3D Viewing, Orthographic and Perspective Projection

COMP557
Paul Kry

Roadmap

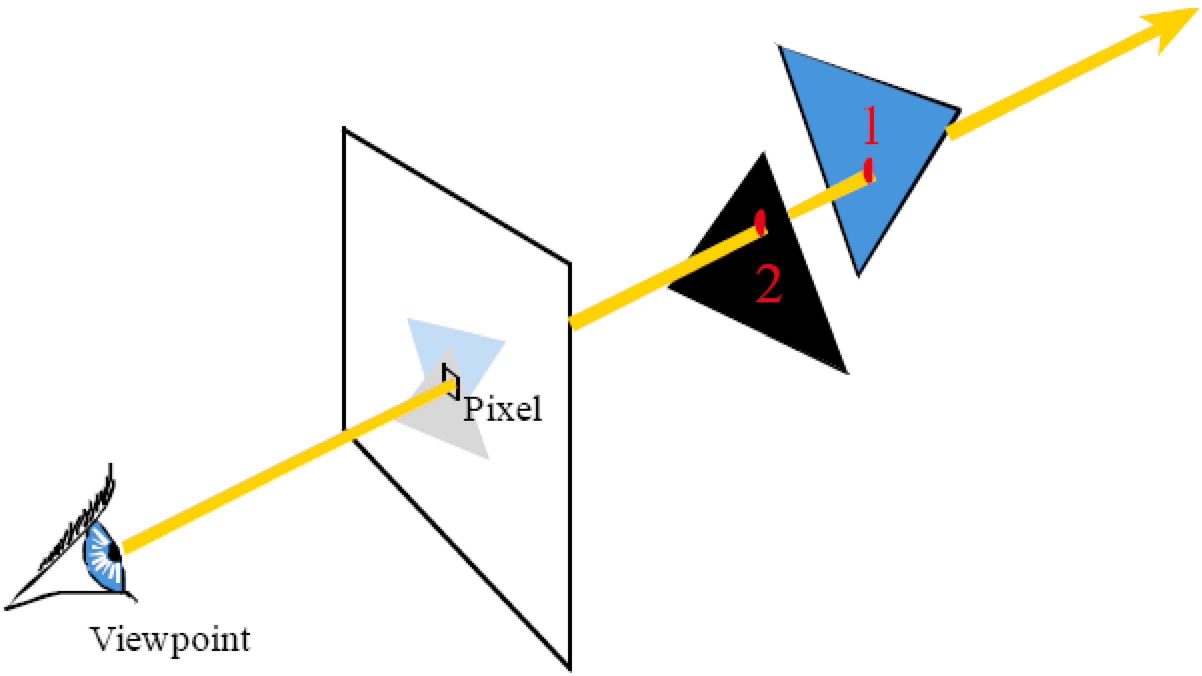
- Transformations (FCA 6) A1
- Scene graphs (FCA 12.2) A2
- Viewing and Projection (FCA 7) A3
- Meshes, simplification (FCA 12.1 + pdf)
- Subdivision (pdf + notes) A4
- …

May insert material on graphics pipeline / rasterization, clipping /culling (FCA 8), light and shadow (FCA 4.5, 10), before meshes.

Recall: ***Image order*** *and* ***Object order***

- Image order: “backward” approach
  - start from pixel
  - 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
  - compute its projection into the image
- matrix transformations critical for object order approach:
  - 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

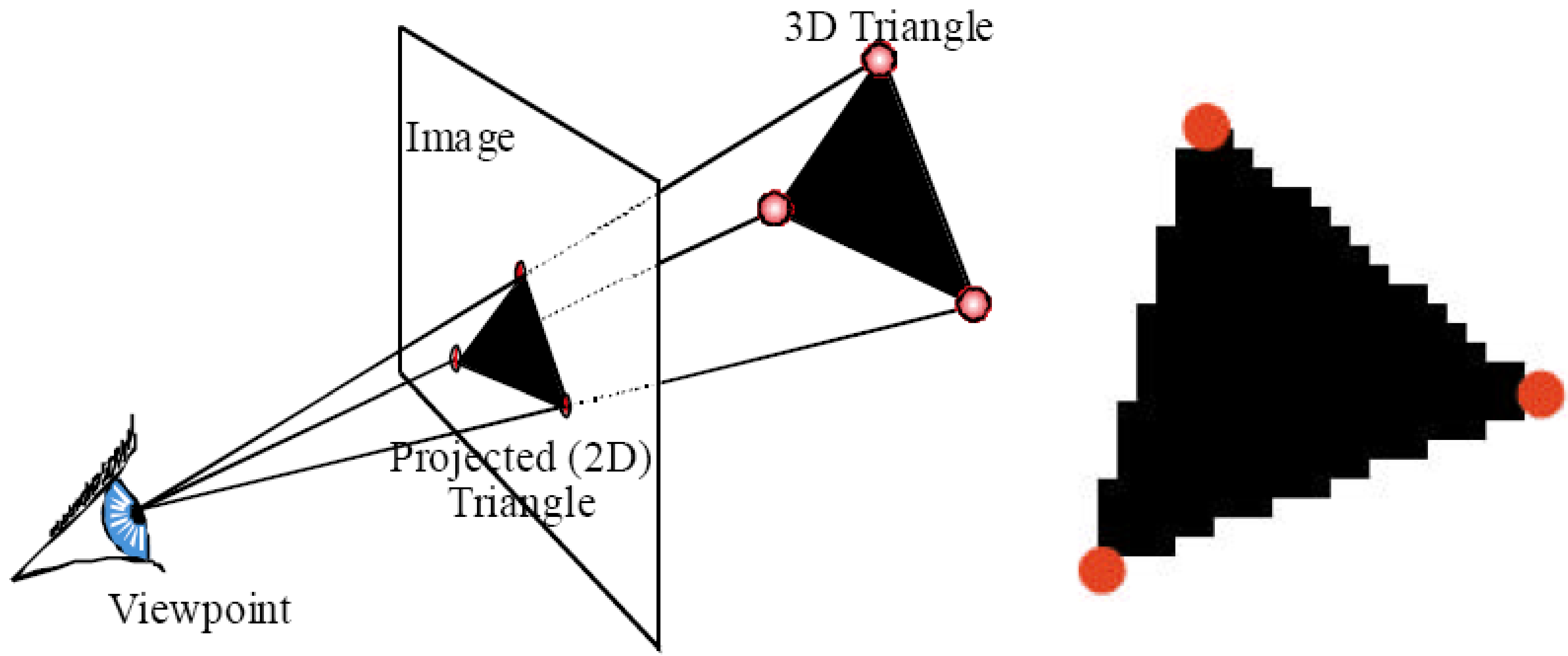


[Durand]

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

Based on slides by Steve Marschner

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)

Based on slides by Steve Marschner

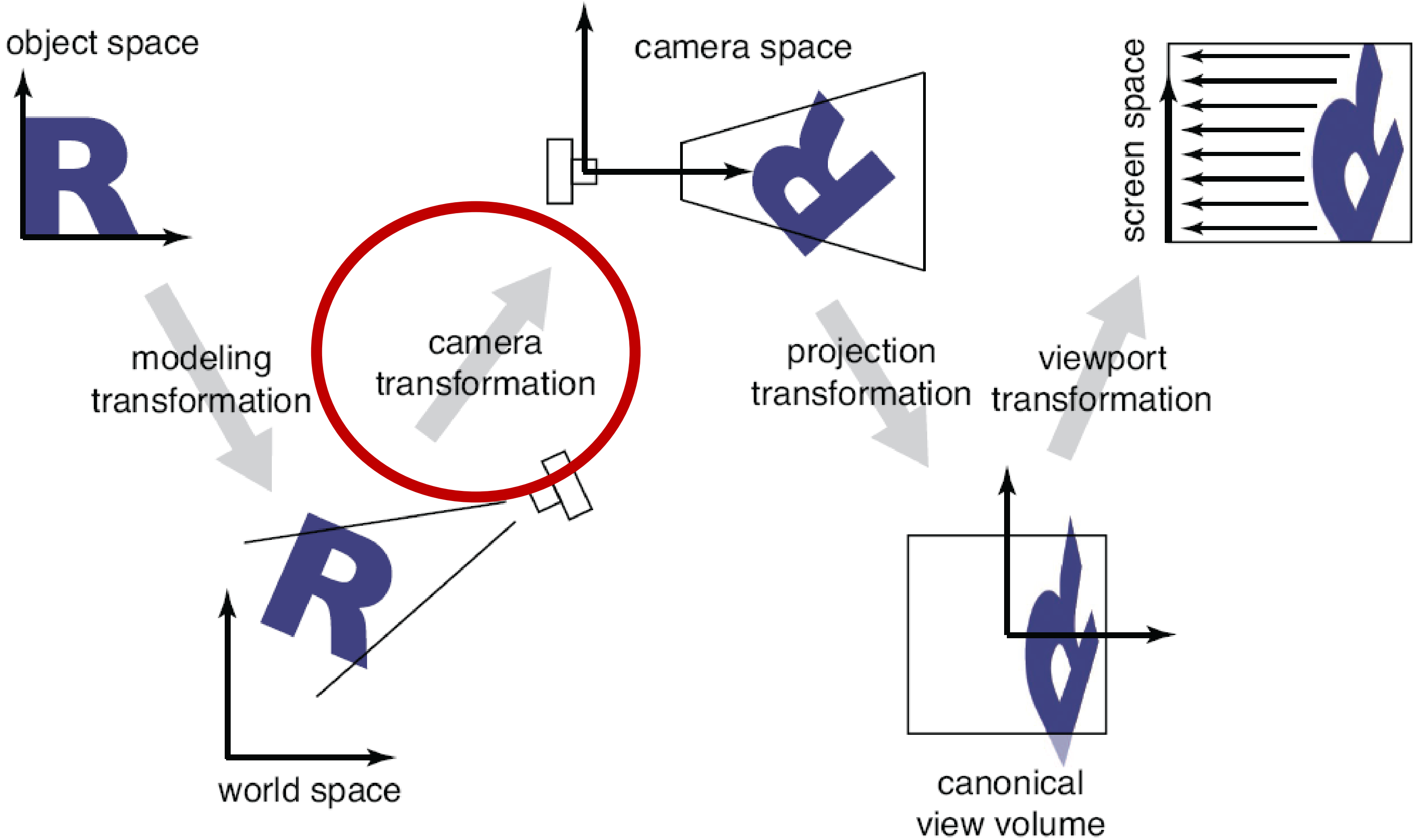
Mathematics of projection

- Always work in eye coords
  - assume eye point at **0** and plane perpendicular to z
- Orthographic case
  - a simple projection: just toss out z
- Perspective case: scale diminishes with z
  - and increases with d

Based on slides by Steve Marschner

Pipeline of transformations

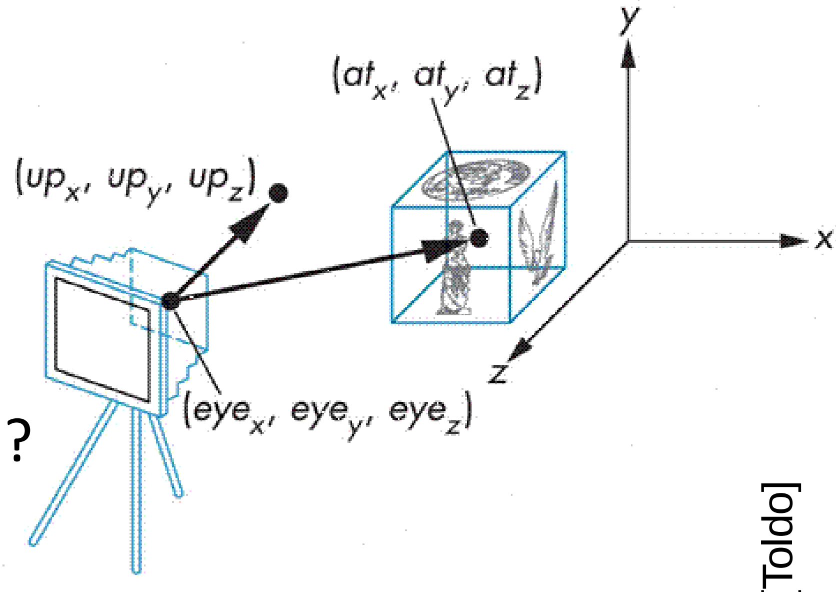
- Standard sequence of transforms



Based on slides by Steve Marschner

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?
  - Can use gluLookat



[Toldo]

Based on slides by Steve Marschner

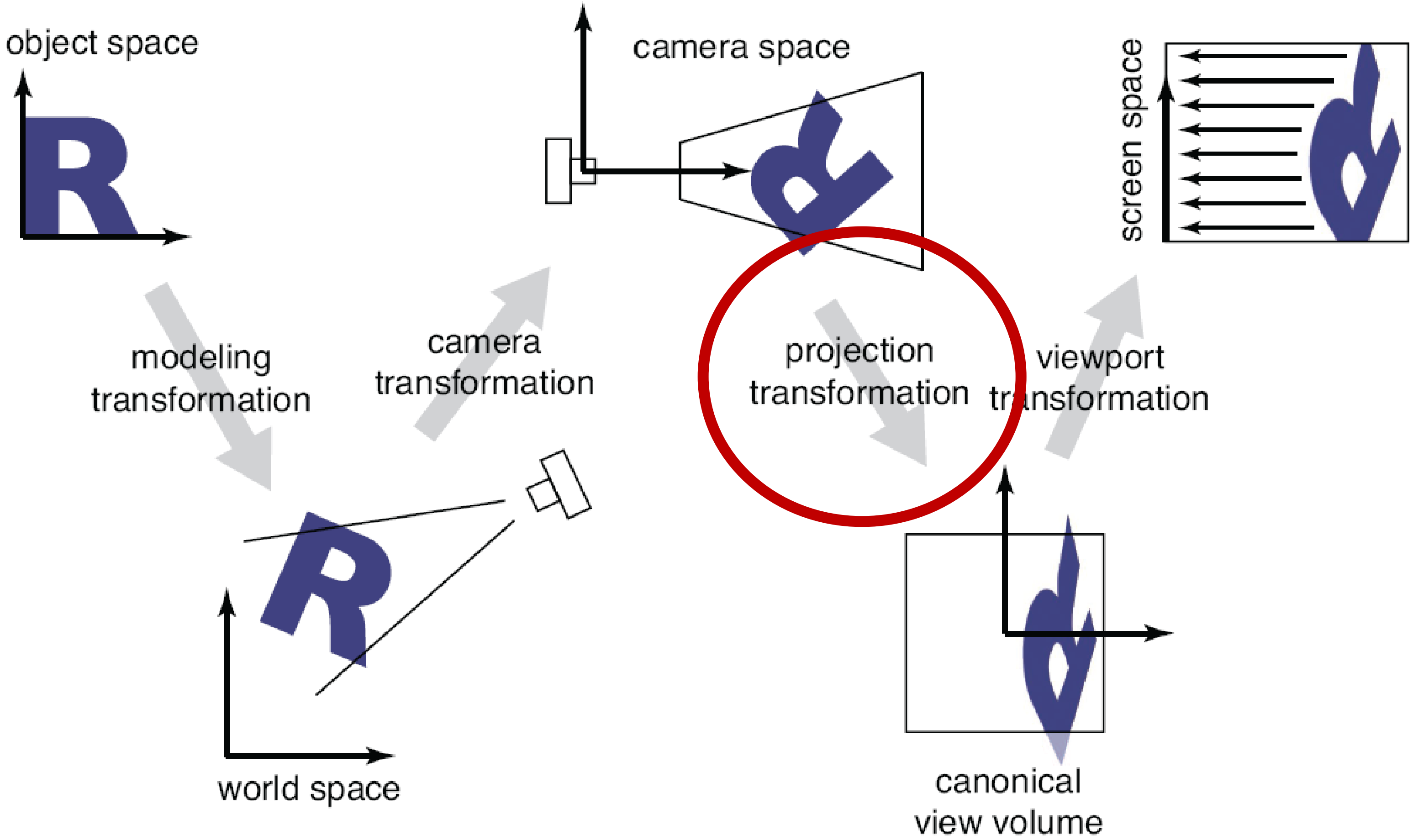
“Lookat” Transformation

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

Based on slides by Steve Marschner

Pipeline of transformations

- Standard sequence of transforms

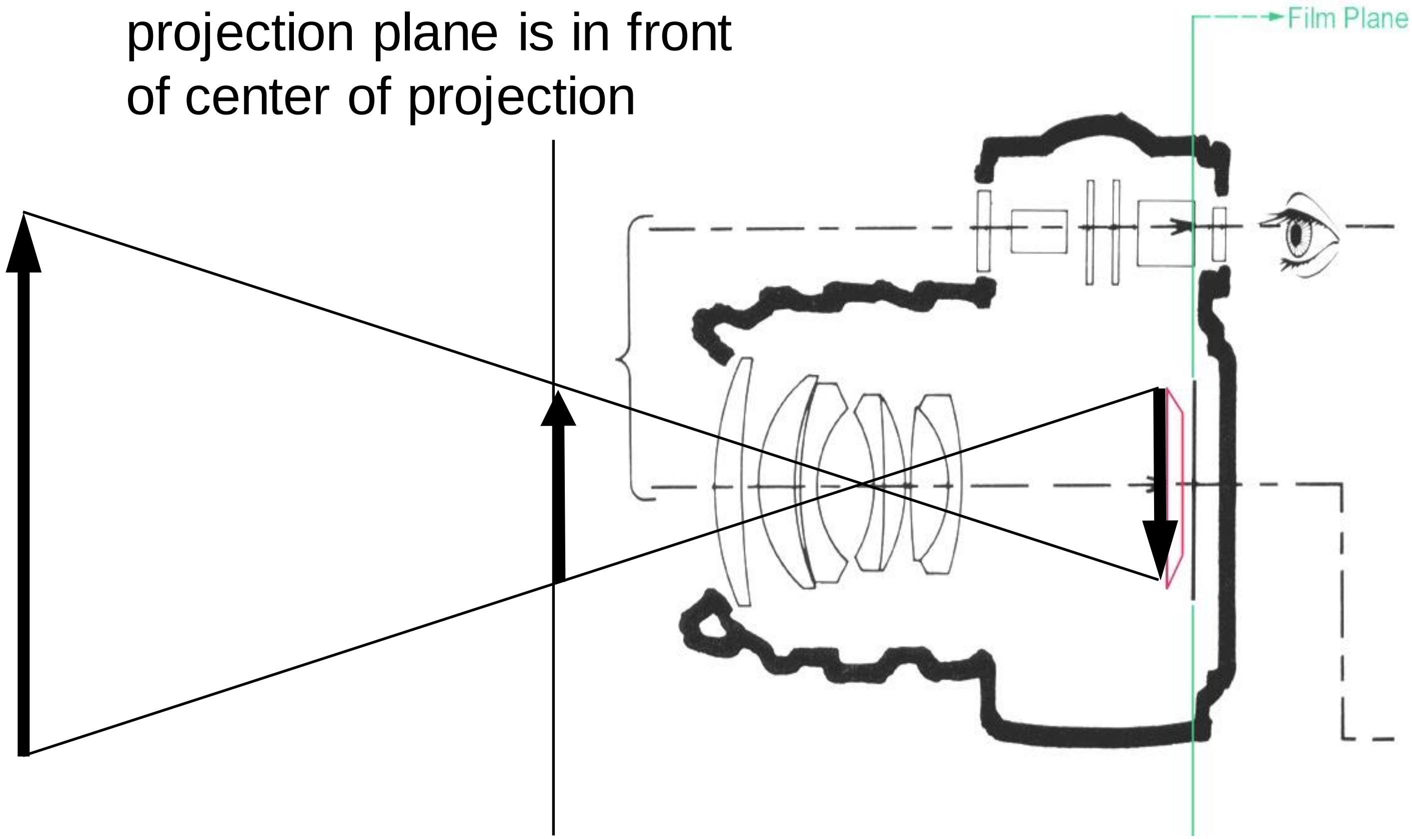


Based on slides by Steve Marschner

Projection

- How to form an image by ***planar perspective projection***?

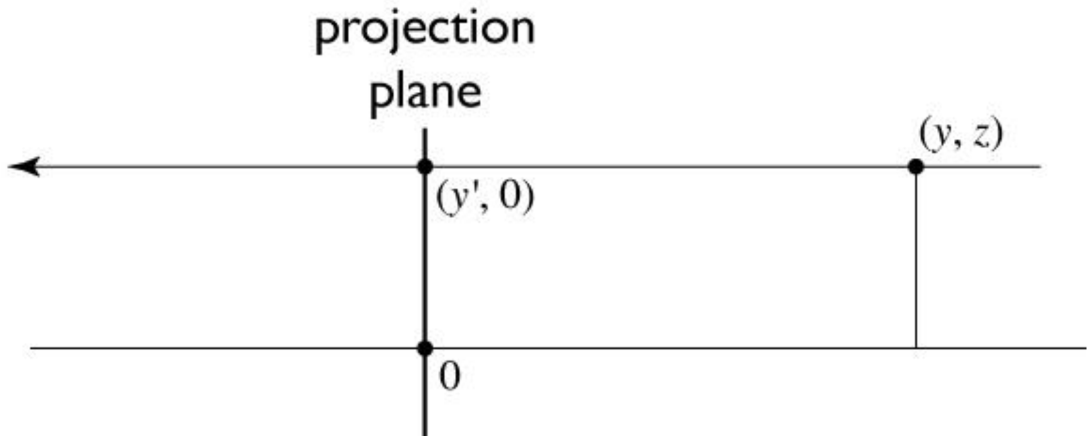
In computer graphics, projection plane is in front of center of projection



[Source unknown]

Based on slides by Steve Marschner

Parallel projection: orthographic



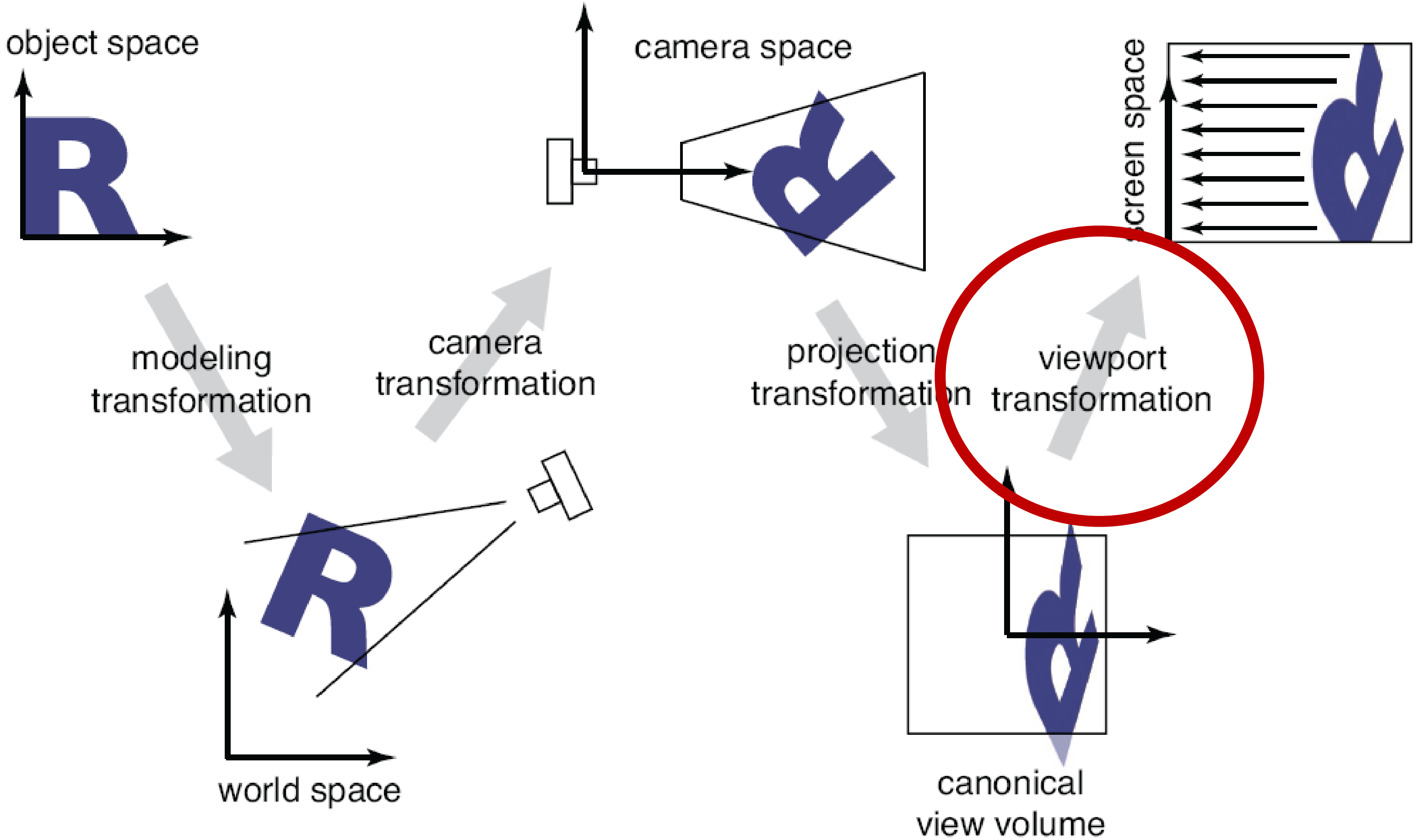
But lets start with something simpler… orthographic projection
to implement orthographic, just toss out *z*:

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

Based on slides by Steve Marschner

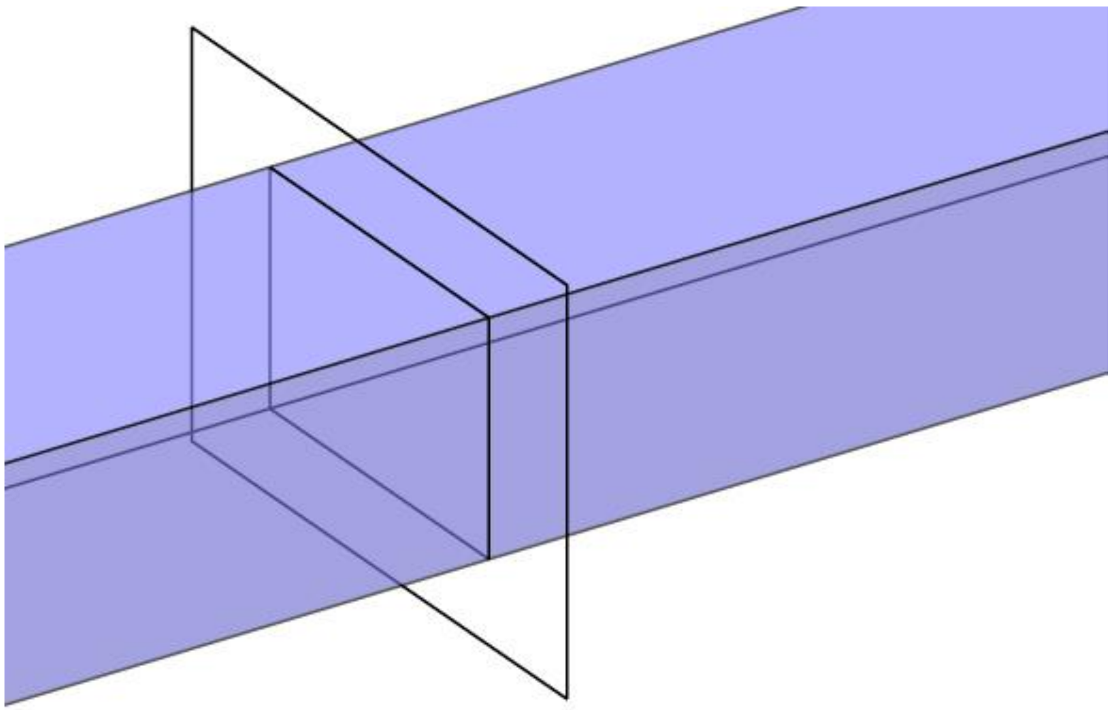
Pipeline of transformations

- Standard sequence of transforms



Based on slides by Steve Marschner

View volume: orthographic



Based on slides by Steve Marschner

Viewing a cube of size 2

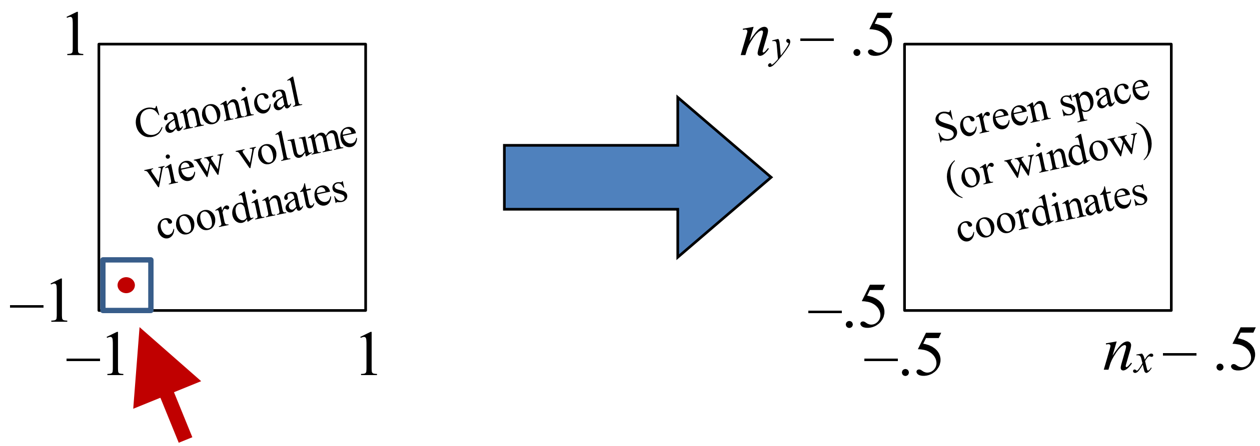
- Start by looking at a restricted case: the *canonical view volume*
- It is the cube $[-1,1]^3$, viewed from the z direction
- Matrix to project it into a square image in $[-1,1]^2$ is trivial:

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

Based on slides by Steve Marschner

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

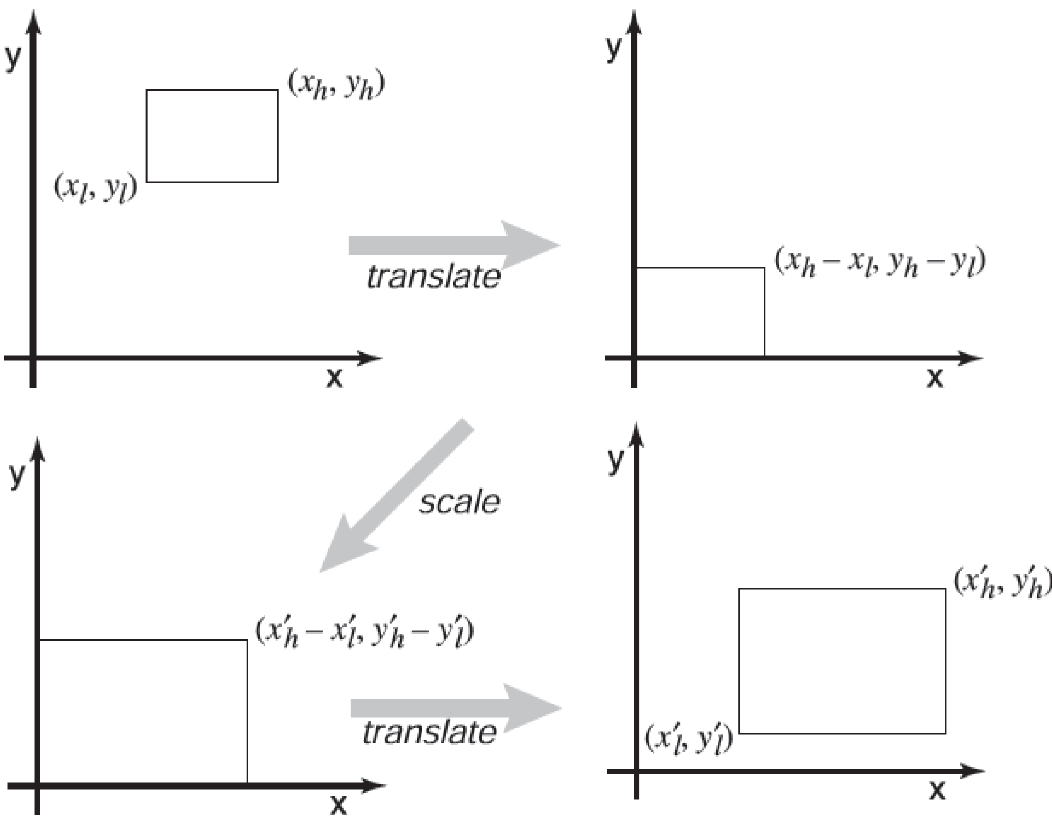


Pixel size in canonical view volume is $2/n_x$ by $2/n_y$
Center is at ½ pixel width and height away from (-1, -1)
i.e., $(-1+1/n_x, -1+1/n_y)$ maps to (0,0) integer pixel location
$(1-1/n_x, 1-1/n_y)$ maps to (n_x-1, n_y-1) integer pixel location

Based on slides by Steve Marschner

Windowing transforms

- This transformation is worth generalizing
  - take one axis-aligned rectangle or box to another
  - a useful transformation chain



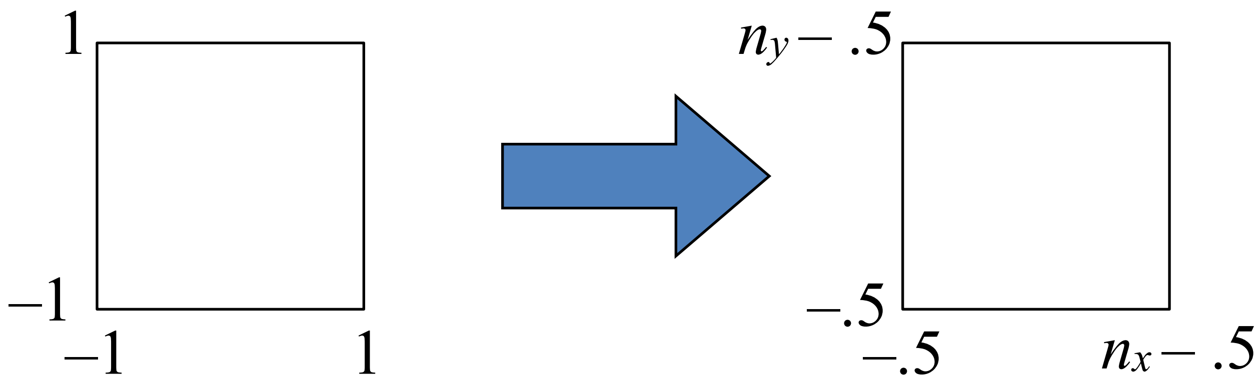
$$\begin{bmatrix} 1 & 0 & x'_l \\ 0 & 1 & y'_l \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & 0 \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_l \\ 0 & 1 & -y_l \\ 0 & 0 & 1 \end{bmatrix}$$

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

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

Based on slides by Steve Marschner

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)

Viewport transformation

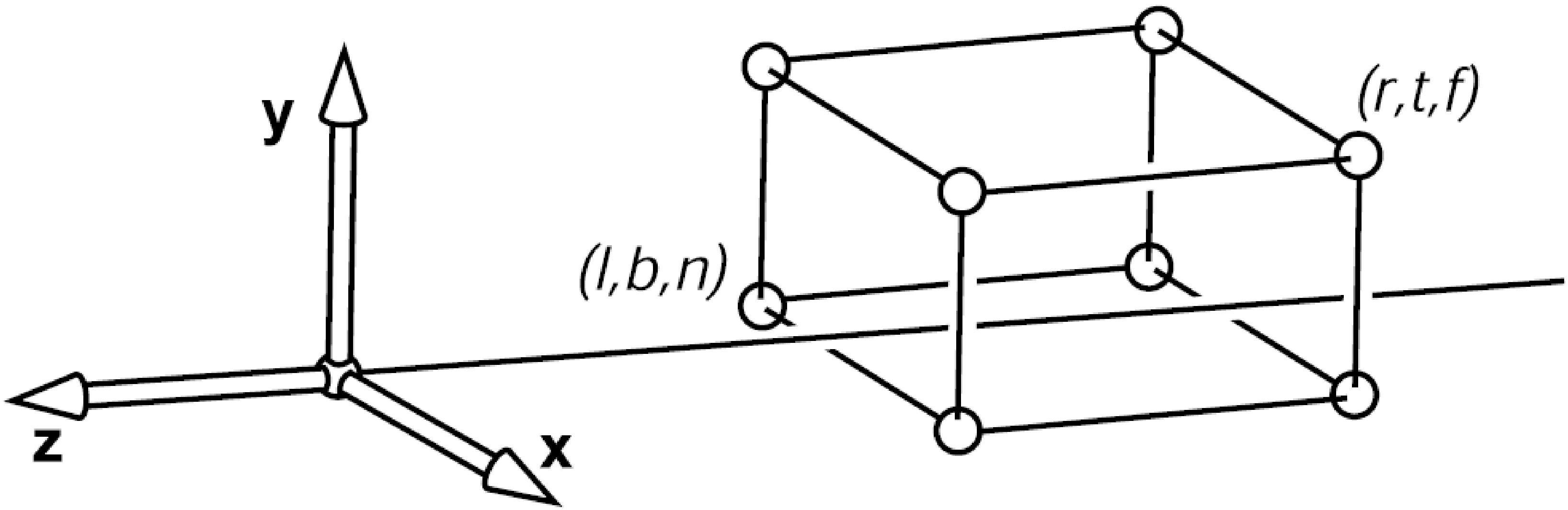
- In 3D, carry along *z* for the ride
  - one extra row and column

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

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

Based on slides by Steve Marschner

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)

Based on slides by Steve Marschner

Orthographic projection

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

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

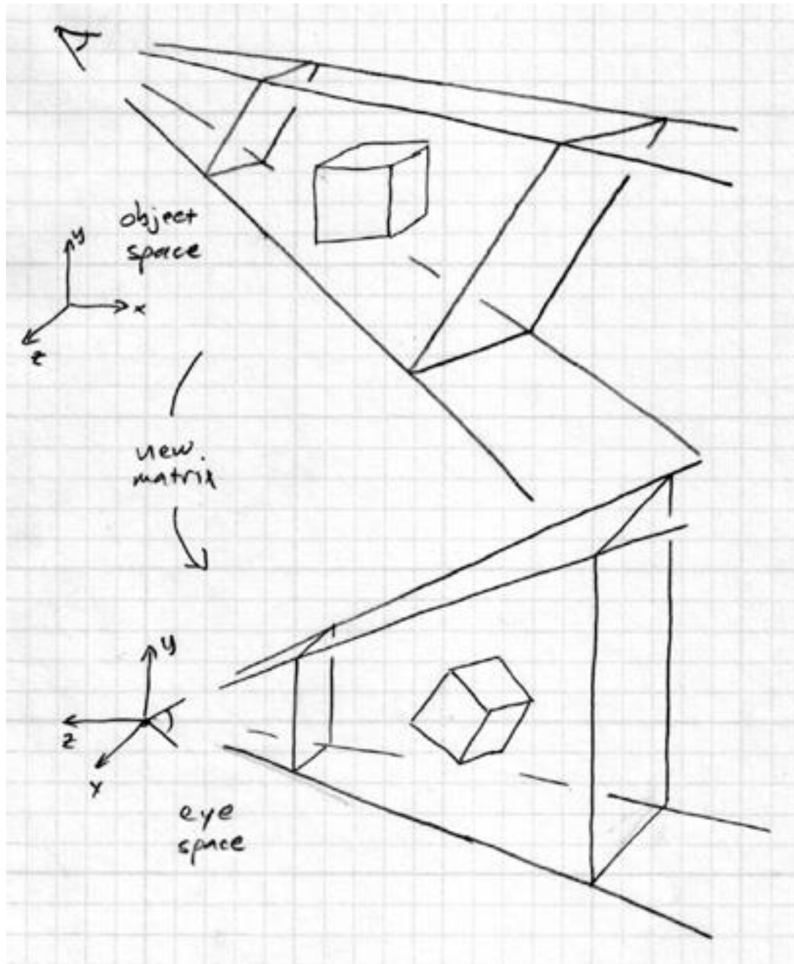
$$\mathbf{M}_{\text{orth}} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Based on slides by Steve Marschner

Camera and modeling matrices

- We worked out all the preceding transforms starting from eye coordinates
  - before we do any of this we need to transform into that space
- Transform from world (canonical) to eye space is traditionally 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

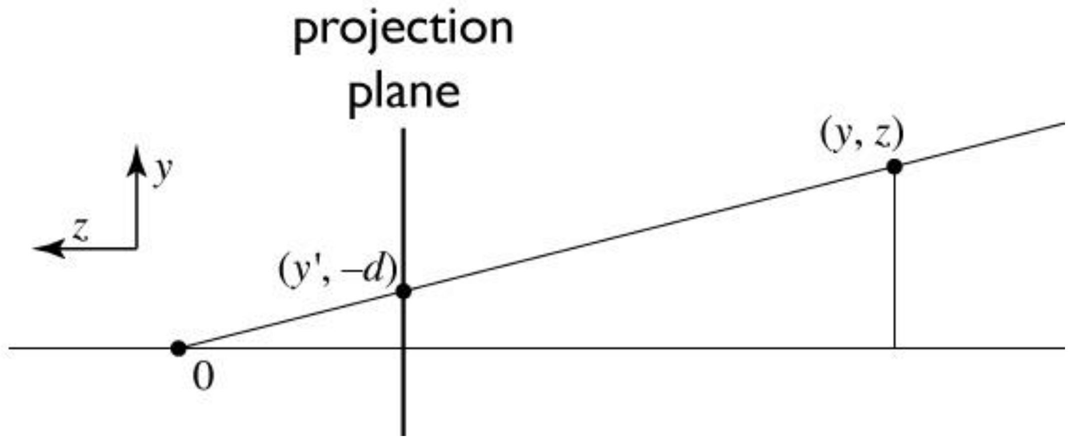
Orthographic transformation chain

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

$$\mathbf{p}_s = \mathbf{M}_{\text{vp}} \mathbf{M}_{\text{orth}} \mathbf{M}_{\text{cam}} \mathbf{M}_{\text{m}} \mathbf{p}_o$$

$$\begin{bmatrix} x_s \\ y_s \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x - 1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y - 1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \mathbf{M}_{\text{m}} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

Planar Perspective projection



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 $-d/z$
- W can think of dividing by negative z so that y’ has the same sign as y!

Based on slides by Steve Marschner

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

Based on slides by Steve Marschner

Homogeneous coordinates revisited

- Introduced *w* = 1 coordinate as a placeholder

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

  – used as a convenience for unifying translation with linear

- Can also allow arbitrary *w*

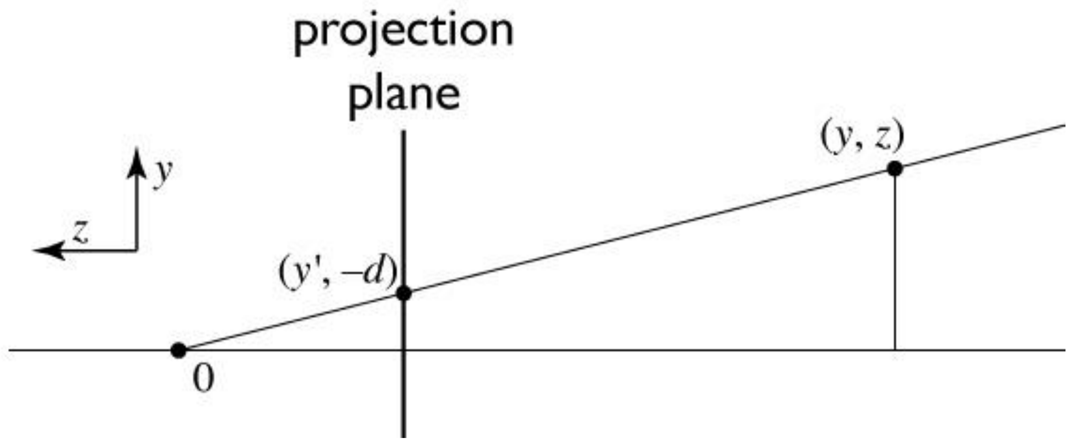
$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \sim \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix}$$

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

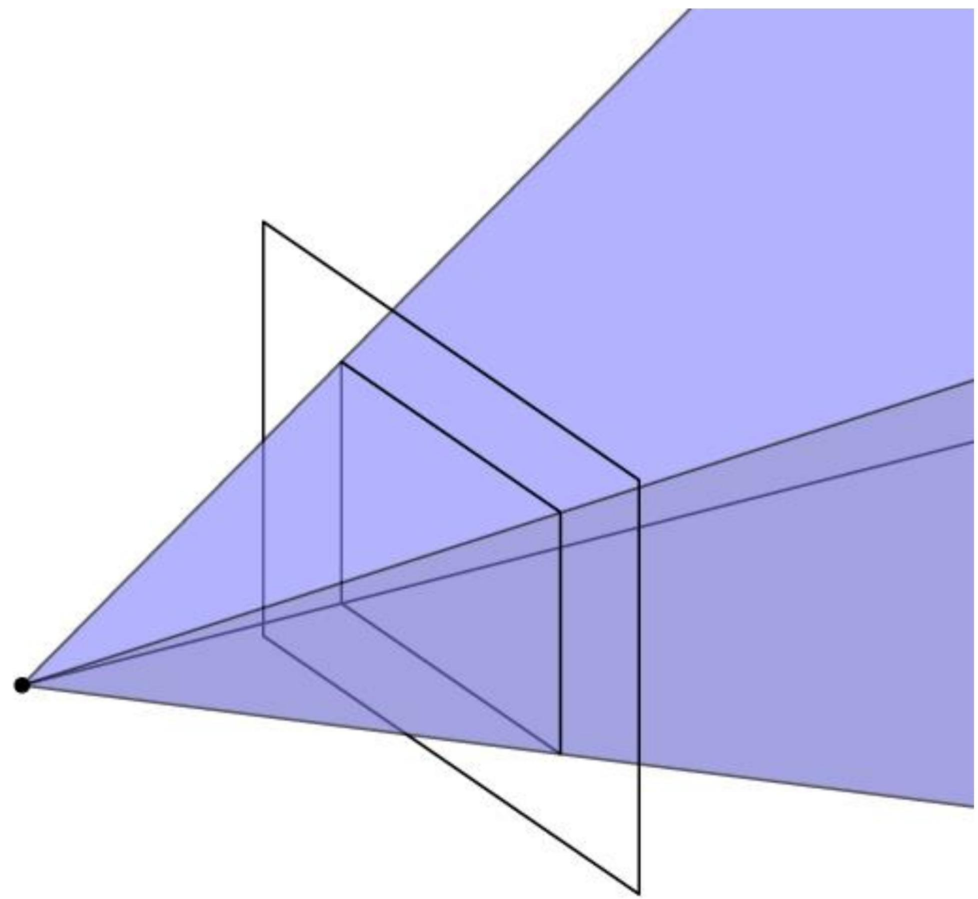


to implement perspective, just move z to w:

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

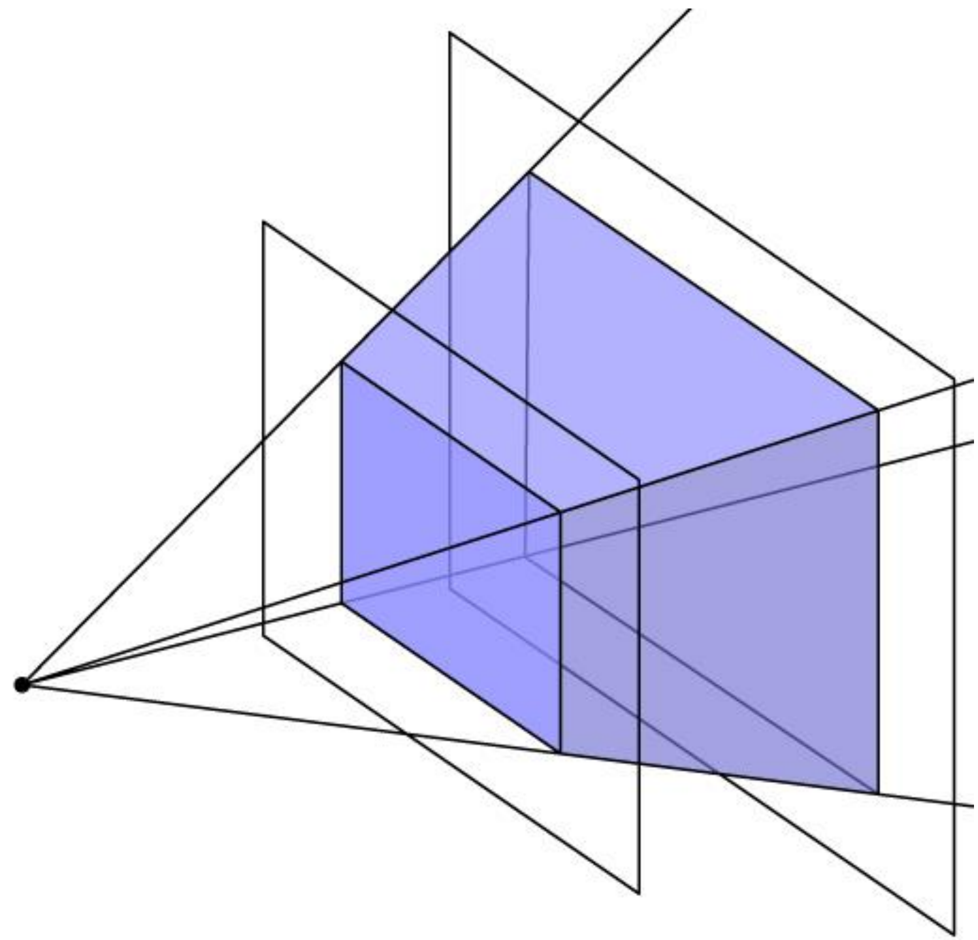
Based on slides by Steve Marschner

View volume: perspective



Based on slides by Steve Marschner

View volume: perspective (clipped)



Based on slides by Steve Marschner

Carrying depth through perspective

- Perspective has a varying denominator—can't preserve depth!
- Compromise: preserve order, and depth on near and far planes

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

  – that is, choose a and b so that $z'(n) = n$ and $z'(f) = f$.

$$\tilde{z}(z) = az + b$$

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

$$\text{want } z'(n) = n \text{ and } z'(f) = f$$

$$\text{result: } a = -(n + f) \text{ and } b = nf \text{ (try it)}$$

Based on slides by Steve Marschner

Official perspective matrix

- Use near plane distance as the projection distance
- Let n be ***negative*** and $d = -n$
- Scale by –1 to have fewer minus signs
  - The matrix is different but this does not change the projective transformation

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

Potential Confusion:
In OpenGL the near and far plane are specified as positive numbers giving the distance along the negative z axis!

Questions

- Questions to better understand the action of a 4x4 homogenous planar perspective transformation matrix
  - What happens to lines through the origin ?
    - They become parallel
  - What happens to vectors (x,y,z,0) ?
    - They map to plane z=near+far
  - 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 points at z = (near+far)/2 ?
    - (n*n+f*f) / (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

Questions (the easier version)

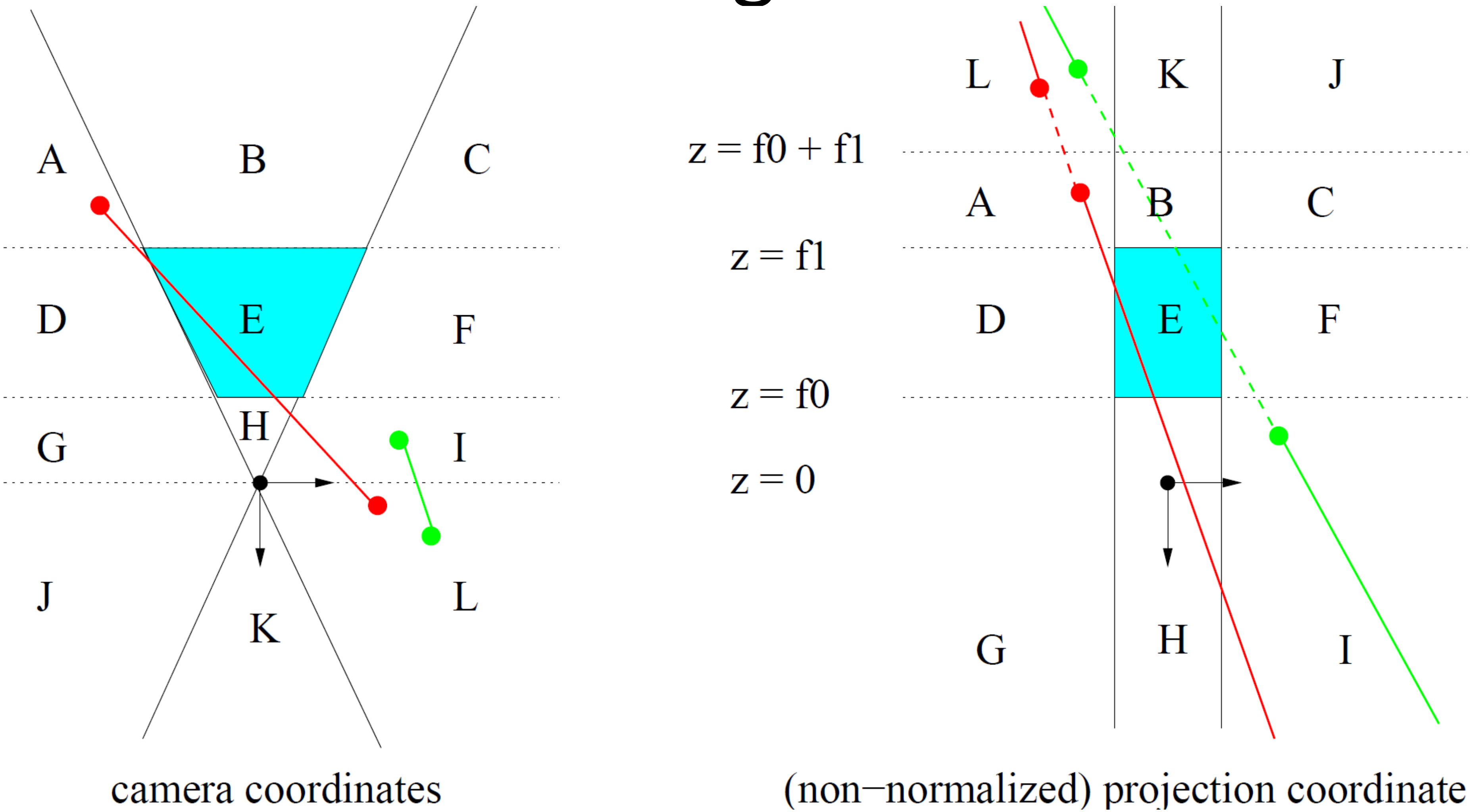
- Given the following projection matrix, with near at 1 and far at 3, answer the following by interpreting the result in non-homogeneous coordinates

$$P = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -4 & -3 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

  – What happens to points on the near plane?
  – What happens to points on the far plane?
  – What happens to points half way between?
  – What happens to points behind the camera?

Based on slides by Steve Marschner

Aside: What goes where?



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

Based on slides by Steve Marschner

Perspective projection matrix

- Need perspective projection to produce points in the canonical view volume, so combine with an orthographic projection matrix (windowing transform)

$$\mathbf{M}_{\text{per}} = \mathbf{M}_{\text{orth}}\mathbf{P}$$

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

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

Is n-f a bug?
Is it not f-n ??
The convention is that -near maps to -1 and -far maps to 1.

Based on slides by Steve Marschner

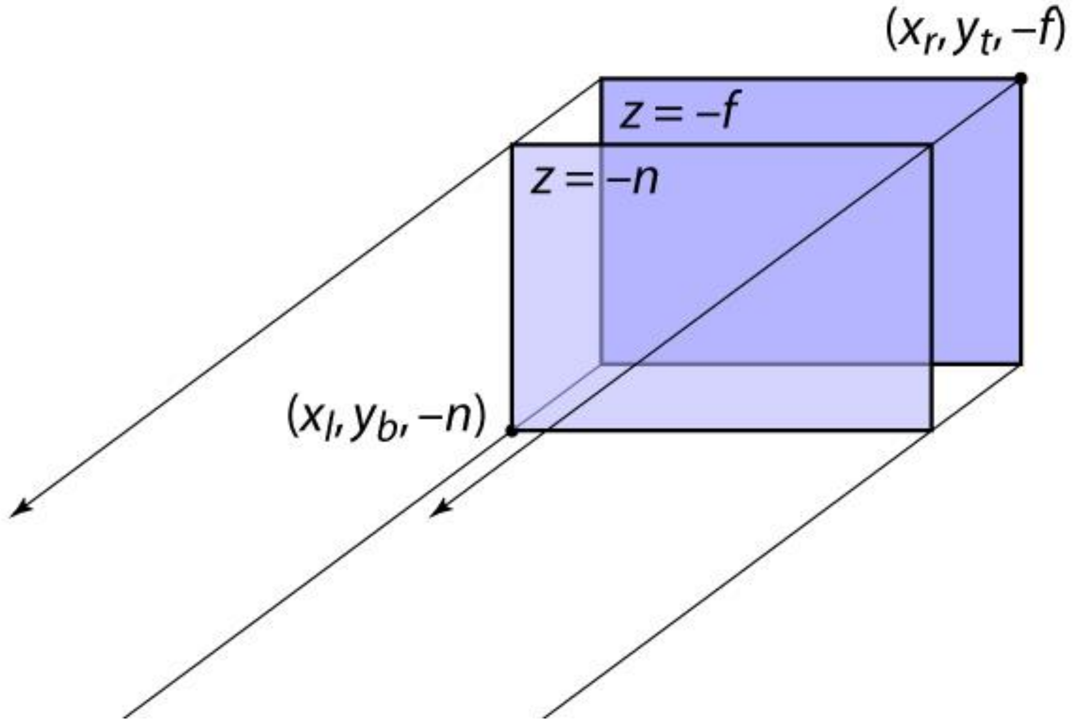
Perspective transformation chain

- Transform into world coords (modeling transform, M_m)
- Transform into eye coords (camera xf., $M_{\text{cam}} = F_c^{-1}$)
- Perspective matrix, P
- Orthographic projection, M_{orth}
- Viewport transform, M_{vp}

$$\mathbf{p}_s = \mathbf{M}_{\text{vp}}\mathbf{M}_{\text{orth}}\mathbf{P}\mathbf{M}_{\text{cam}}\mathbf{M}_{\text{m}}\mathbf{p}_o$$

$$\begin{bmatrix} x_s \\ y_s \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y-1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix} \mathbf{M}_{\text{cam}}\mathbf{M}_{\text{m}} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

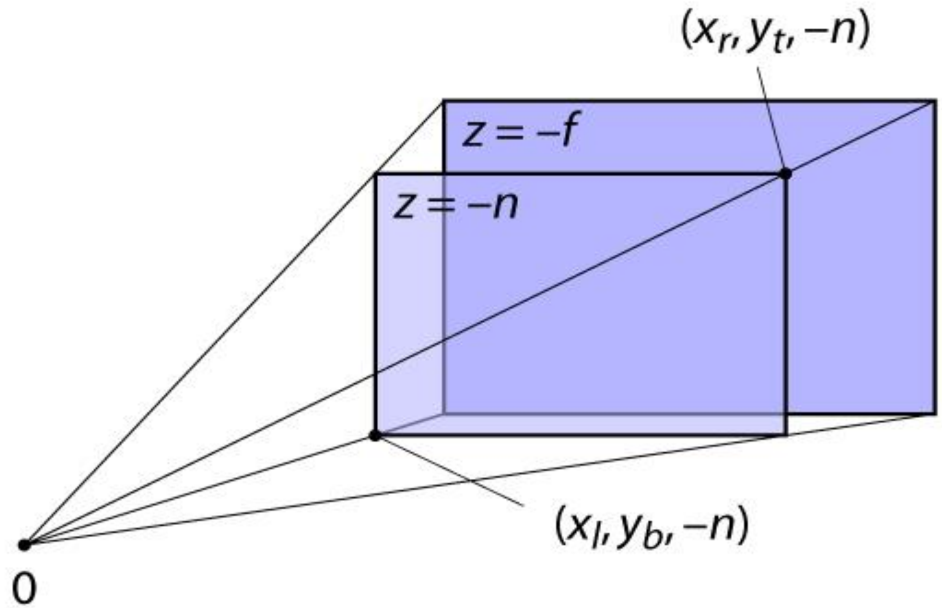
OpenGL view frustum: orthographic



Note OpenGL puts the near and far planes at –*n* and –*f* so the user should give positive numbers

Based on slides by Steve Marschner

OpenGL view frustum: perspective



Note OpenGL puts the near and far planes at $-n$ and $-f$ so that the user should give positive numbers

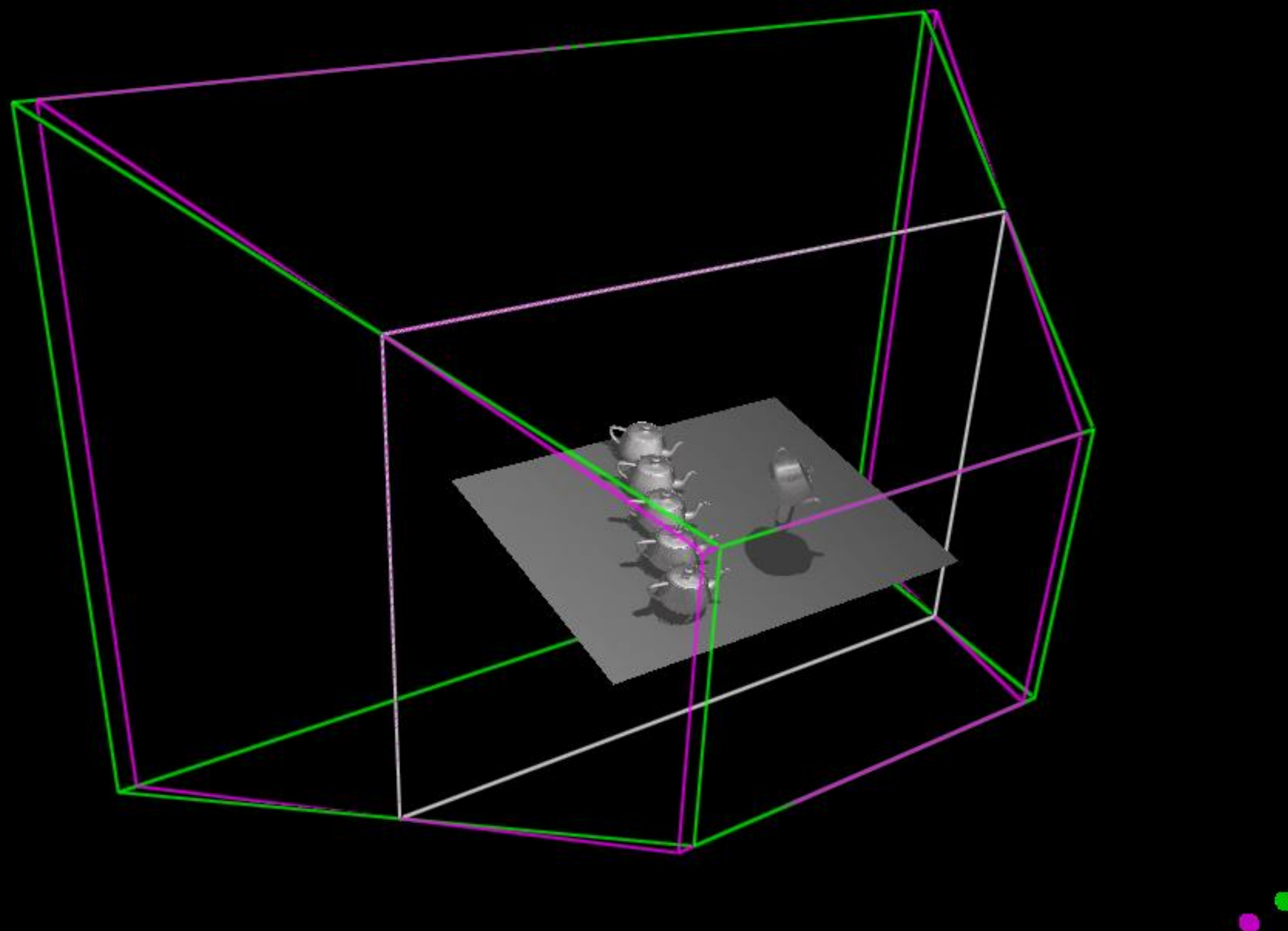
gluPerspective(fovy, aspect, near, far)

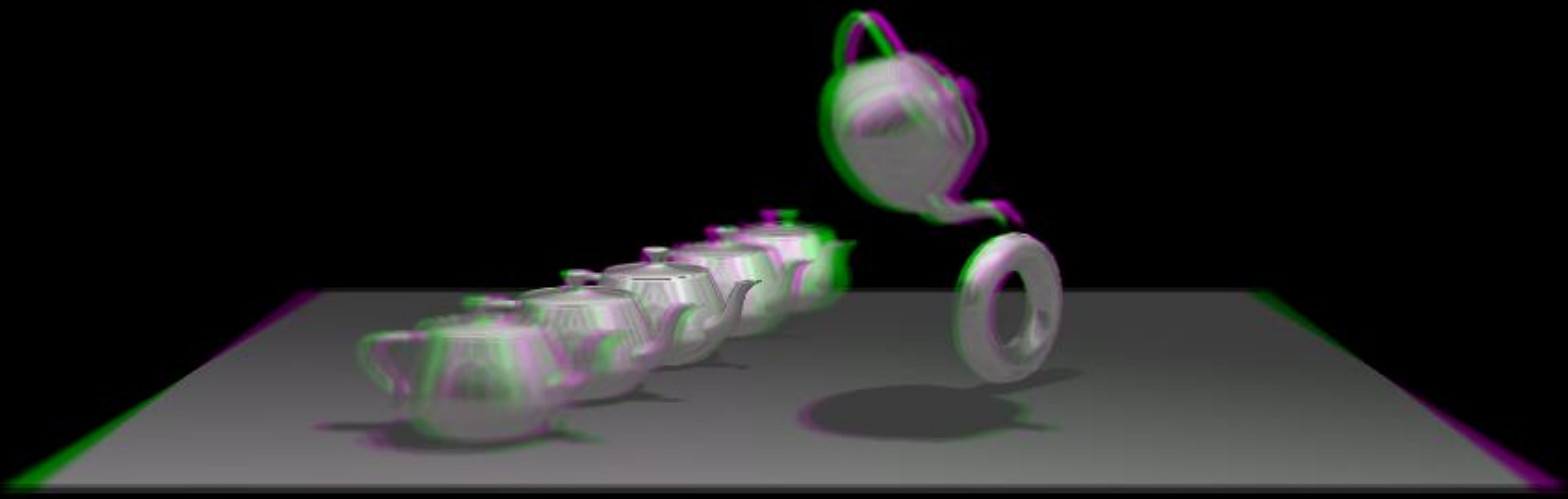
glFrustum($\underbrace{\text{left, right, bottom, top}}_{\text{Defined on near plane}}$, near, far)

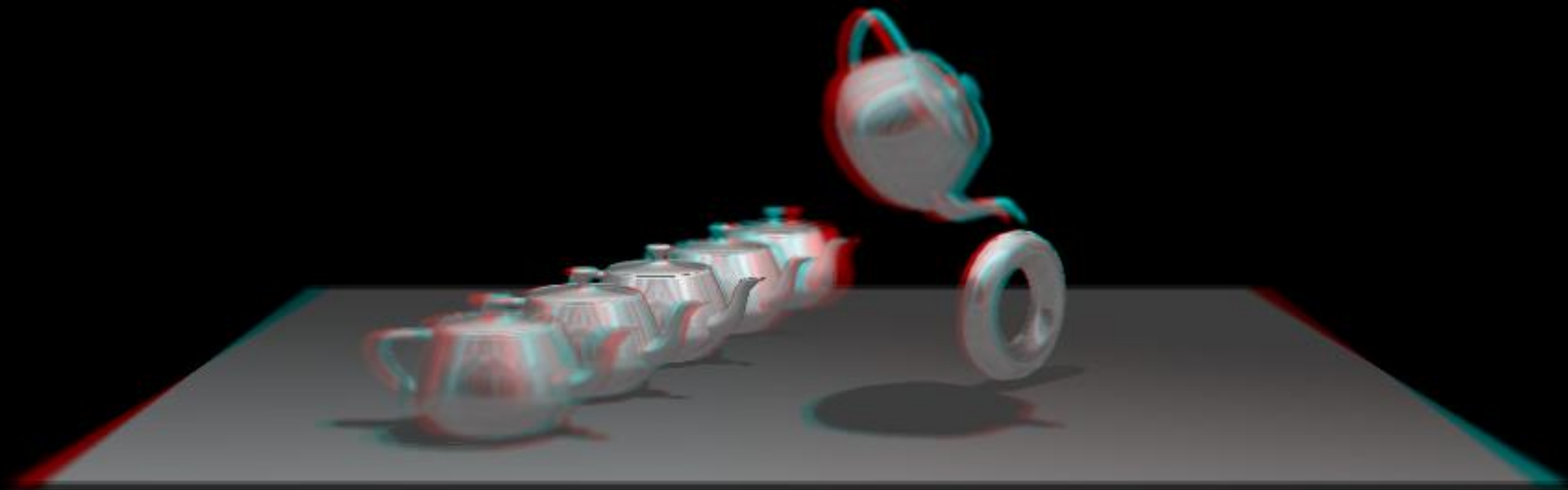
Frustum applications

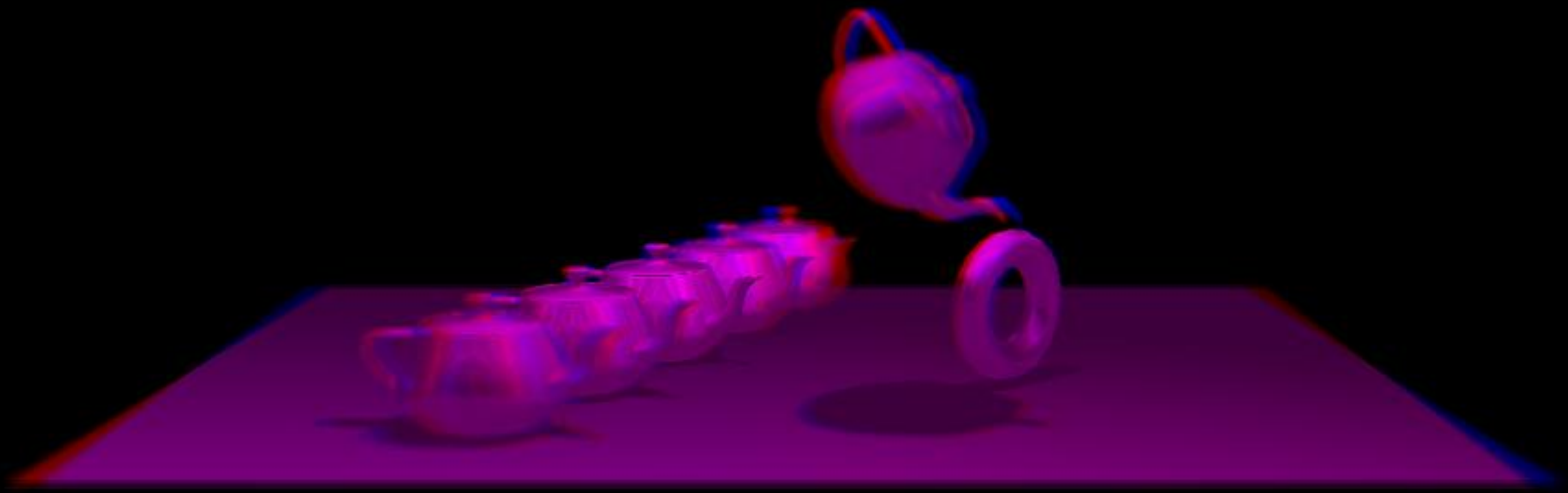
- Shifted perspective
- Tiled rendering (e.g., render very high resolutions)
- 3D viewing (i.e., left eye right eye)
- Depth of field (i.e., accumulating multiple render passes)

Based on slides by Steve Marschner









Review and more information

- Textbook chapter 7
  - Viewing and projection