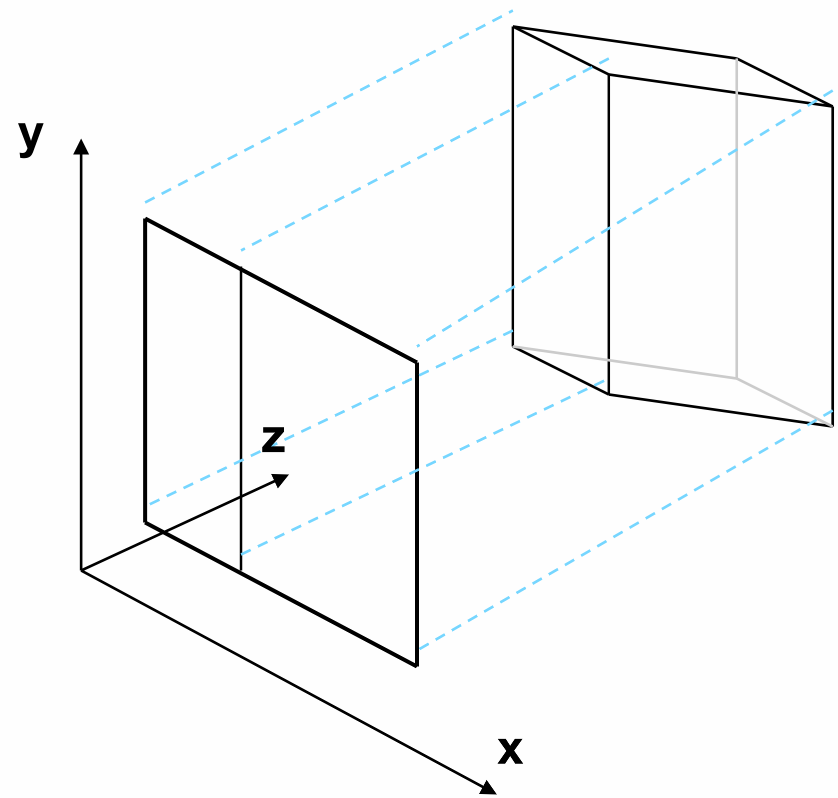
GPU Programming for Video Games

3D to 2D Projection

Prof. Aaron Lanterman
(Based on slides by Prof. Hsien-Hsin Sean Lee)
School of Electrical and Computer Engineering
Georgia Institute of Technology

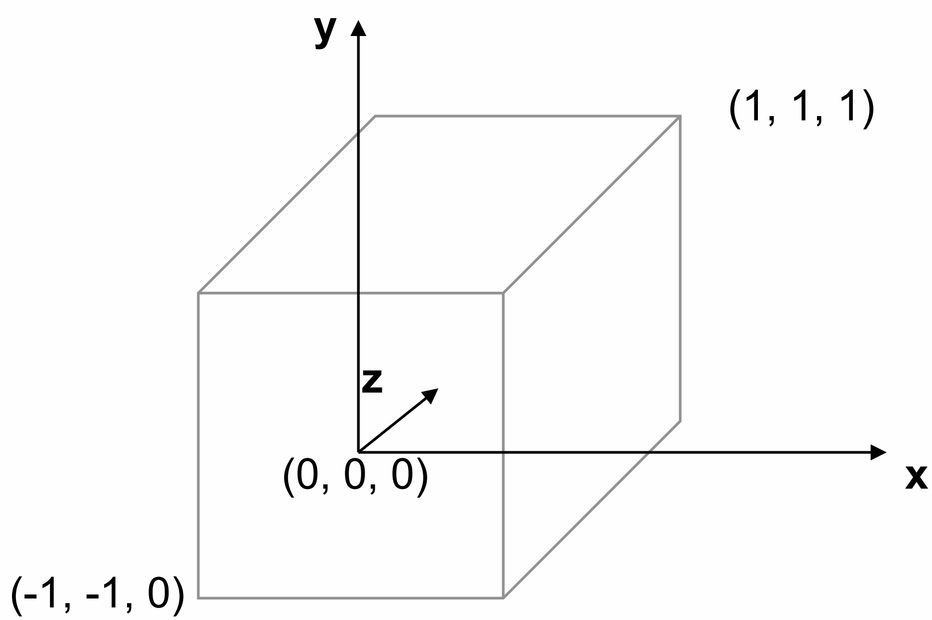
Projection from 3D space



Much discussion adapted from Joe Farrell's article:
http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123__1/)

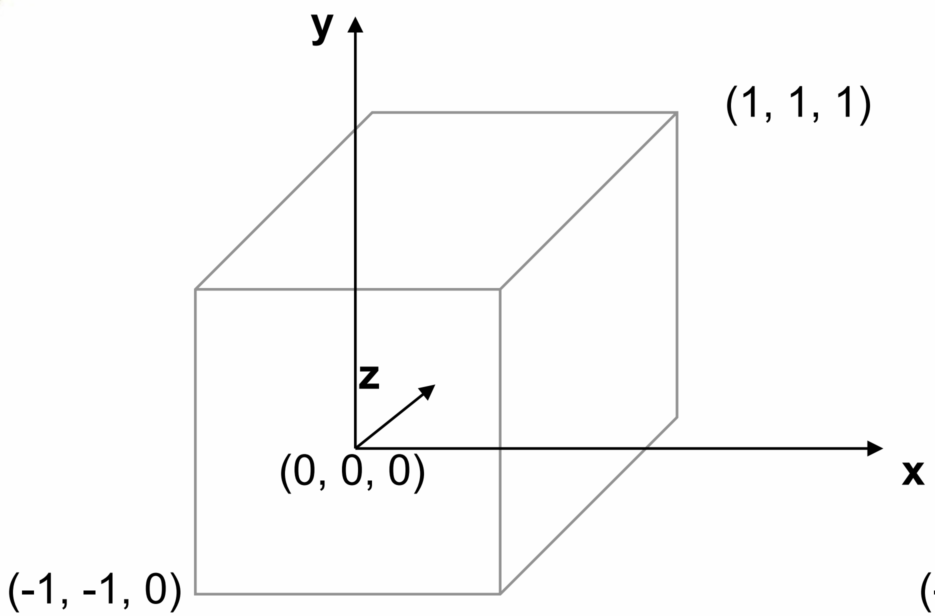
Canonical view volume

- Projection transforms your geometry into a canonical view volume in *normalized device coordinates ("clip space")*
- Only X- and Y-coordinates will be mapped onto the screen
- Z will be almost useless, but used for depth test



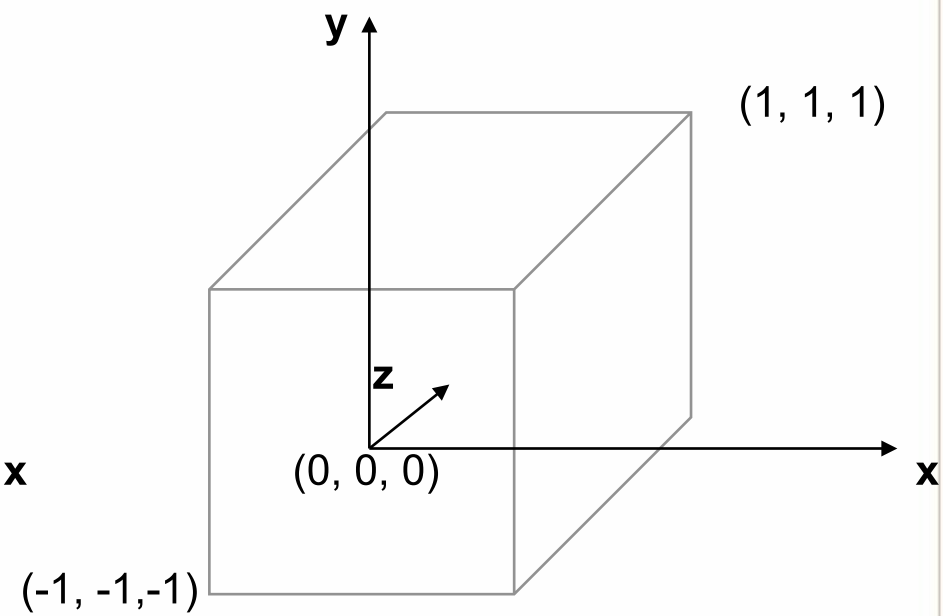
Canonical view volume (LHS)
(-1, -1, 0) to (1,1,1) used by Direct3D

Strange "conventions"



Canonical "clips space" view volume (LHS)
(-1, -1, 0) to (1,1,1) used by
D3D/XNA

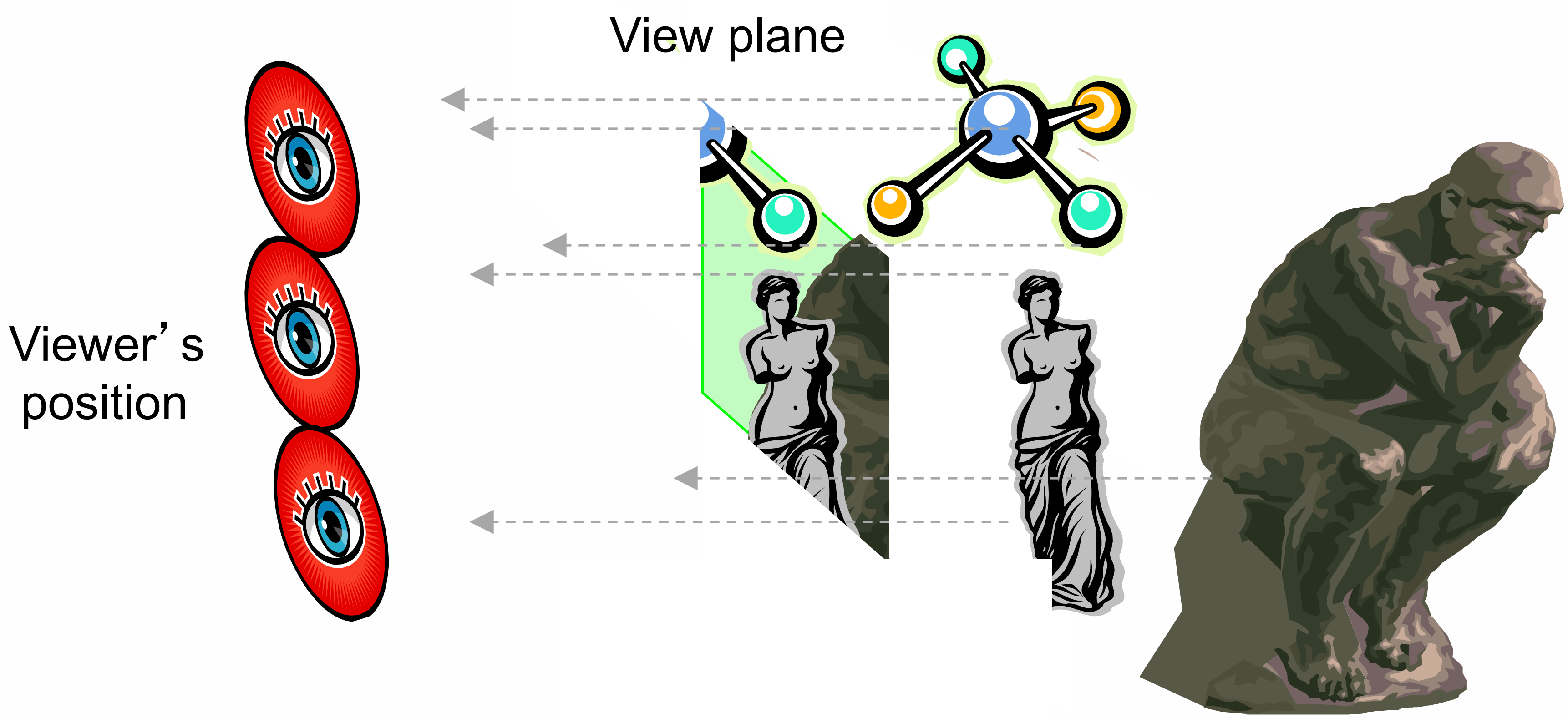
(remember unnormalized
eye-space coordinates in
Direct3D are in a LHS,
but in XNA are in a RHS!!!)



Canonical "clip space" view volume (LHS)
(-1, -1, 1) to (1,1,1) used by
OpenGL/Unity

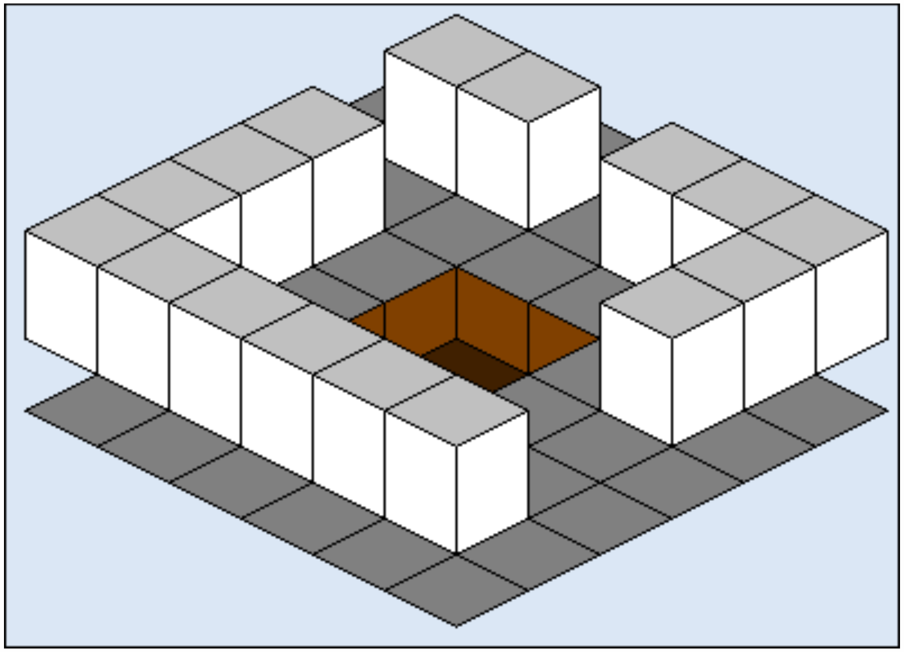
(remember unnormalized
eye-space coordinates in
OpenGL are in a RHS,
but in Unity are in a LHS!)

Orthographic (or parallel) projection



- Project from 3D space to the viewer's 2D space

Style of orthographic projection

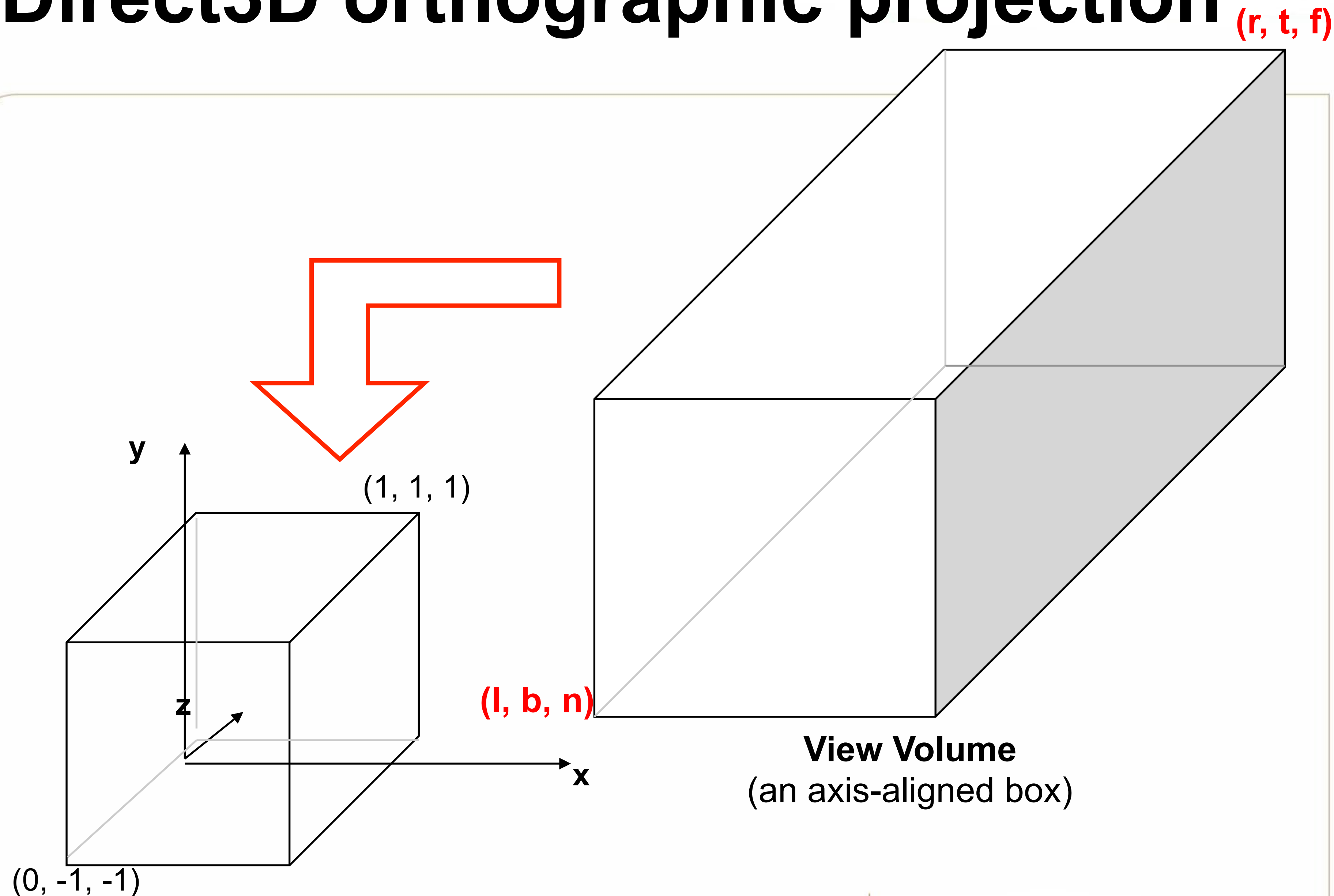


- Same size in 2D and 3D
- No sense of distance
- Parallel lines remain parallel
- Good for tile-based games where camera is in fixed location (e.g., Mahjong or 3D Tetris)

Georgia Institute of Technology

See http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123__2/Deriving-Projection-Matrices.htm

Direct3D orthographic projection



Canonical view volume (Direct3D)

View Volume
(an axis-aligned box)

General orthographic math

- Derive x’ and y’

$$x \in [l, r] \quad x' \in [-1, 1]$$

$$l \le x \le r$$

$$0 \le x - l \le r - l$$

$$0 \le \frac{x-l}{r-l} \le 1$$

$$0 \le \frac{2(x-l)}{r-l} \le 2$$

$$-1 \le \frac{2(x-l)}{r-l} - 1 \le 1$$

$$-1 \le \frac{2x - 2l - r + l}{r-l} \le 1$$

$$-1 \le \frac{2x}{r-l} - \frac{r+l}{r-l} \le 1$$

$$\boxed{\therefore \quad x' = \frac{2x}{r-l} - \frac{r+l}{r-l}}$$

See http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123__2/Deriving-Projection-Matrices.htm

Georgia Institute of Technology

D3D orthographic math for Z (LHS default)

- Derive z'

$$z \in [n, f] \qquad z' \in [0, 1]$$

$$n \le z \le f$$

$$0 \le z - n \le f - n$$

$$0 \le \frac{z - n}{f - n} \le 1$$

$$0 \le \frac{z}{f - n} - \frac{n}{f - n} \le 1$$

$$\boxed{\therefore \quad z' = \frac{z}{f - n} - \frac{n}{f - n}}$$

See http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123__2/Deriving-Projection-Matrices.htm

D3D orthographic results (LHS)

$$\boxed{x' = \frac{2x}{r-l} - \frac{r+l}{r-l}}$$

$$\boxed{y' = \frac{2y}{t-b} - \frac{t+b}{t-b}}$$

$$\boxed{z' = \frac{z}{f-n} - \frac{n}{f-n}}$$

Georgia Institute of Technology

D3D orthographic matrix (LHS default) ✓

- Direct3D primarily uses LHS, z from 0 to 1, row vectors

$$[x',y',z',1]=[x,y,z,1]P \quad \text{where } P=\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{1}{f-n} & 0 \\ \frac{l+r}{l-r} & \frac{t+b}{b-t} & \frac{n}{n-f} & 1 \end{bmatrix}$$

- In Direct3D: D3DXMatrixOrthoOffCenterLH(*o,l,r,b,t,n,f)

http://msdn.microsoft.com/en-us/library/bb205347(VS.85).aspx

D3D orthographic math for Z (RHS weird)

- For RHS, in most API calls z clip parameters are positive, and clip space switches to using a LHS
- Derive z’

$$-z \in [n, f] \qquad z' \in [0,\ 1]$$

$$n \le -z \le f$$

$$0 \le -z - n \le f - n$$

$$0 \le \frac{-z - n}{f - n} \le 1$$

$$0 \le \frac{-z}{f-n} - \frac{n}{f-n} \le 1$$

$$\boxed{\therefore\ z' = \frac{-z}{f-n} - \frac{n}{f-n}}$$

See http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123__2/Deriving-Projection-Matrices.htm

Georgia Institute of Technology

D3D orthographic matrix (RHS weird) ✓

$$[x', y', z', 1] = [x, y, z, 1]P \quad \text{where } P = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \boxed{\frac{1}{n-f}} & 0 \\ \frac{l+r}{l-r} & \frac{t+b}{b-t} & \frac{n}{n-f} & 1 \end{bmatrix}$$

- In Direct3D: D3DXMatrixOrthoOffCenterRH(*o,l,r,b,t,n,f)
- In XNA: Matrix.CreateOrthographicOffCenter(l,r,b,t,n,f)

http://msdn.microsoft.com/en-us/library/bb205348(VS.85).aspx
http://www.cs.utk.edu/~vose/c-stuff/opengl/glOrtho.html

Simpler D3D ortho matrix (LHS default) ✓

- Most orthographic projection setups
  - Z-axis passes through the center of your view volume
  - Field of view (FOV) extends equally far
    - To the *left* as to the *right* **(i.e., r = -l)**
    - To the *top* as to the *below* **(i.e., t=-b)**

$$[x', y', z', 1] = [x, y, z, 1]P \quad \text{where } P = \begin{bmatrix} \frac{2}{w} & 0 & 0 & 0 \\ 0 & \frac{2}{h} & 0 & 0 \\ 0 & 0 & \frac{1}{f-n} & 0 \\ 0 & 0 & \frac{n}{n-f} & 1 \end{bmatrix}$$

- In Direct3D: D3DXMatrixOrthoLH(*o,w,h,n,f)

See http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123__2/Deriving-Projection-Matrices.htm

Simpler D3D ortho matrix (RHS weird)

✓

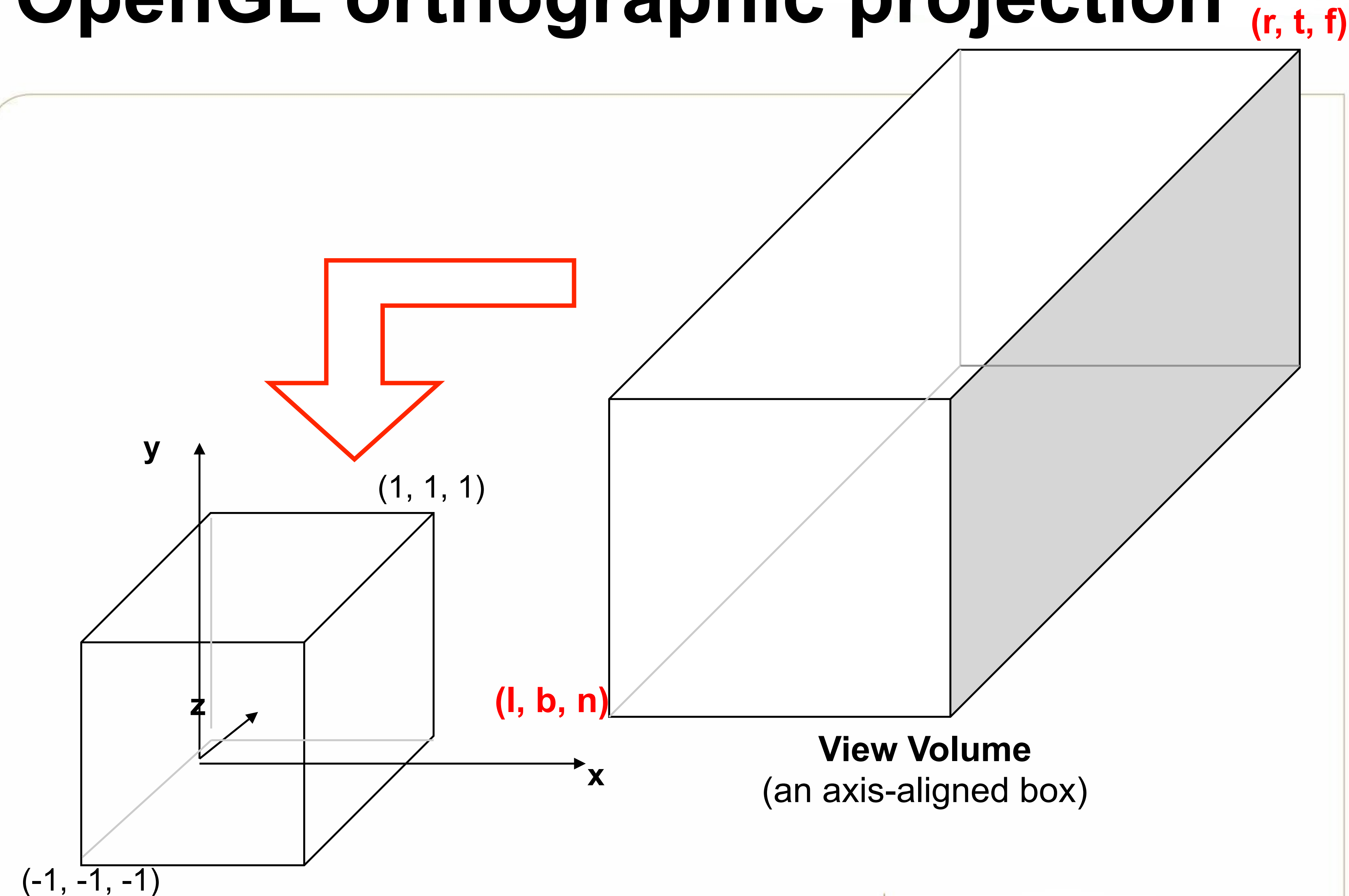
- For RHS, in most API calls z input parameters are positive, and clip space switches to using a LHS

$$[x',y',z',1]=[x,y,z,1]P \quad \text{where } P=\begin{bmatrix} \frac{2}{w} & 0 & 0 & 0 \\ 0 & \frac{2}{h} & 0 & 0 \\ 0 & 0 & \boxed{\frac{1}{n-f}} & 0 \\ 0 & 0 & \frac{n}{n-f} & 1 \end{bmatrix}$$

- In Direct3D: D3DXMatrixOrthoRH(*o,w,h,n,f)
- In XNA: Matrix.CreateOrthographic(w,h,n,f)

See http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123__2/Deriving-Projection-Matrices.htm

OpenGL orthographic projection



Canonical view volume (OpenGL)

View Volume
(an axis-aligned box)

OpenGL orthographic math for Z (RHS)

- Derive z'

$$-z \in [n, f] \qquad z' \in [-1,\ 1]$$

$$n \le -z \le f$$

$$0 \le -z - n \le f - n$$

$$0 \le \frac{-z-n}{f-n} \le 1$$

$$0 \le \frac{2(-z-n)}{f-n} \le 2$$

$$-1 \le \frac{2(-z-n)}{f-n} - 1 \le 1$$

$$-1 \le \frac{-2z-2n-f+n}{f-n} \le 1$$

$$-1 \le \frac{-2z}{f-n} - \frac{f+n}{f-n} \le 1$$

$$\boxed{\therefore \quad z' = \frac{-2z}{f-n} - \frac{f+n}{f-n}}$$

See http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123__2/Deriving-Projection-Matrices.htm

OpenGL orthographic results

$$\boxed{x' = \frac{2x}{r-l} - \frac{r+l}{r-l}}$$

$$\boxed{y' = \frac{2y}{t-b} - \frac{t+b}{t-b}}$$

$$\boxed{z' = \frac{-2z}{f-n} - \frac{f+n}{f-n}}$$

Ortho proj matrix (OpenGL/Unity) ✓

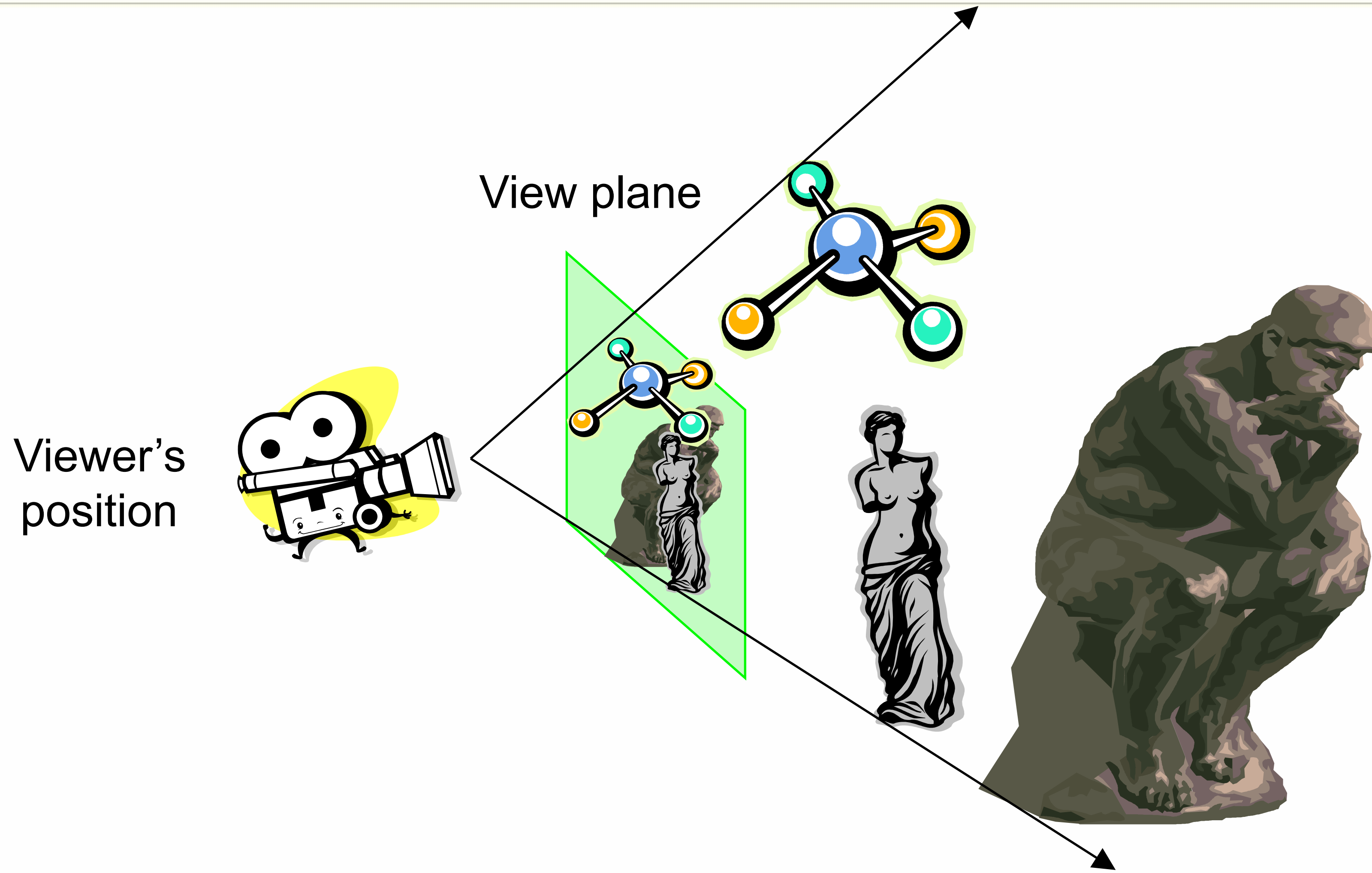
- For RHS, in most API calls z input parameters are positive, and clip space switches to using a LHS

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1' \end{bmatrix} = P \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \text{ where } P = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \boxed{\frac{-2}{f-n}} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- In OpenGL: glOrtho(l,r,b,t,n,f)
- In Unity: Matrix4x4.Ortho(l,r,b,t,n,f) ????????????

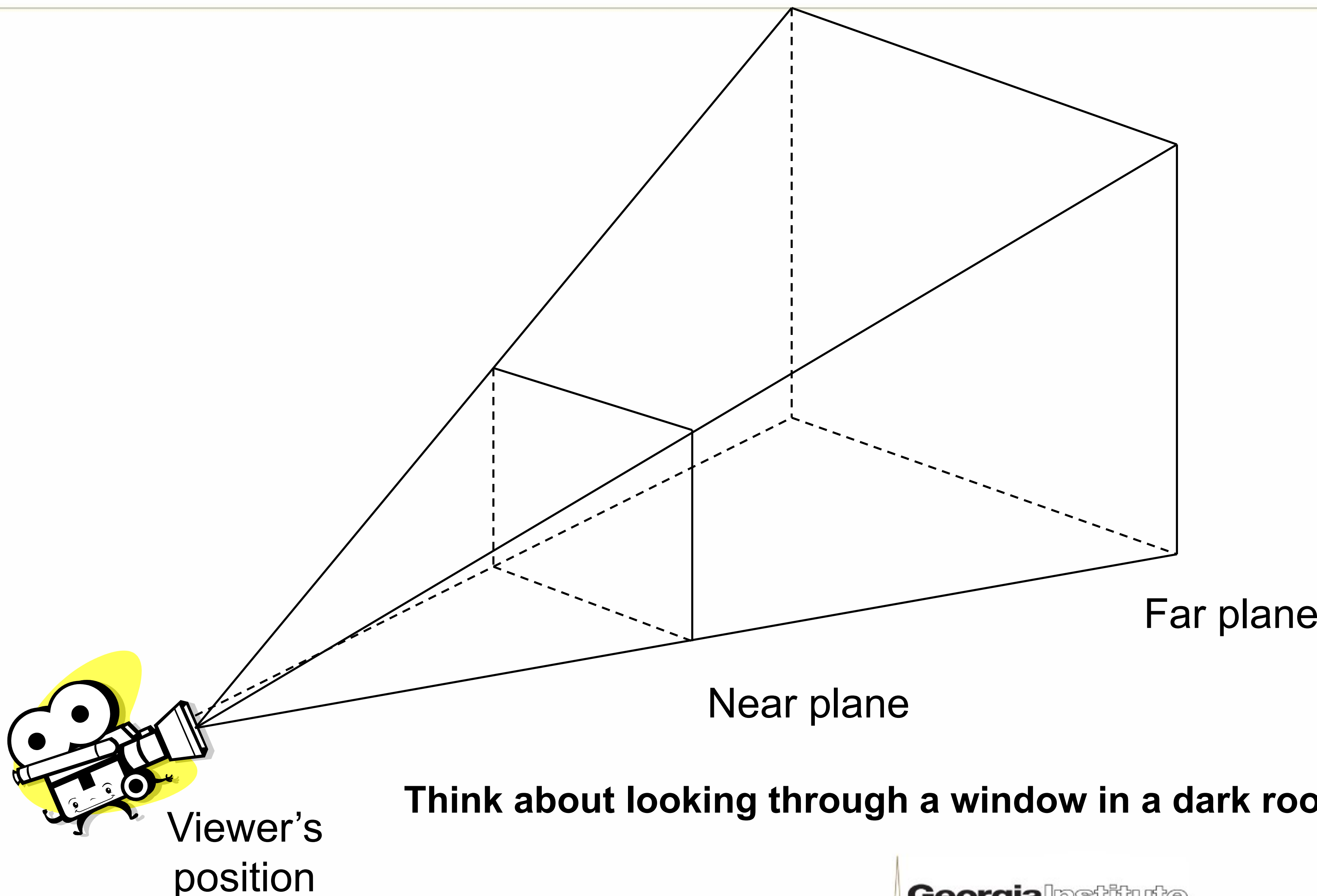
See http://www.songho.ca/opengl/gl_projectionmatrix.html#ortho

Perspective projection



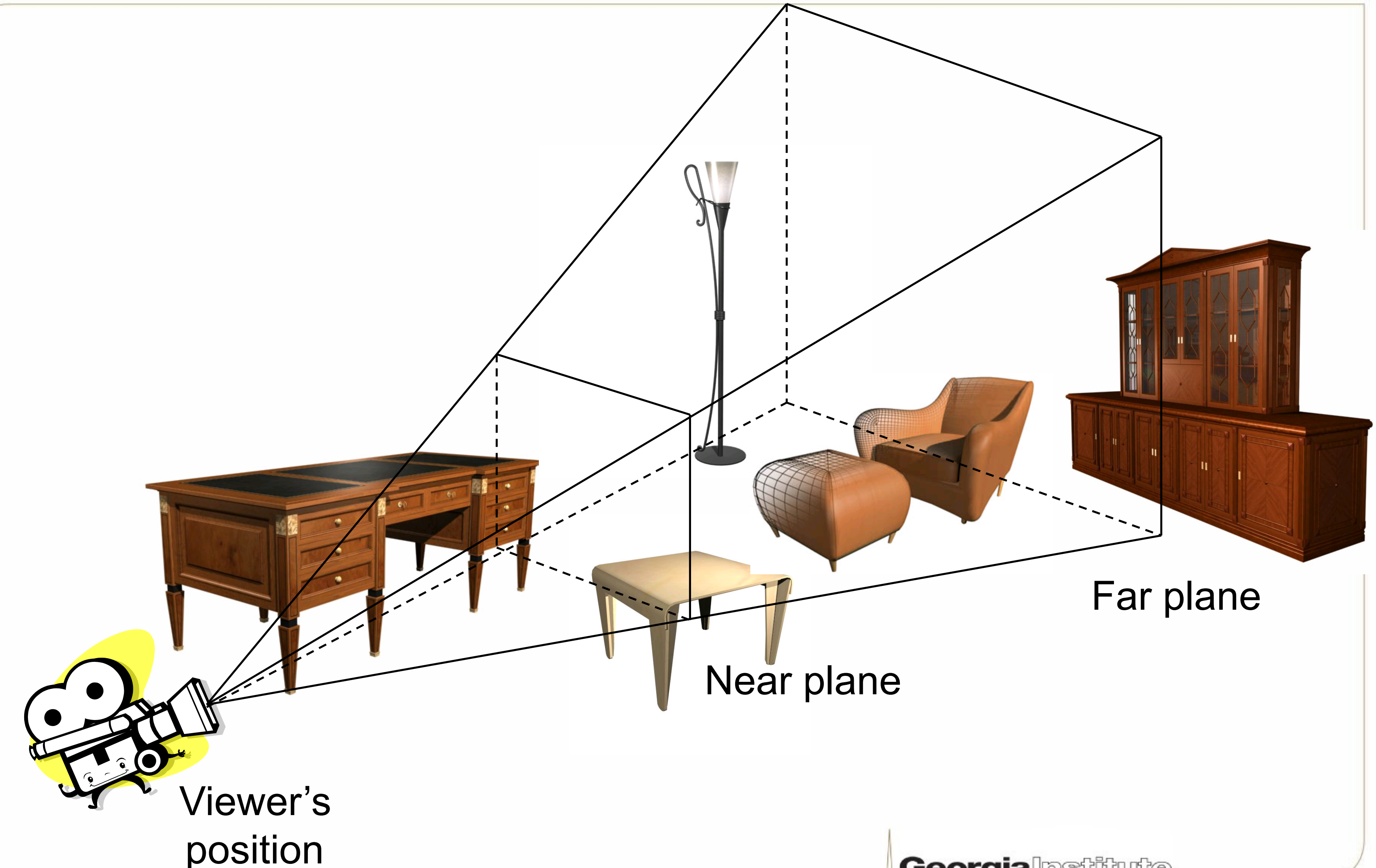
Georgia Institute of Technology

Viewing frustum



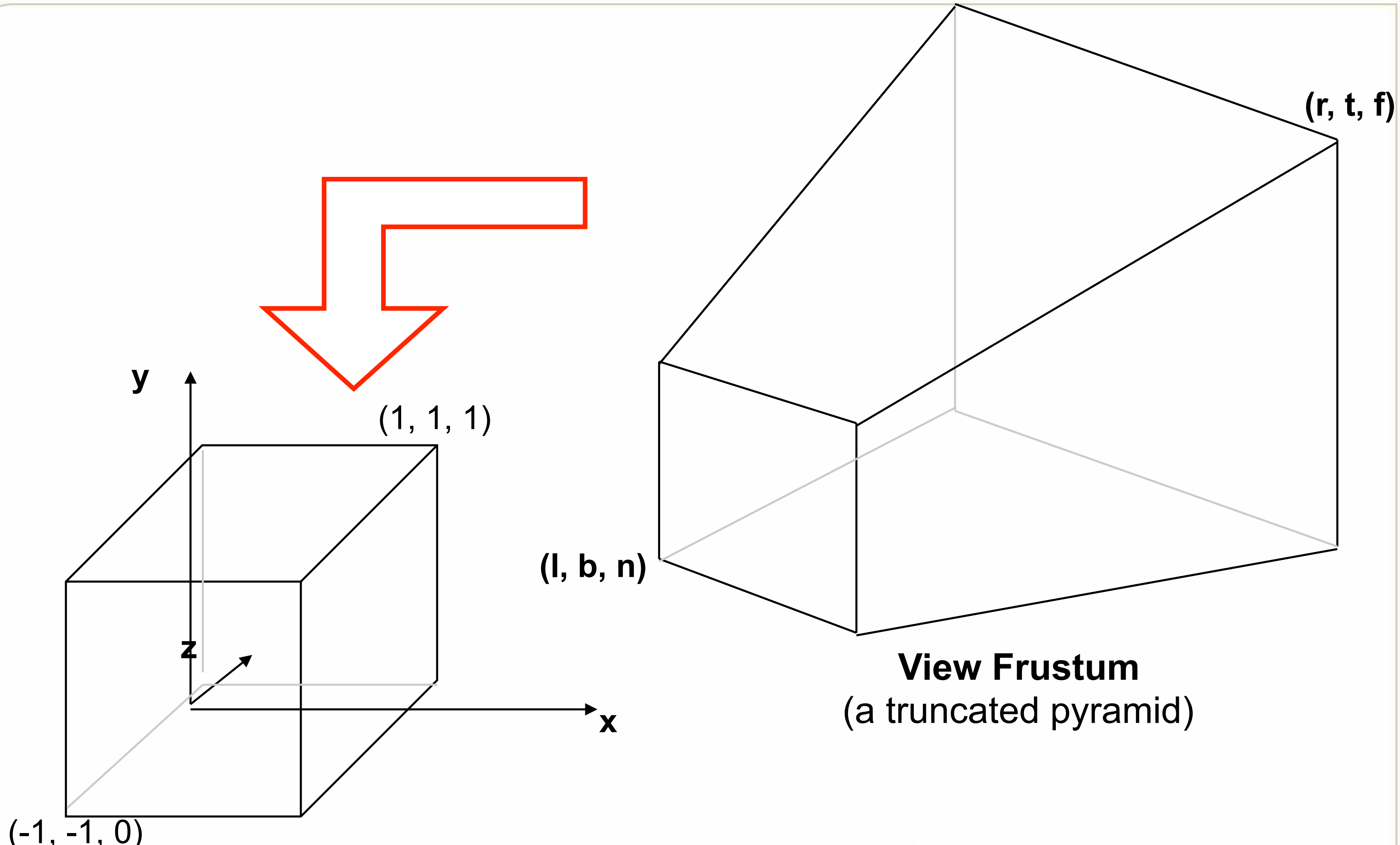
Think about looking through a window in a dark room

Viewing frustum with furniture



Georgia Institute of Technology

Direct3D perspective projection

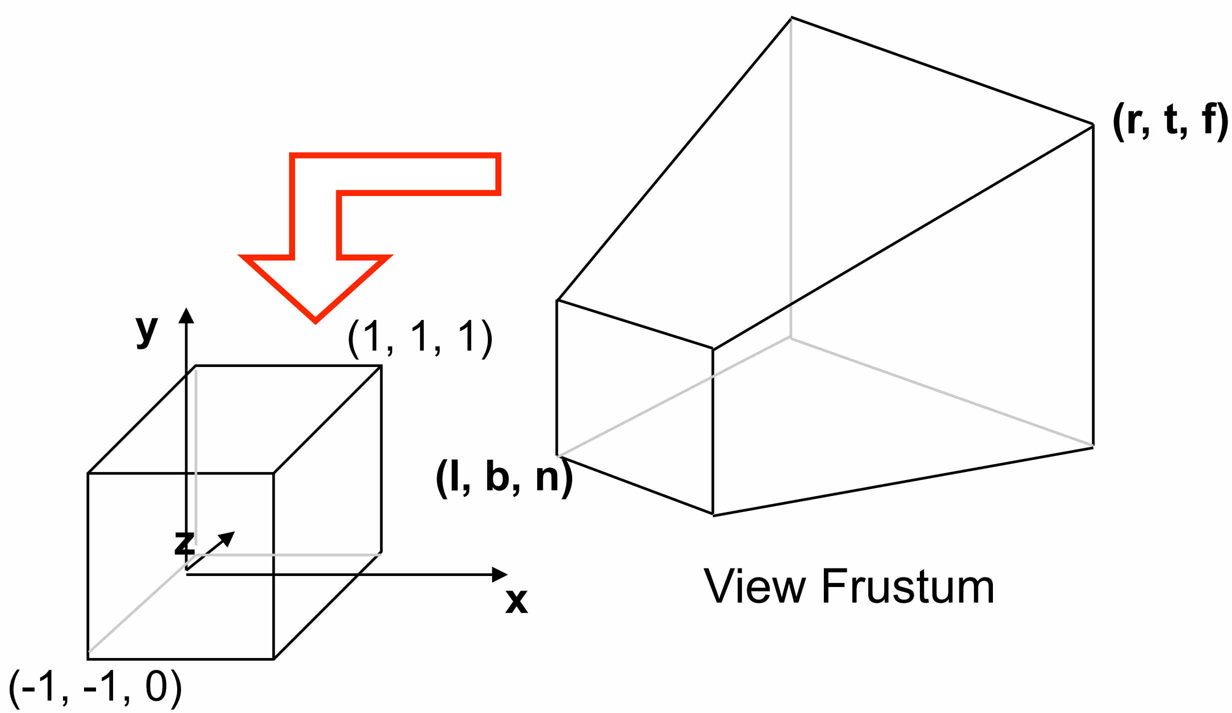


Canonical view volume (Direct3D)

View Frustum
(a truncated pyramid)

D3D perspective proj mapping

- Given a point (x,y,z) within the view frustum, project it onto the near plane z=n
- We will map x from [l,r] to [-1,1] and y from [b,t] to [-1,1]

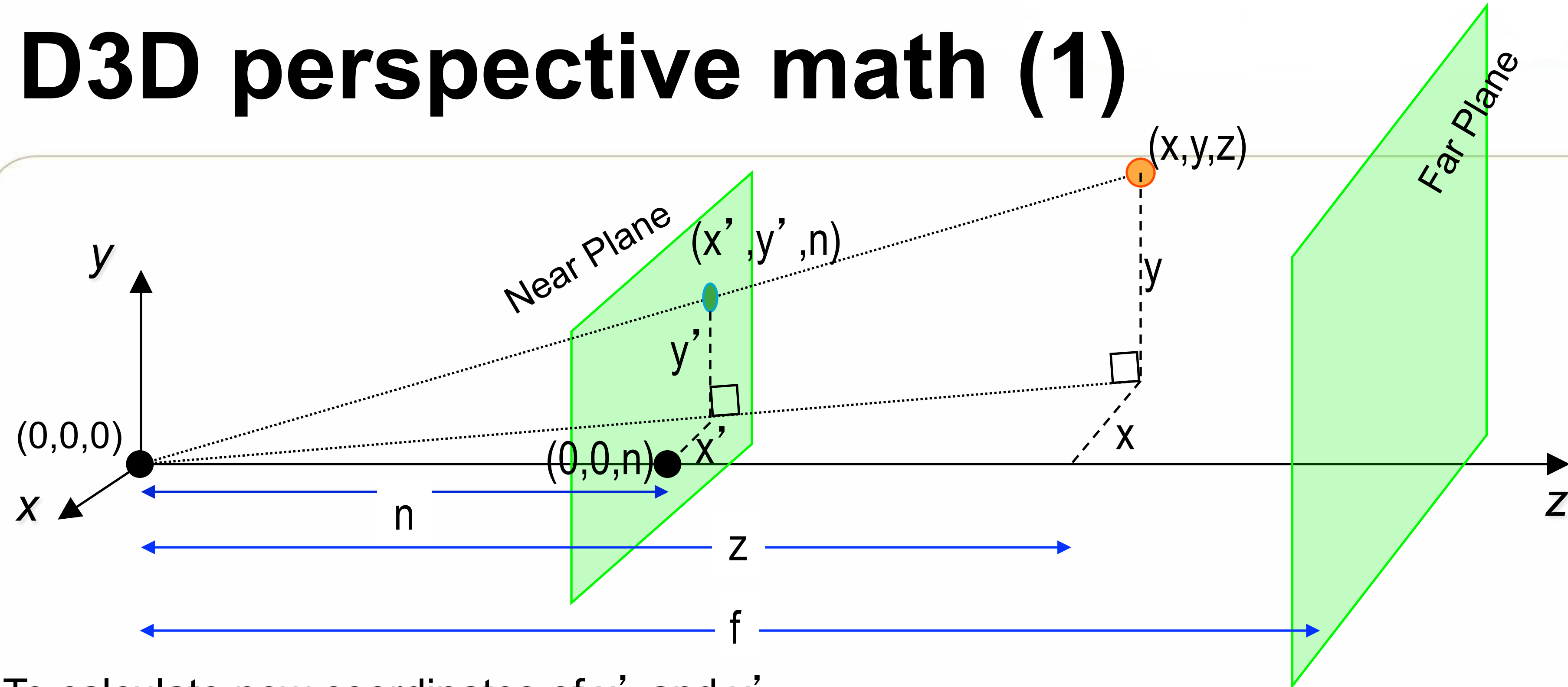


Canonical view volume (Direct3D)

See http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123__3/Deriving-Projection-Matrices.htm

Georgia Institute of Technology

D3D perspective math (1)



To calculate new coordinates of x' and y'

$$\frac{x'}{x} = \frac{n}{z} \Rightarrow x' = \frac{nx}{z}$$

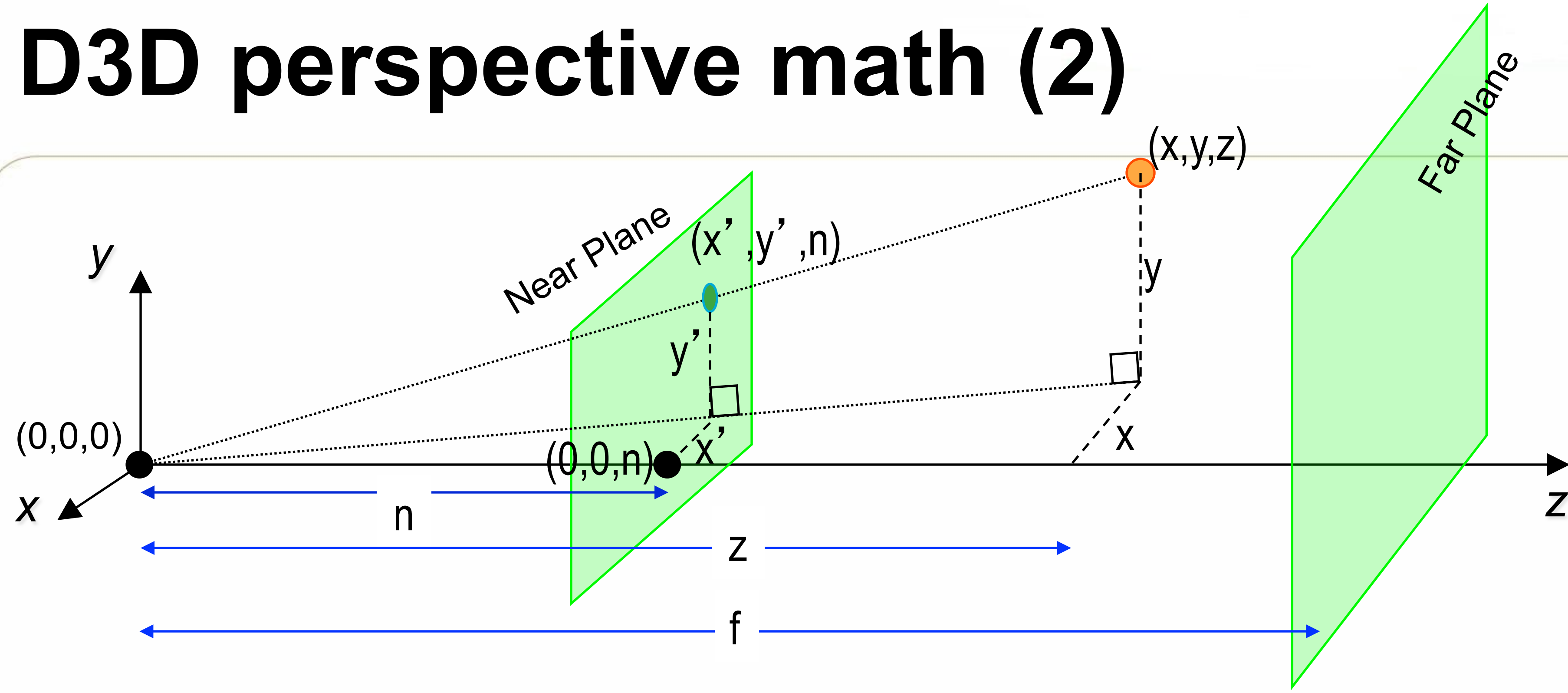
$$\frac{y'}{x'} = \frac{y}{x} \Rightarrow y' = \frac{yx'}{x} = \frac{y}{x} \cdot \frac{nx}{z} = \frac{ny}{z}$$

Next apply our orthographic projection formulas

See http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123__3/Deriving-Projection-Matrices.htm

Georgia Institute of Technology

D3D perspective math (2)



$$x' = \frac{2}{r-l}\cdot\frac{nx}{z} - \frac{r+l}{r-l} \qquad y' = \frac{2n}{t-b}\cdot\frac{ny}{z} - \frac{t+b}{t-b}$$

$$x'z = \frac{2n}{r-l}x - \frac{r+l}{r-l}z \qquad y'z = \frac{2n}{t-b}y - \frac{t+b}{t-b}z$$

Now let's tackle the z' component

See http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123__3/Deriving-Projection-Matrices.htm

D3D perspective math (3)

$$x'z = \frac{2n}{r-l}x - \frac{r+l}{r-l}z$$

$$y'z = \frac{2n}{t-b}y - \frac{t+b}{t-b}z$$

$$z'z = pz + q \quad \text{where } p \text{ and } q \text{ are constants}$$

- We know z (depth) transformation has nothing to do with x and y

See http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123__3/Deriving-Projection-Matrices.htm

Georgia Institute of Technology

D3D perspective math (4)

$z'z = pz + q$ where p and q are constants

$$\boxed{\begin{aligned} 0 &= pn + q \\ f &= pf + q \end{aligned}} \qquad \therefore p = \frac{f}{f-n} \quad \text{and} \quad q = -\frac{fn}{f-n}$$

$$z'z = \frac{f}{f-n} z - \frac{fn}{f-n}$$

- We know (boxed equations above)
  - z' = 0 when z=n (near plane)
  - z' = 1 when z=f (far plane)

See http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123__3/Deriving-Projection-Matrices.htm

General D3D perspective matrix

$$x'z = \frac{2n}{r-l}x - \frac{r+l}{r-l}z$$

$$y'z = \frac{2n}{t-b}y - \frac{t+b}{t-b}z$$

$$z'z = \frac{f}{f-n}z - \frac{fn}{f-n}$$

$$w'z = z$$

$$[x'z, y'z, z'z, w'z] = [x, y, z, 1]P \quad \text{where } P = \begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ -\frac{r+l}{r-l} & -\frac{t+b}{t-b} & \frac{f}{f-n} & 1 \\ 0 & 0 & -\frac{fn}{f-n} & 0 \end{bmatrix}$$

Georgia Institute of Technology

See http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123__3/Deriving-Projection-Matrices.htm

Simpler D3D perspective matrix

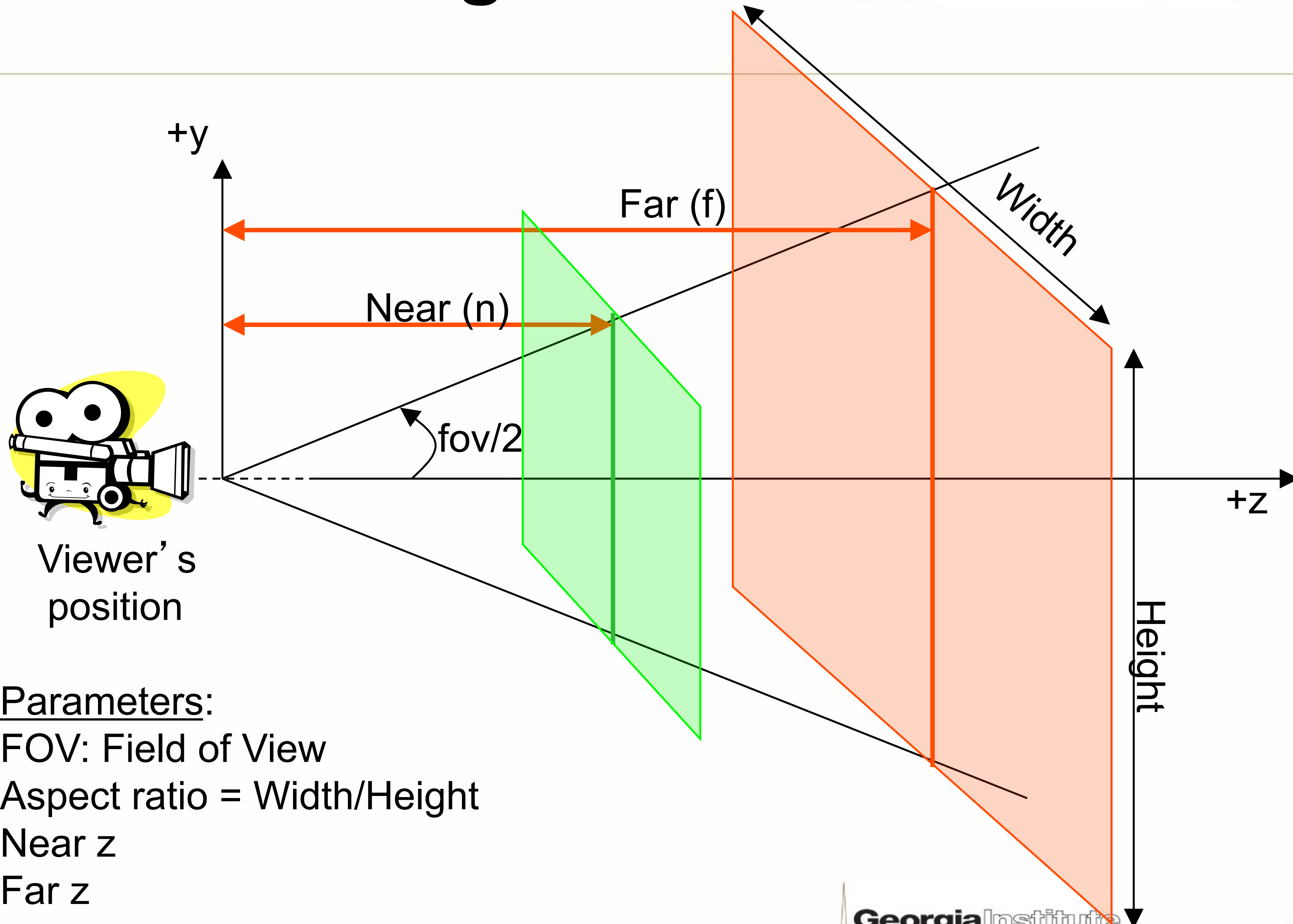
- Similar to orthographic projection, if *l=-r and t=-b,* we can simplify to

$$[x'z, y'z, z'z, w'z] = [x, y, z, 1]P \quad \text{where } P = \begin{bmatrix} \frac{2n}{w} & 0 & 0 & 0 \\ 0 & \frac{2n}{h} & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & 1 \\ 0 & 0 & -\frac{fn}{f-n} & 0 \end{bmatrix}$$

- In any case, we will have to divide by z to obtain [x',y', z', w']
  - Implemented by dividing by the fourth (w'z) coordinate

See http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123__3/Deriving-Projection-Matrices.htm

Define viewing frustum

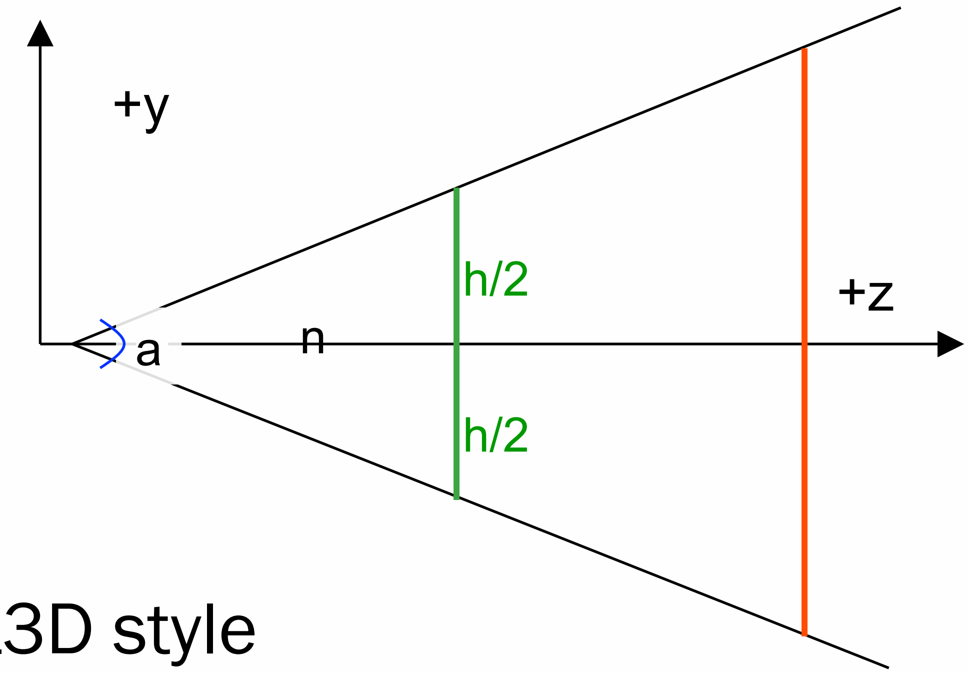


<u>Parameters</u>:
FOV: Field of View
Aspect ratio = Width/Height
Near z
Far z

See http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123__3/Deriving-Projection-Matrices.htm

Reparameterized D3D matrix

$$P = \begin{bmatrix} \boxed{\frac{2n}{w}} & 0 & 0 & 0 \\ 0 & \boxed{\frac{2n}{h}} & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & 1 \\ 0 & 0 & -\frac{fn}{f-n} & 0 \end{bmatrix}$$



$$\cot(\frac{a}{2}) = \frac{2n}{h}$$

Direct3D style

$$r = \frac{w}{h}$$

$$\frac{2n}{w} = \frac{2n}{rh} = \frac{2n}{r\frac{2n}{\cot(\frac{a}{2})}} = \frac{1}{r}\cot(\frac{a}{2})$$

Need to replace w and h with FOV and aspect ratio

See http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123__3/Deriving-Projection-Matrices.htm

D3D perspective matrix (LHS default)

✓

$$[x'z, y'z, z'z, w'z] = [x, y, z, 1]P \quad \text{where } P = \begin{bmatrix} \frac{1}{r}\cdot\cot(\frac{a}{2}) & 0 & 0 & 0 \\ 0 & \cot(\frac{a}{2}) & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & 1 \\ 0 & 0 & -\frac{fn}{f-n} & 0 \end{bmatrix}$$

a: Field of View (FOV) r: aspect ratio $= \frac{\text{width}}{\text{height}}$ n: near plan f: far plane

- In Direct3D: D3DXMatrixPerspectiveFovLH(*o,a,r,n,f)

D3D perspective matrix (RHS weird) ✓

$$[x'z, y'z, z'z, w'z] = [x, y, z, 1]P \quad \text{where } P = \begin{bmatrix} \frac{1}{r}\cot(\frac{a}{2}) & 0 & 0 & 0 \\ 0 & \cot(\frac{a}{2}) & 0 & 0 \\ 0 & 0 & \frac{f}{n-f} & -1 \\ 0 & 0 & \frac{fn}{n-f} & 0 \end{bmatrix}$$

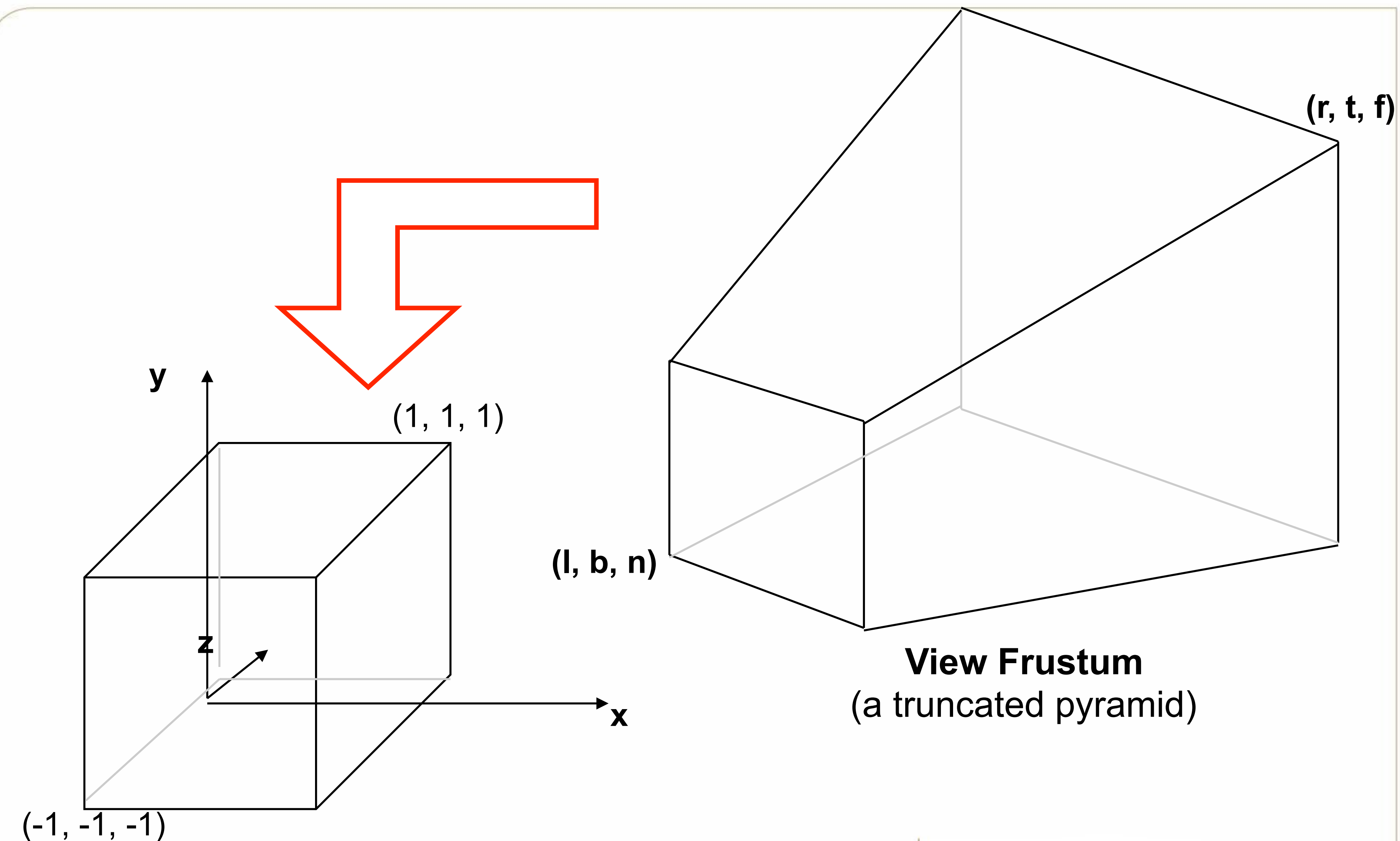
a: Field of View (FOV) r: aspect ratio $= \dfrac{\text{width}}{\text{height}}$ n: near plan f: far plane

- In Direct3D: D3DXMatrixPerspectiveFovRH(*o,a,r,n,f)
- In XNA: Matrix.CreatePerspectiveFieldOfView(a,r,n,f)

http://msdn.microsoft.com/en-us/library/bb205351(VS.85).aspx

Georgia Institute of Technology

OpenGL/LHS perspective projection



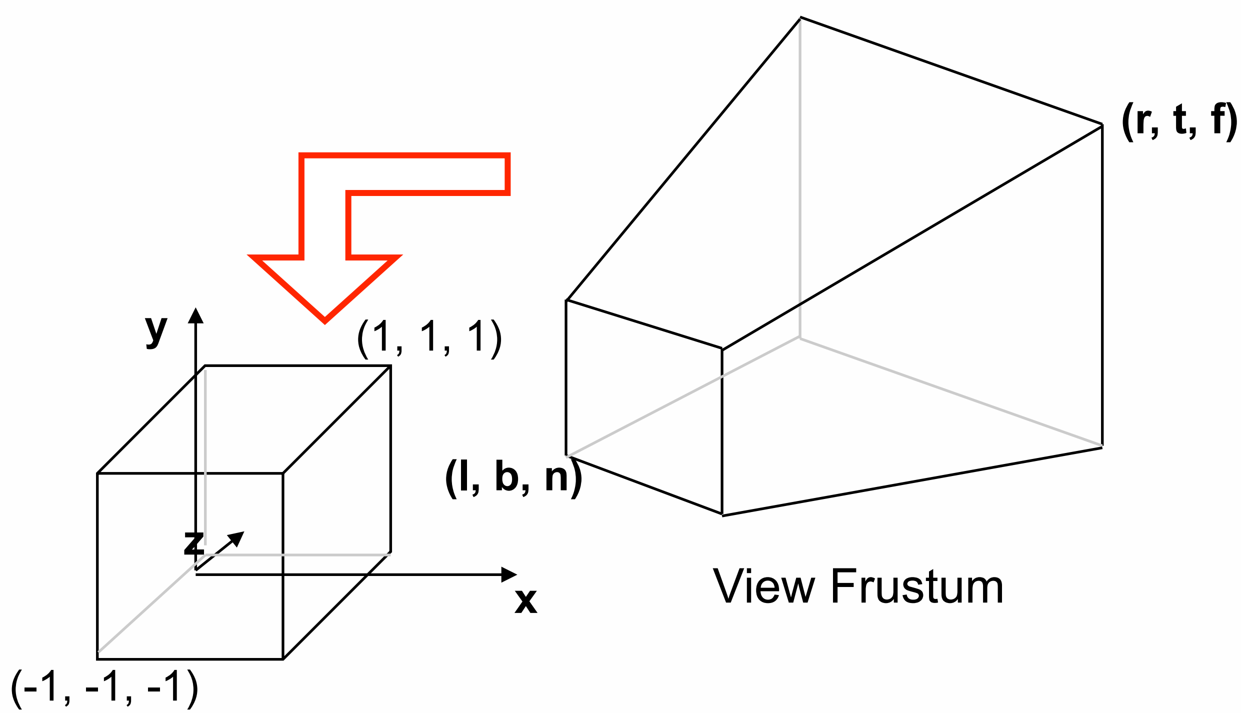
Canonical view volume (Unity)

View Frustum
(a truncated pyramid)

See http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123__3/Deriving-Projection-Matrices.htm

OpenGL/LHS perspective proj mapping

- Given a point (x,y,z) within the view frustum, project it onto the near plane z=n
- We will map x from [l,r] to [-1,1] and y from [b,t] to [-1,1]



Canonical view volume (Unity)

See http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123__3/Deriving-Projection-Matrices.htm

OpenGL/LHS perspective projection math

$z'z = pz + q$ where p and q are constants

$$\boxed{\begin{aligned} -n &= pn + q \\ f &= pf + q \end{aligned}} \quad ? \quad \therefore p = \frac{f+n}{f-n} \quad \text{and} \quad q = -\frac{2fn}{f-n}$$

$$z'z = \frac{f+n}{f-n}z - \frac{2fn}{f-n}$$

- We know (boxed equations above)
  - z' = -1 when z=n (near plane)
  - z' = 1 when z=f (far plane)

See http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123__3/Deriving-Projection-Matrices.htm

General OpenGL/LHS perspective matrix

$$x'z = \frac{2n}{r-l}x - \frac{r+l}{r-l}z$$

$$y'z = \frac{2n}{t-b}y - \frac{t+b}{t-b}z$$

$$z'z = \frac{f+n}{f-n}z - \frac{2fn}{f-n}$$

$$w'z = z$$

$$\begin{bmatrix} x'z \\ y'z \\ z'z \\ w'z \end{bmatrix} = P \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{where } P = \begin{bmatrix} \frac{2n}{r-l} & 0 & -\frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & -\frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

See http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123__3/Deriving-Projection-Matrices.htm

Simpler OpenGL/LHS perspective matrix

- Similar to orthographic projection, if *l=-r and t=-b,* we can simplify to

$$\begin{bmatrix} x'z \\ y'z \\ z'z \\ w'z \end{bmatrix} = P \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{where } P = \begin{bmatrix} \frac{2n}{w} & 0 & 0 & 0 \\ 0 & \frac{2n}{h} & 0 & 0 \\ 0 & 0 & \frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

- In any case, we will have to divide by z to obtain [x',y', z', w']
  - Implemented by dividing by the fourth (w'z) coordinate

See http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123__3/Deriving-Projection-Matrices.htm

Georgia Institute of Technology

Simpler OpenGL/RHS perspective matrix ✓

$$\begin{bmatrix} x'z \\ y'z \\ z'z \\ w'z \end{bmatrix} = P \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{where } P = \begin{bmatrix} \frac{1}{r}\cot(\frac{a}{2}) & 0 & 0 & 0 \\ 0 & \cot(\frac{a}{2}) & 0 & 0 \\ 0 & 0 & \boxed{\frac{f+n}{n-f}} & \frac{2fn}{n-f} \\ 0 & 0 & \boxed{-1} & 0 \end{bmatrix}$$

a: Field of View (FOV) r: aspect ratio $= \dfrac{\text{width}}{\text{height}}$ n: near plan f: far plane

- In OpenGL: gluPerspective(a,r,n,f)

http://msdn.microsoft.com/en-us/library/bb205350(VS.85).aspx

Georgia Institute of Technology

Unity perspective matrix ✓

$$\begin{bmatrix} x'z \\ y'z \\ z'z \\ w'z \end{bmatrix} = P \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{where } P = \begin{bmatrix} \frac{1}{r}\cot(\frac{a}{2}) & 0 & 0 & 0 \\ 0 & \cot(\frac{a}{2}) & 0 & 0 \\ 0 & 0 & \frac{f+n}{f-n} & \boxed{\frac{2fn}{f-n}} \\ 0 & 0 & \boxed{-1} & 0 \end{bmatrix}$$

a: Field of View (FOV) r: aspect ratio $= \dfrac{\text{width}}{\text{height}}$ n: near plan f: far plane

- In Unity: Matrix4x4.Perspective(a,r,n,f) ?????????????

http://msdn.microsoft.com/en-us/library/bb205350(VS.85).aspx

Custom projections in Unity

- From Camera.projectionMatrix documentation:

"Use a custom projection only if you really need a non-standard projection.

This property is used by Unity's water rendering to setup an *oblique projection* matrix.

Using custom projections requires good knowledge of transformation and projection matrices."

Unity's 2-D coordinate systems

- Viewport space:
  - (0,0) is bottom-left
  - (1,1) is top-right
- Screen space coordinates:
  - z "is in world units from the camera"
  - (0,0) is bottom-left
  - (Camera.pixelWidth,Camera.pixelHeight) is top-right
- GUI space coordinates:
  - (0,0) is upper-left
  - (Camera.pixelWidth,Camera.pixelHeight) is bottom-right

Georgia Institute of Technology

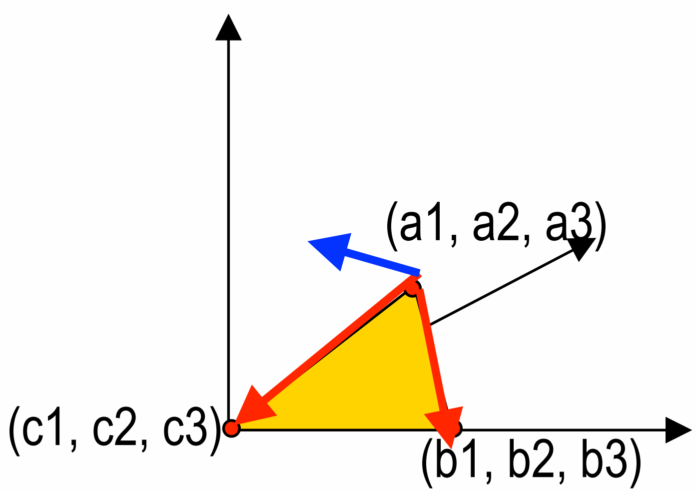
Viewport transformation

- The actual 2D projection to the viewer
- Copy to your back buffer (frame buffer)
- Can be programmed, scaled, ...

Georgia Institute of Technology

Backface culling

- Determine "facing direction"
- Triangle order matters
- How to compute a normal vector for 2 given vectors?
  - Using **cross product** of 2 given vectors



2 Vectors

$$\overrightarrow{V1} = (b1 - a1)\mathbf{i} + (b2 - a2)\mathbf{j} + (b3 - a3)\mathbf{k}$$

$$\overrightarrow{V2} = (c1 - a1)\mathbf{i} + (c2 - a2)\mathbf{j} + (c3 - a3)\mathbf{k}$$

Cross product

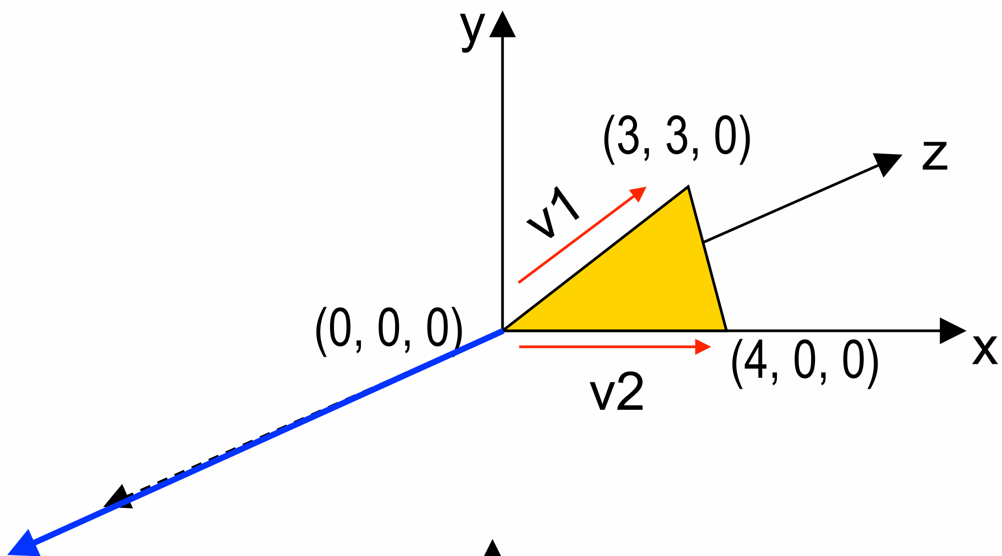
$$\overrightarrow{V1} = x_1\mathbf{i} + x_2\mathbf{j} + x_3\mathbf{k}$$

$$\overrightarrow{V2} = y_1\mathbf{i} + y_2\mathbf{j} + y_3\mathbf{k}$$

$$\overrightarrow{V1} \times \overrightarrow{V2} = (x_2 y_3 - x_3 y_2)\mathbf{i} + (x_3 y_1 - x_1 y_3)\mathbf{j} + (x_1 y_2 - x_2 y_1)\mathbf{k}$$

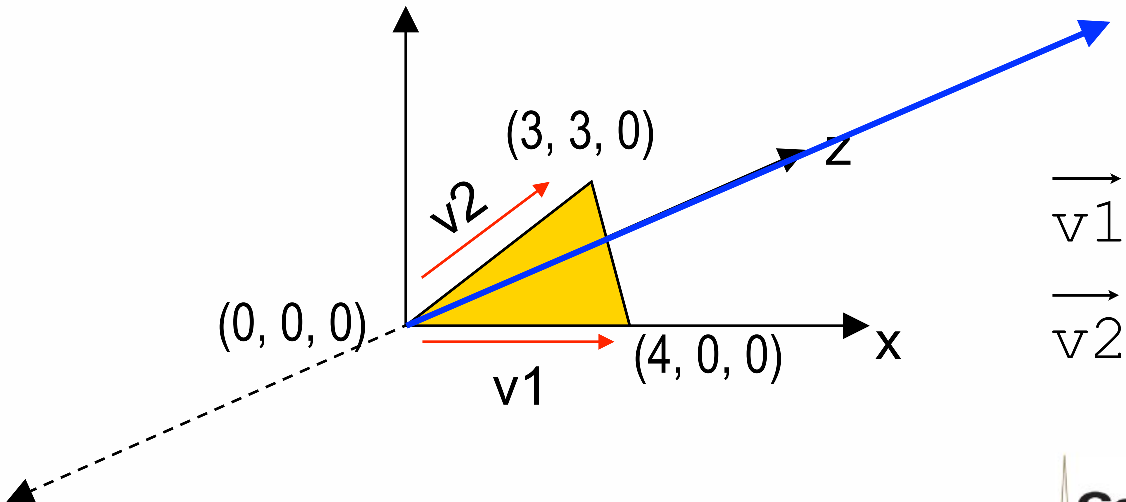
Compute the surface normal for a triangle

- Clockwise normals, LHS



$$\overrightarrow{v1} = 3\mathbf{i} + 3\mathbf{j} + 0\mathbf{k}$$

$$\overrightarrow{v2} = 4\mathbf{i} + 0\mathbf{j} + 0\mathbf{k}$$



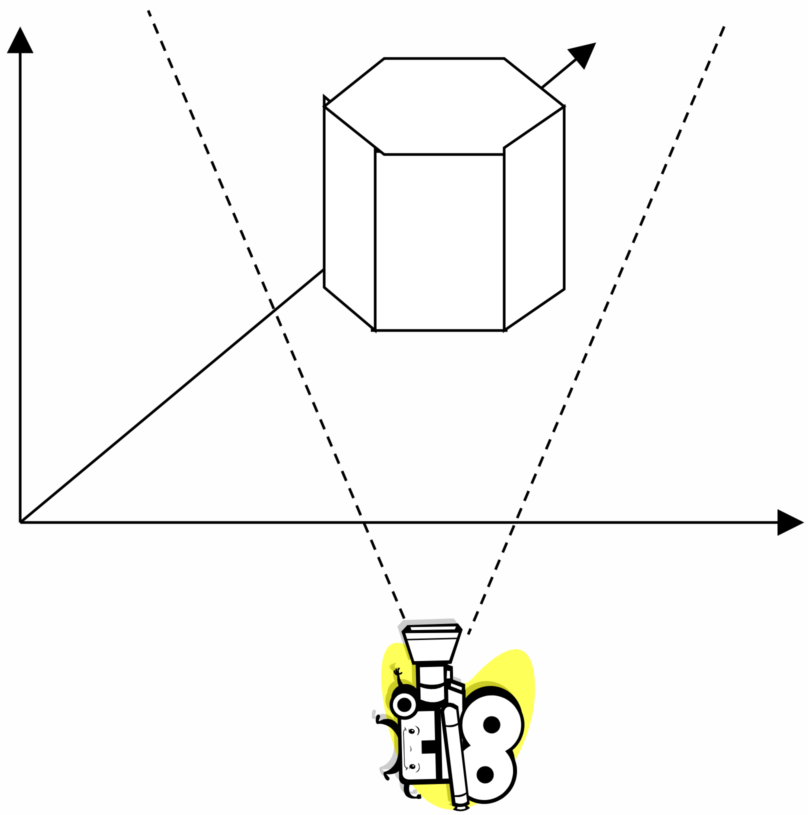
$$\overrightarrow{v1} = 4\mathbf{i} + 0\mathbf{j} + 0\mathbf{k}$$

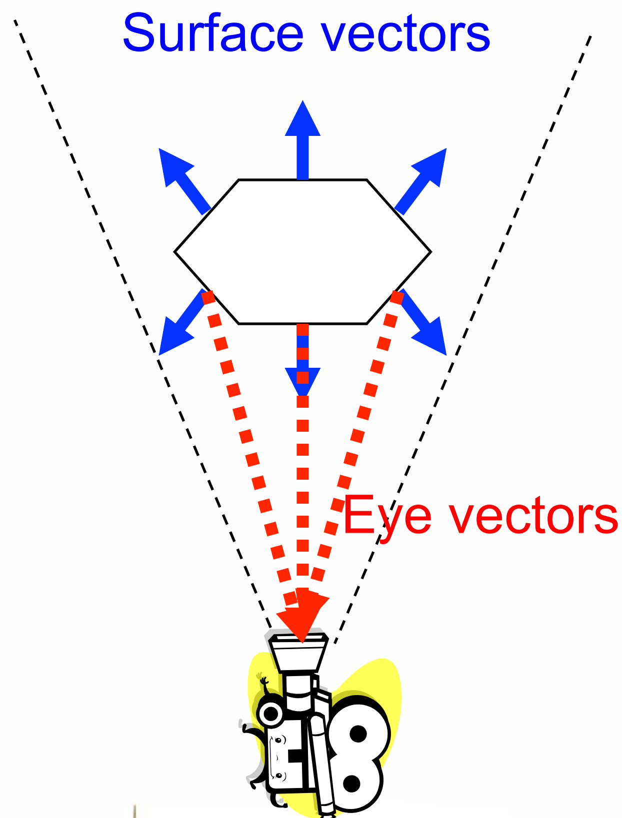
$$\overrightarrow{v2} = 3\mathbf{i} + 3\mathbf{j} + 0\mathbf{k}$$

Georgia Institute of Technology

Backface culling method (1)

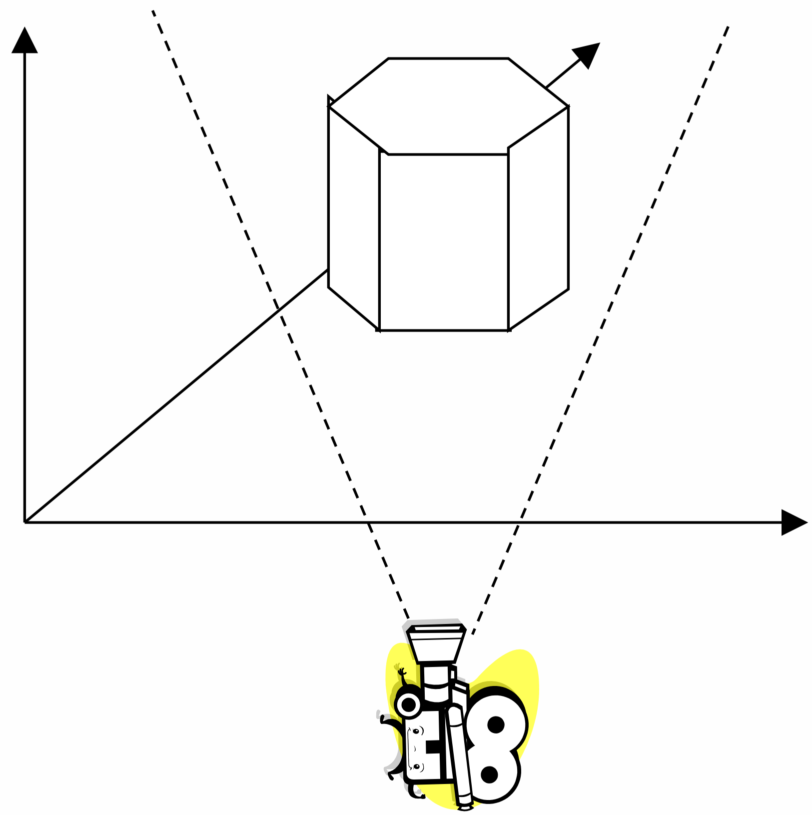
- Check if the normal is facing the camera
- How to determine that?
  - Use Dot Product

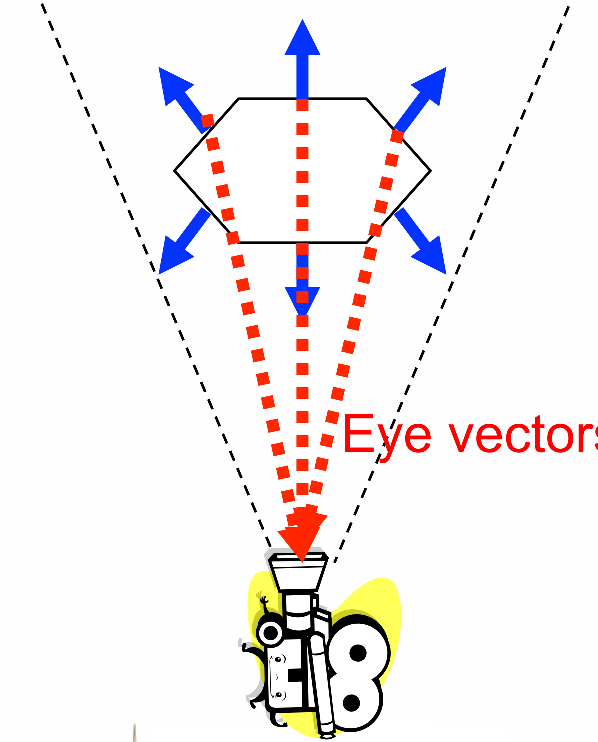




Backface culling method (2)

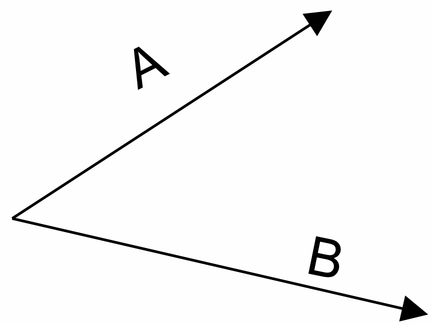
- Check if the normal is facing the camera
- How to determine that?
  - Use Dot Product





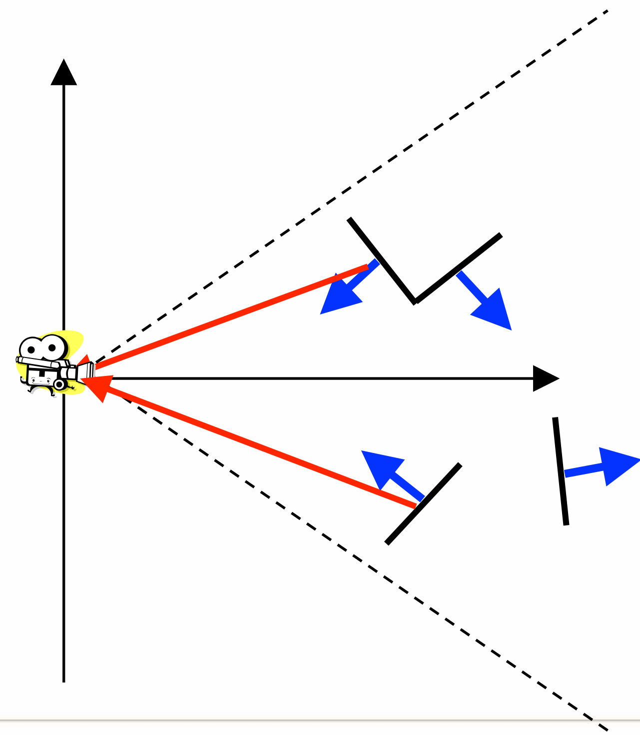
Georgia Institute of Technology

Dot product method (1)

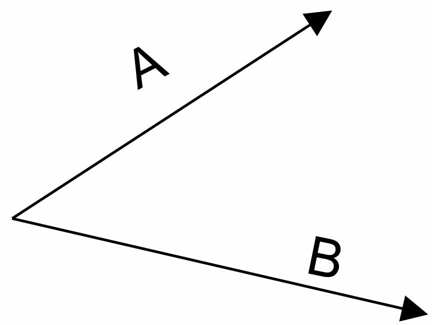


$$\vec{A} \bullet \vec{B} = |A||B|\cos\theta$$

$$\vec{A} \bullet \vec{B} > 0 \;\; \Rightarrow \;\; -\frac{\pi}{2} < \theta < \frac{\pi}{2}$$

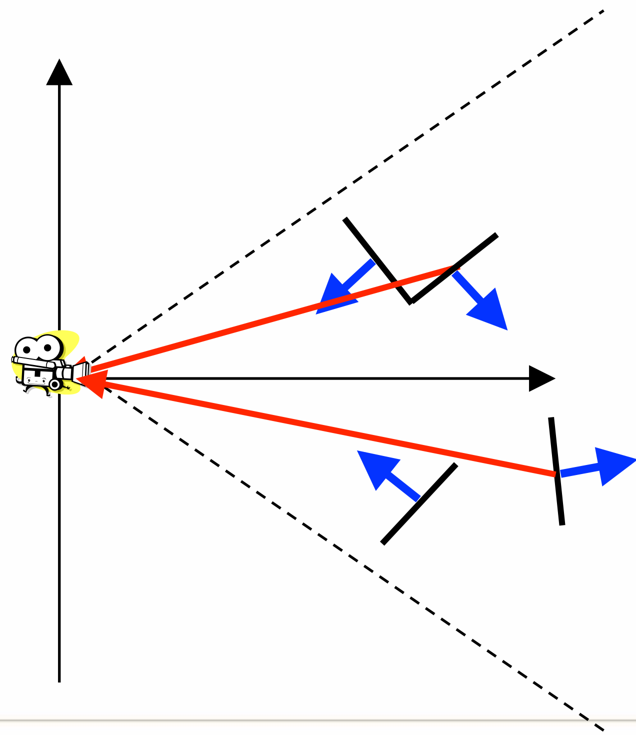


Dot product method (2)

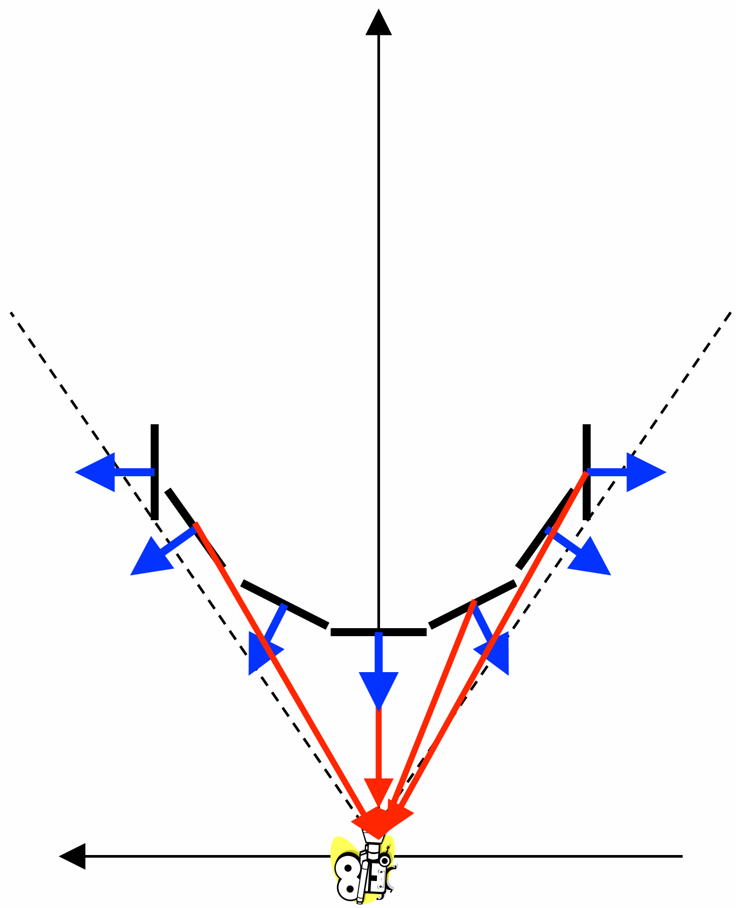


$$\vec{A} \bullet \vec{B} = |A||B|\cos\theta$$

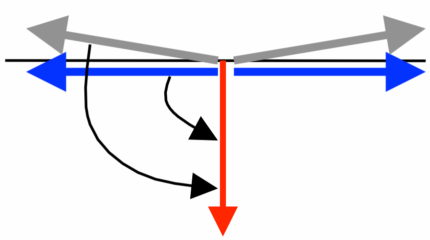
$$\vec{A} \bullet \vec{B} > 0 \;\Rightarrow\; -\frac{\pi}{2} < \theta < \frac{\pi}{2}$$



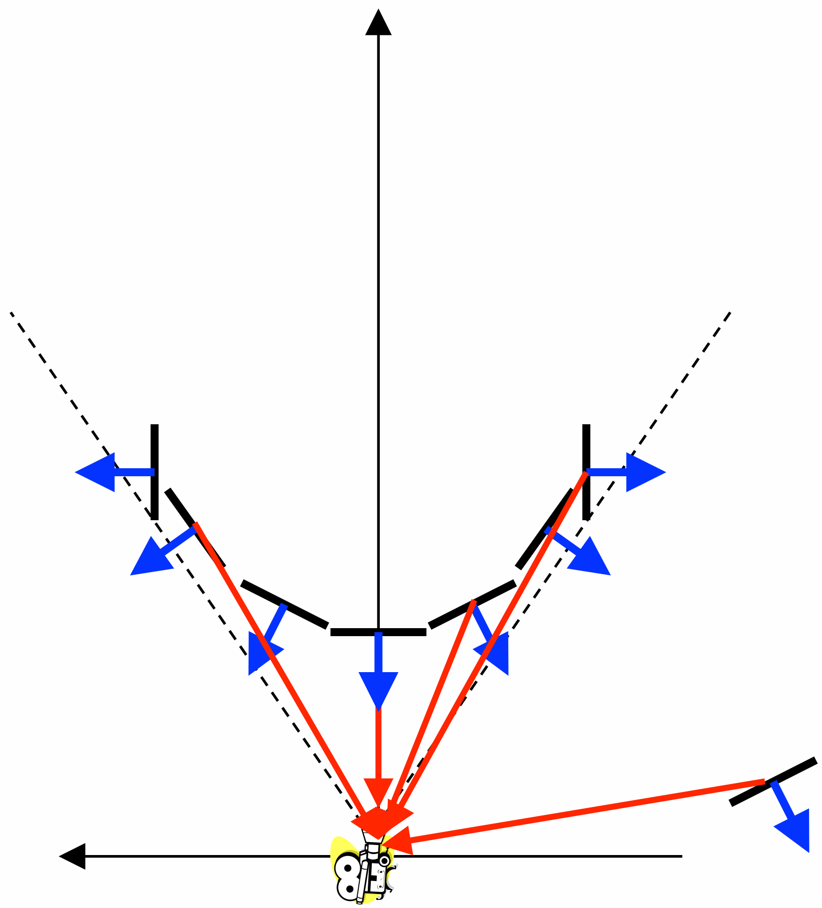
Dot product method (3)



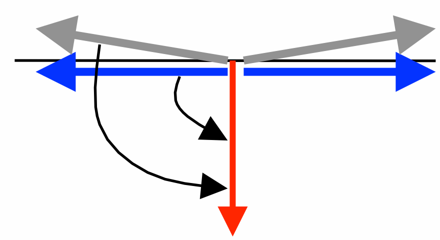
$$\vec{A} \bullet \vec{B} > 0 \;\Rightarrow\; -\frac{\pi}{2} < \theta < \frac{\pi}{2}$$



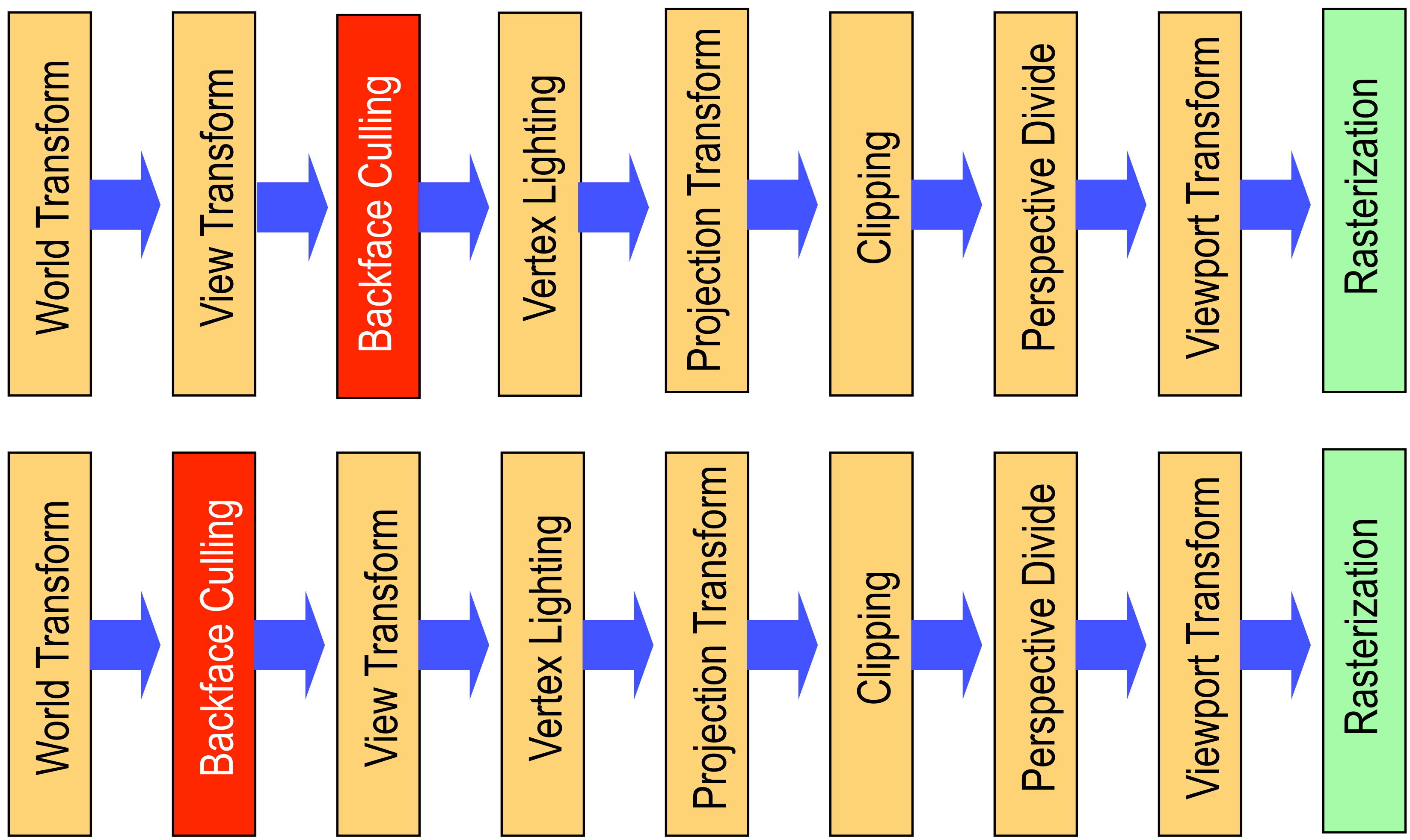
Caution!



$$\vec{A} \bullet \vec{B} > 0 \;\Rightarrow\; -\frac{\pi}{2} < \theta < \frac{\pi}{2}$$

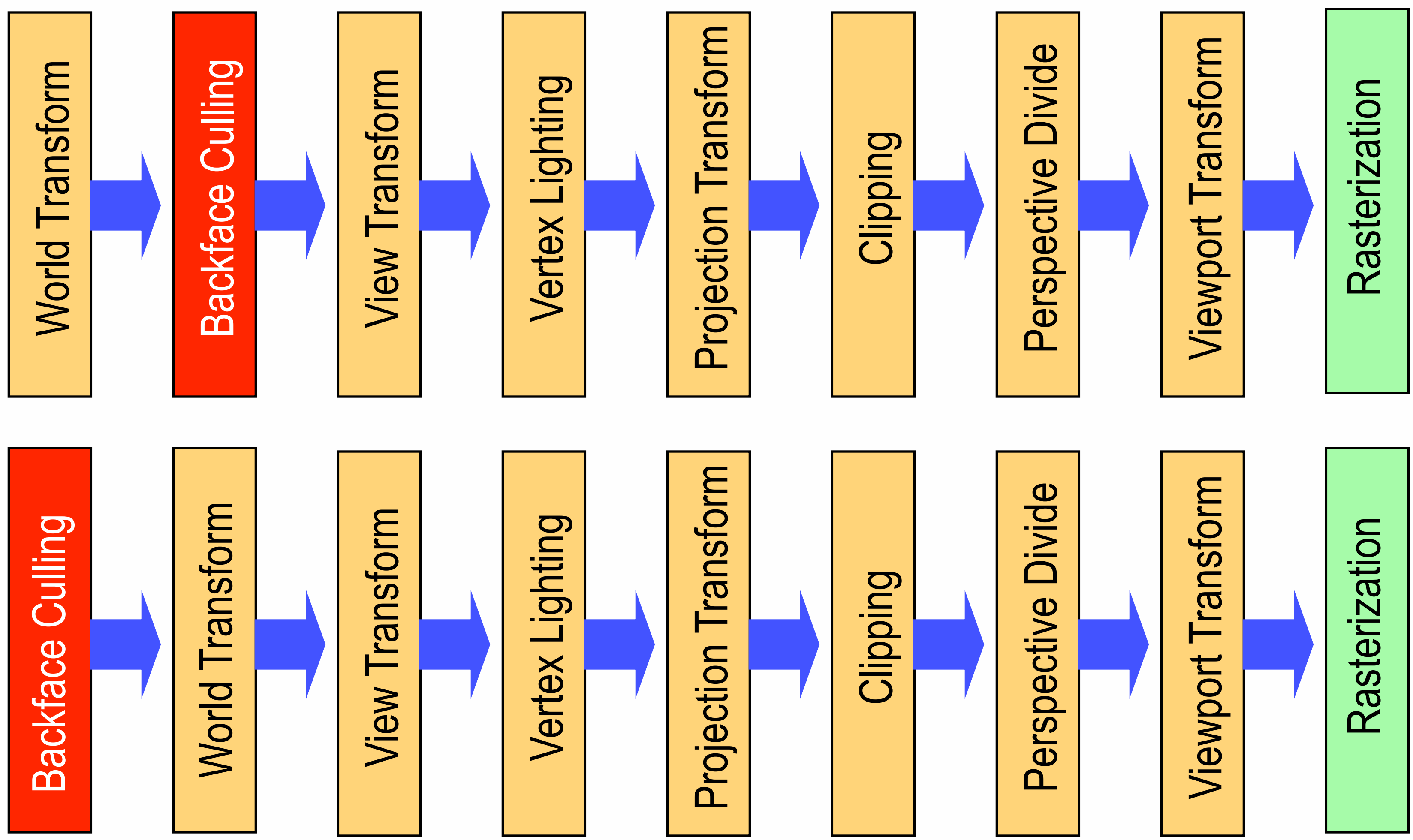


When to perform backface culling?



Make sure camera is in correct coordinates!

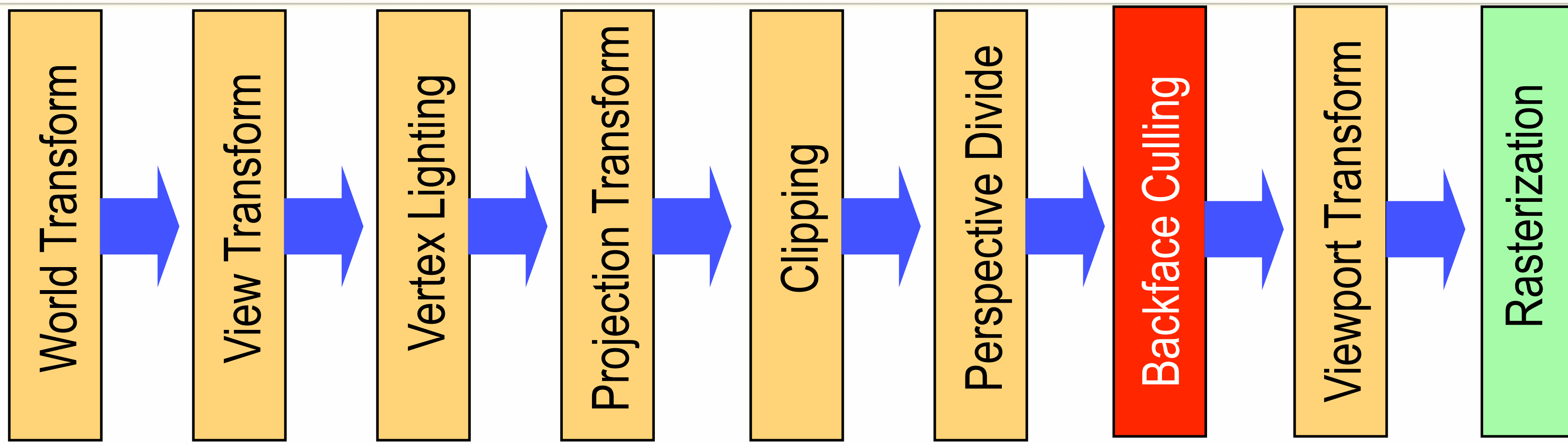
How about before you even start?



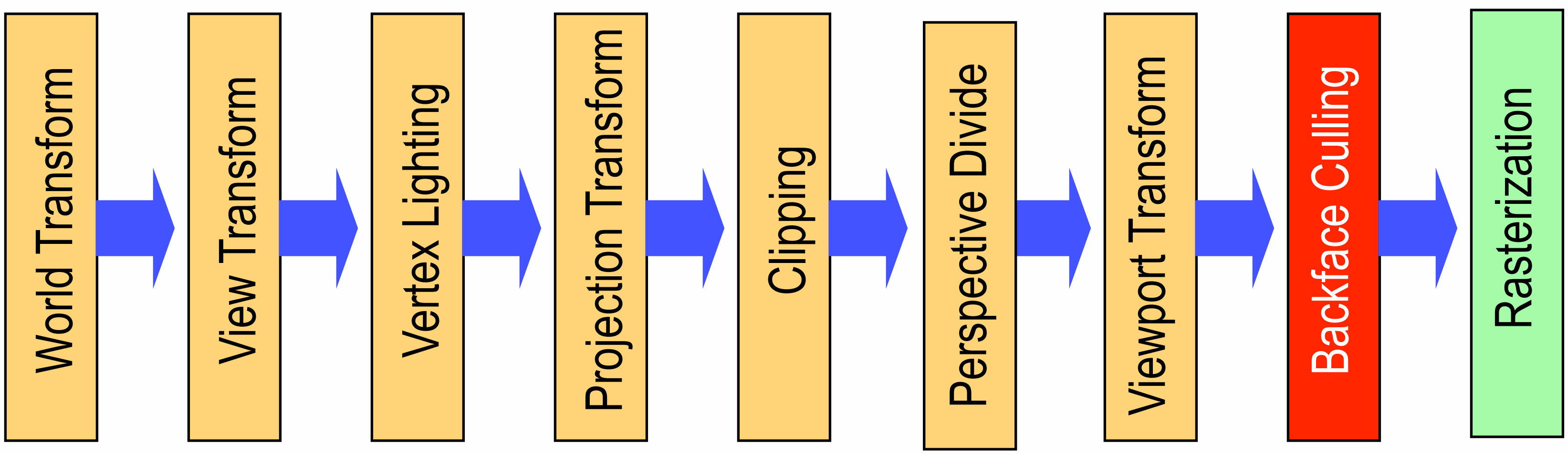
Transform camera vectors into object spaces

Georgia Institute of Technology

Or how about at the very end?



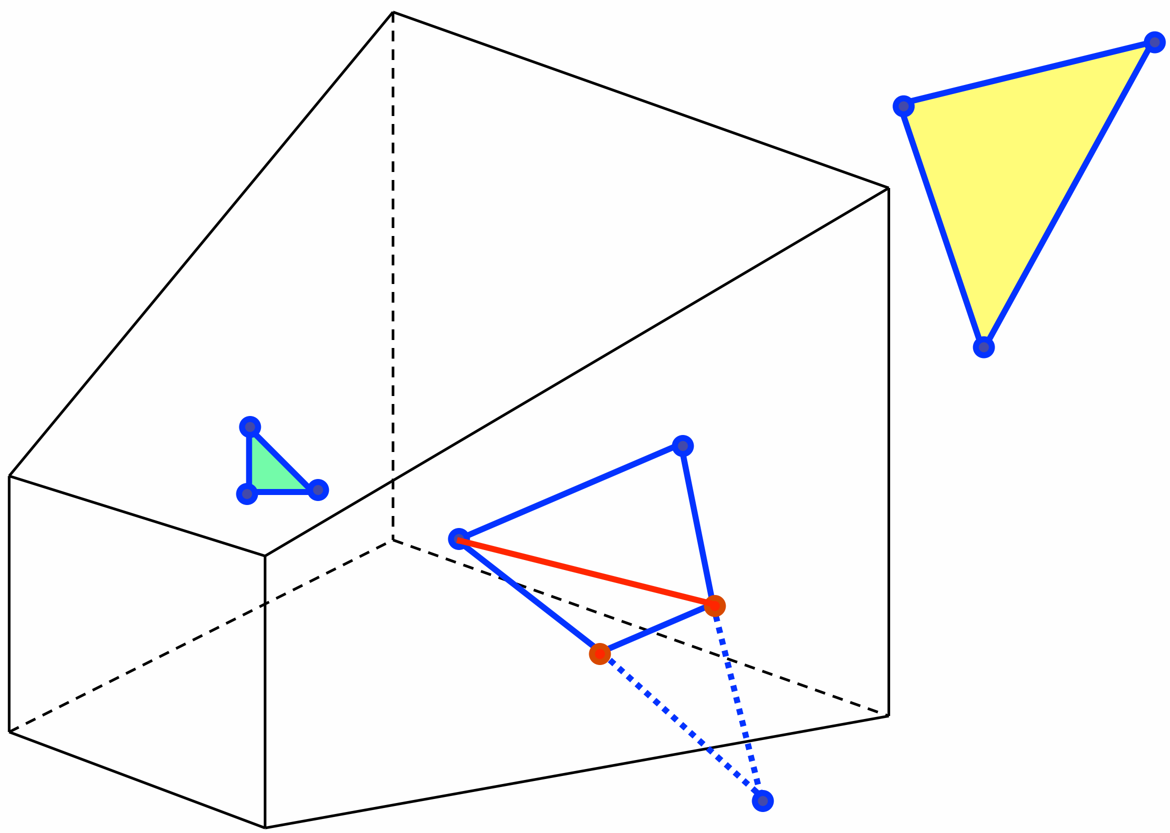
- Now you can just check "winding order" in 2D



- As "officially" done by OpenGL

Georgia Institute of Technology

3D clipping



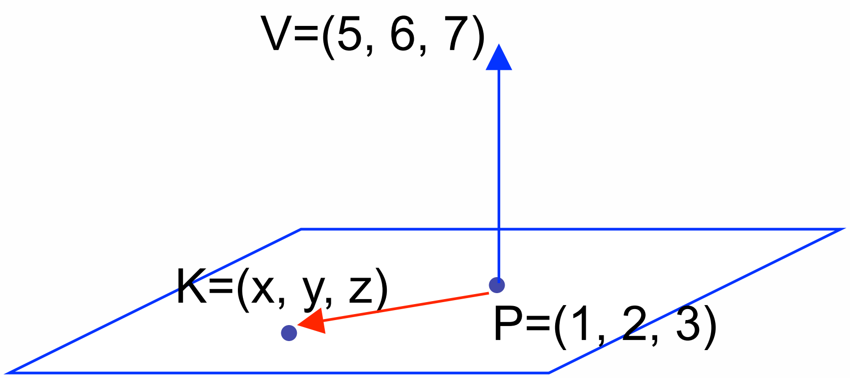
- Test 6 planes if a triangle is inside, outside, or partially inside the view frustum
- Clipping creates new triangles (triangulation)
  - Interpolate new vertices info

Appendix

Clipping against a plane

- Test each vertex of a triangle
  - Outside
  - Inside
  - Partially inside
- Incurred computation overhead
- Save unnecessary computation (and bandwidth) later
- Need to know how to determine a plane
- Need to know how to determine a vertex is inside or outside a plane

Specifying a plane

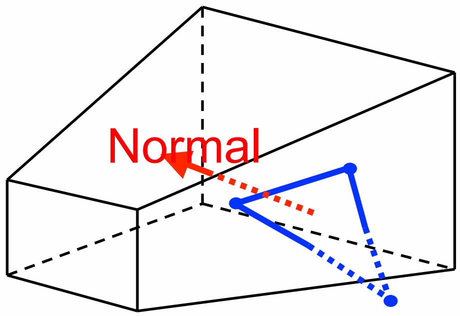


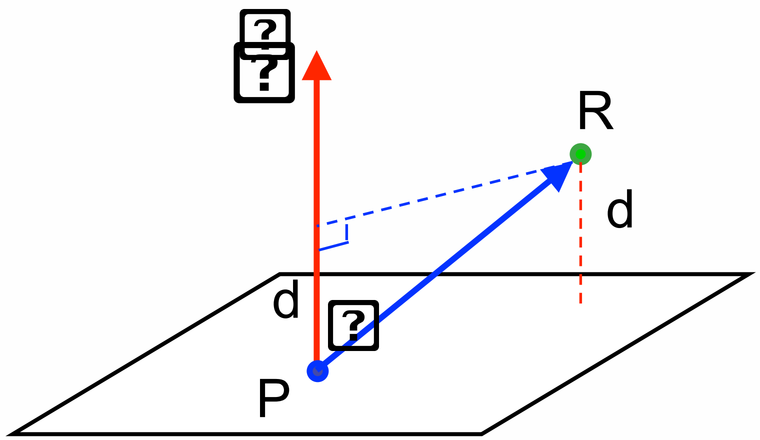
Plane equation

5*(x-1)+6*(y-2)+7*(z-3)=0

- You need two things to specify a plane
  - A point on the plane *(p0, p1, p2)*
  - A vector (normal) perpendicular to the plane *(a, b, c)*
  - Plane → $a*(x - p0) + b*(y - p1) + c*(z - p2) = 0$

Distance calculation from a plane (1)

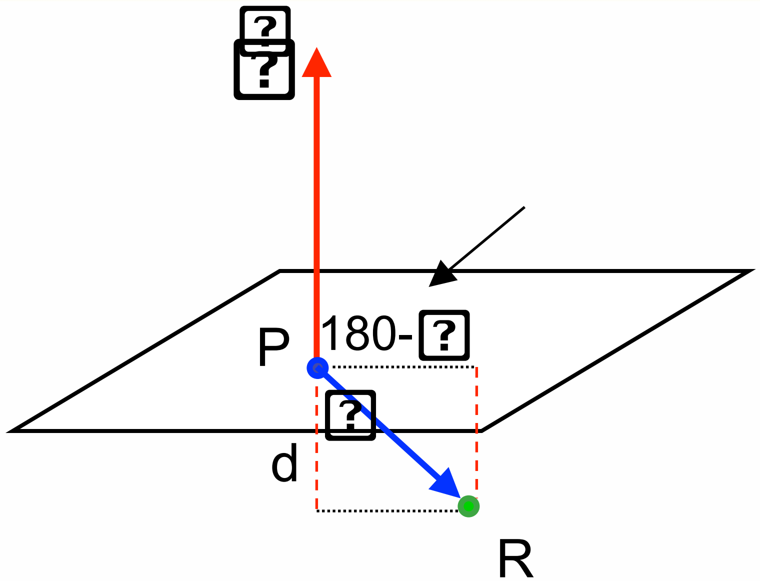




$$d = |R-P| \cdot \cos\theta = |R-P| \cdot \frac{\vec{v} \bullet (R-P)}{\left|\vec{v}\right| \cdot \left|R-P\right|} = \frac{\vec{v} \bullet (R-P)}{\left|\vec{v}\right|}$$

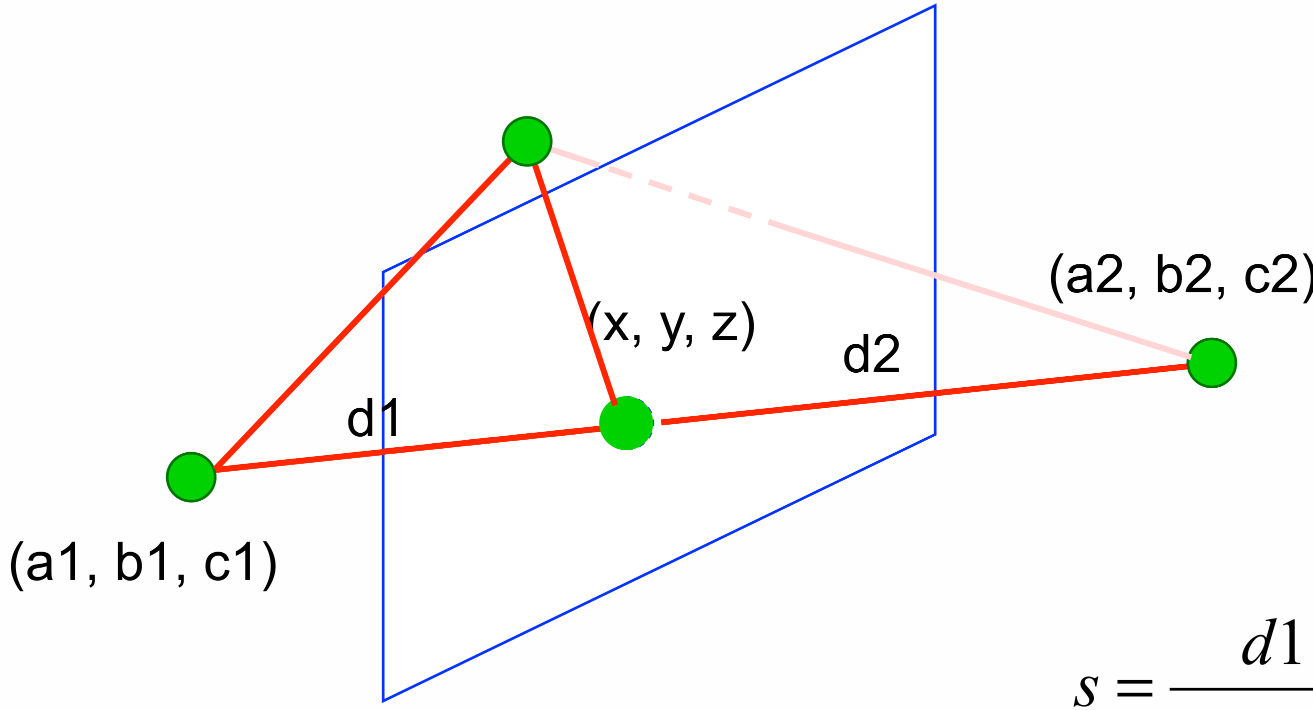
- Given a point R, calculate the distance
  - Distance > 0 inside the plane
  - Distance = 0 on the plane
  - Distance < 0 outside the plane

Distance calculation from a plane (2)



$$\mathrm{d} = |R - P| \cdot \cos(180 - \theta)$$

Triangulation using interpolation



$$s = \frac{d1}{d1 + d2}$$

$$x = a1 + s \cdot (a2 - a1)$$

$$y = b1 + s \cdot (b2 - b1)$$

$$z = c1 + s \cdot (c2 - c1)$$

Georgia Institute of Technology