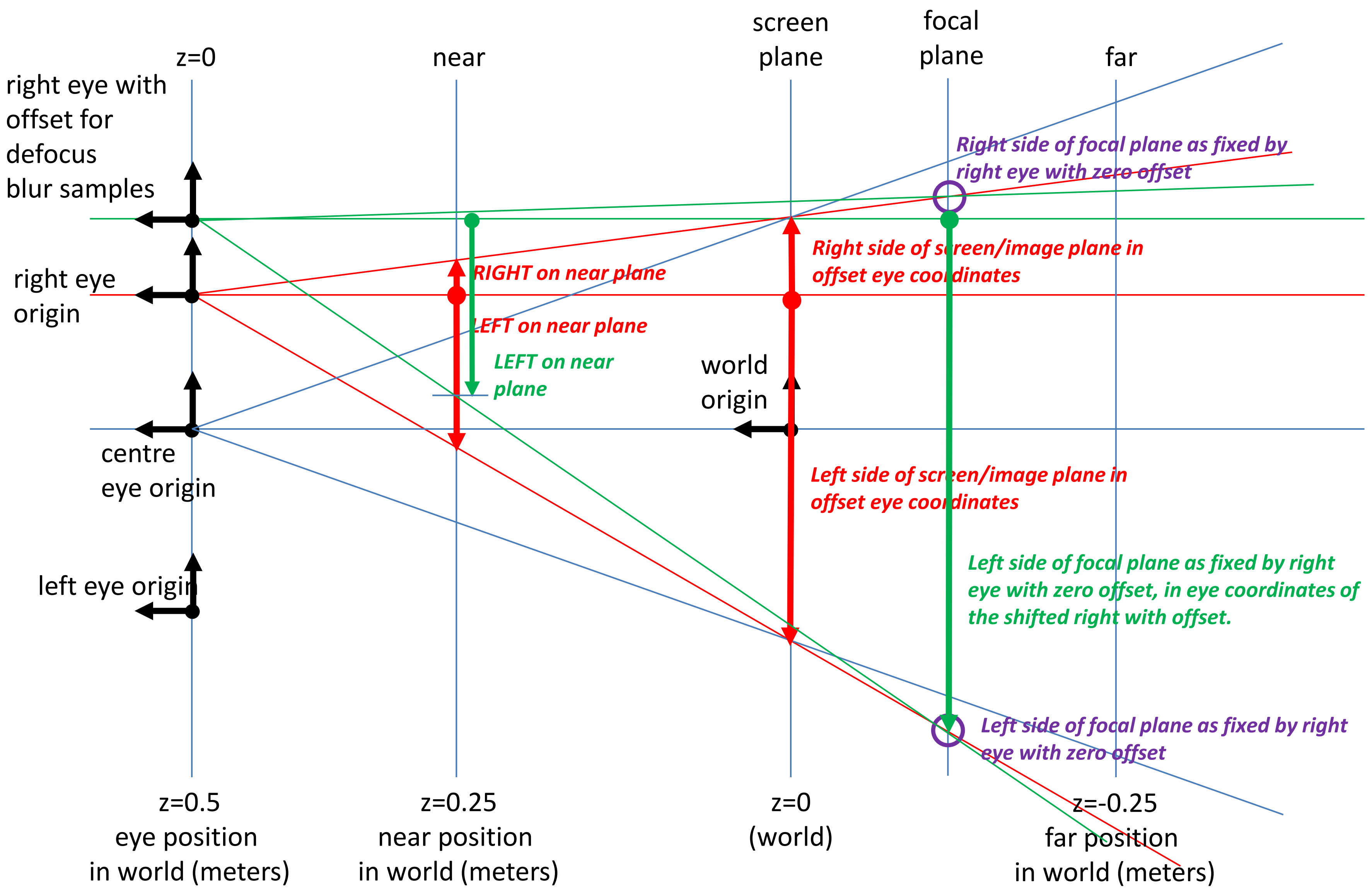
bottom view! x axis up !


z=0
near
screen plane
focal plane
far
right eye origin
RIGHT on near plane
LEFT on near plane
Right side of screen/image plane in offset eye coordinates
world origin
centre eye origin
Left side of screen/image plane in offset eye coordinates
left eye origin
z=0.5 eye position in world (meters)
z=0.25 near position in world (meters)
z=0 (world)
z=-0.25 far position in world (meters)

Based on slides by Steve Marschner

Example depth of field from shifted perspective

bottom view! x axis up !



z=0
near
screen plane
focal plane
far
right eye with offset for defocus blur samples
right eye origin
centre eye origin
left eye origin
world origin
Right side of focal plane as fixed by right eye with zero offset
Right side of screen/image plane in offset eye coordinates
RIGHT on near plane
LEFT on near plane
LEFT on near plane
Left side of screen/image plane in offset eye coordinates
Left side of focal plane as fixed by right eye with zero offset, in eye coordinates of the shifted right with offset.
Left side of focal plane as fixed by right eye with zero offset
z=0.5 eye position in world (meters)
z=0.25 near position in world (meters)
z=0 (world)
z=-0.25 far position in world (meters)

Based on slides by Steve Marschner

Review and more information

- FCG chapter 7
  - Viewing and projection

Review and More Information

- CAPP Section 10.7
  - Windowing Transformations
- CAPP Chapter 13
  - View Specification 13.3
  - Perspective 13.6
  - Orthographic cameras 13.8
- Can also see FCG Chapter 7