CPSC 314
Computer Graphics

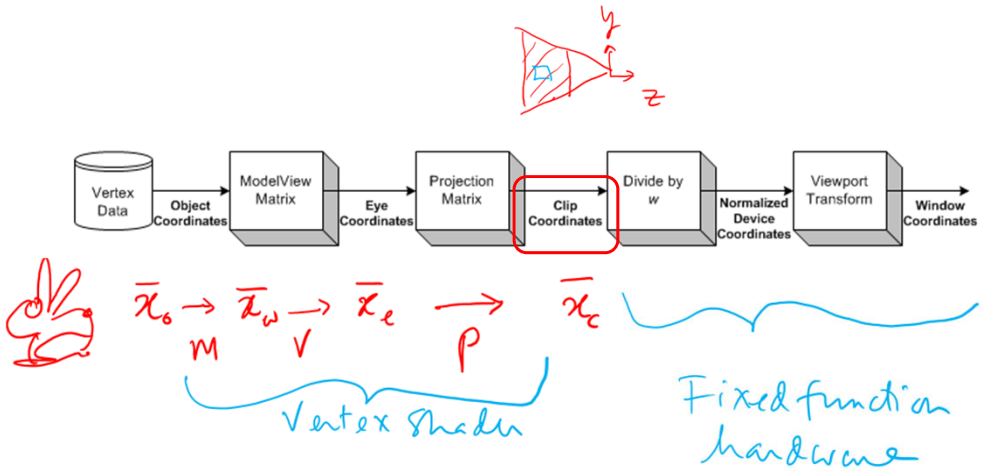
Dinesh K. Pai

Some remaining Vertex Shader topics

Preliminaries

- Today
  - Projective Transformations wrap up
  - Additional Vertex to Pixel steps: Clipping, Normalized Device Coordinates

Review: OpenGL pipeline



http://www.songho.ca/opengl/gl_transform.html

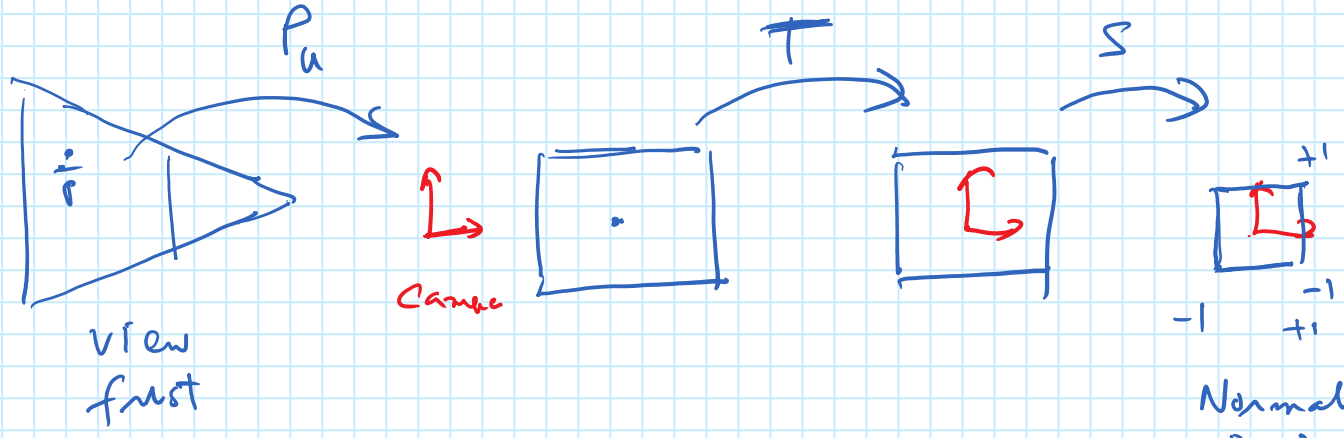
3

Some remaining questions

- What is the general form of the projective transformation?
- What exactly is clipping? Why are we doing this before dividing by w?
- What is the viewport transformation?

4

February 9, 2022 10:11 AM



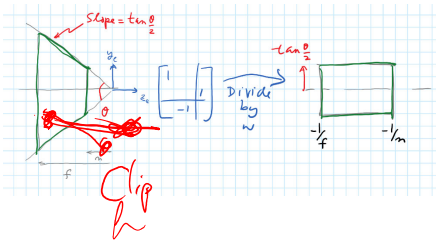
Projective Transform $P = S\,T\,P_u$

Visualzing Transforms, again

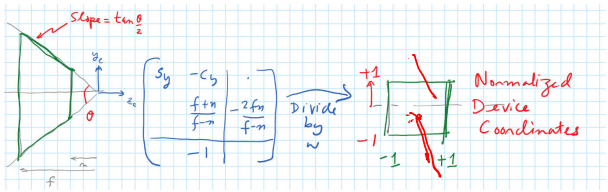
- https://www.realtimerendering.com/udacity/transforms.html

General Projective Transformation

- With the basic projective transformation, view frustum turns into a box whose size depends on the view frustum



- We can scale and shift that box to make it a standard size, with each coordinate from -1 to +1



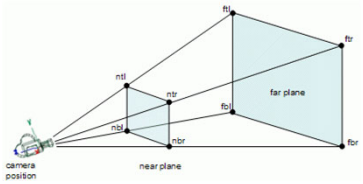
- See Chapter 10.3 of text

7

Normalized projective transformation

- Eye coordinates (projected) → clip coordinates → normalized device coordinates (NDCs)
- Dividing clip coordinates (x_c, y_c, z_c, w_c) by the $w_c (w_c = w_n)$ component (the fourth component in the homogeneous coordinates) yields normalized device coordinates (NDCs).

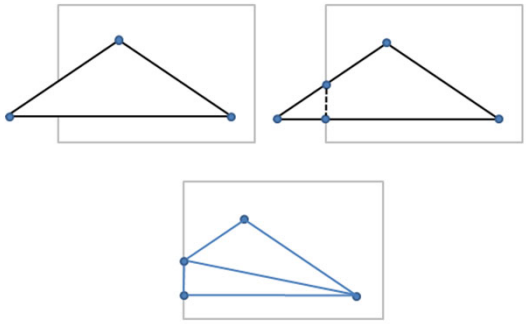
$$\begin{bmatrix} x_n w_n \\ y_n w_n \\ z_n w_n \\ w_n \end{bmatrix} = \begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} = \begin{bmatrix} s_x & 0 & -c_x & 0 \\ 0 & s_y & -c_y & 0 \\ 0 & 0 & \frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix}$$



8

Clipping

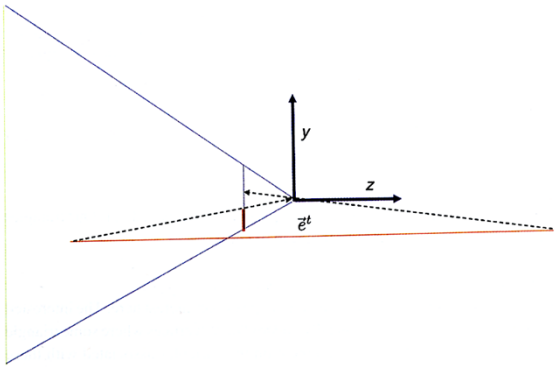
- Primitives totally inside the clipping volume are not altered. Primitives outside the viewing volume are discarded. Primitives whose edges intersect the boundaries of the clipping volume are clipped



9

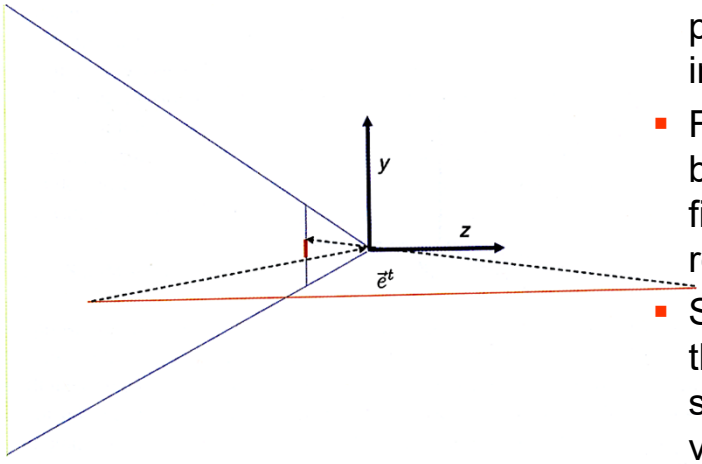
Clipping

- What if there is a vertex behind you?



10

Clipping

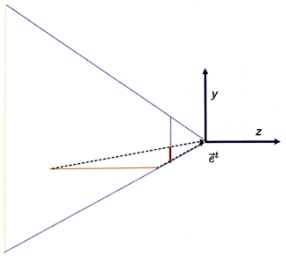


- Back vertex projects higher up in the image
- Filling in the in-between pixels will fill in the wrong region.
- Solution: slice up the geometry by the six faces of the view frustum

11

Clipping coordinates

- If you wait for normalized device coordinates (NDCs) where the vertex has flipped, and it's too late to do the clipping.
- The solution is to use clip coordinates: post-matrix-multiply but pre-divide.
- No divide = no flipping

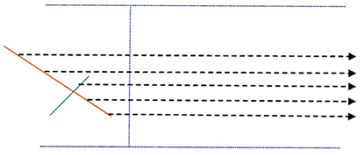


12

Clipping coordinates

- We want points in the NDC box, i.e.,

$$-1 < x_n < 1$$
$$-1 < y_n < 1$$
$$-1 < z_n < 1$$

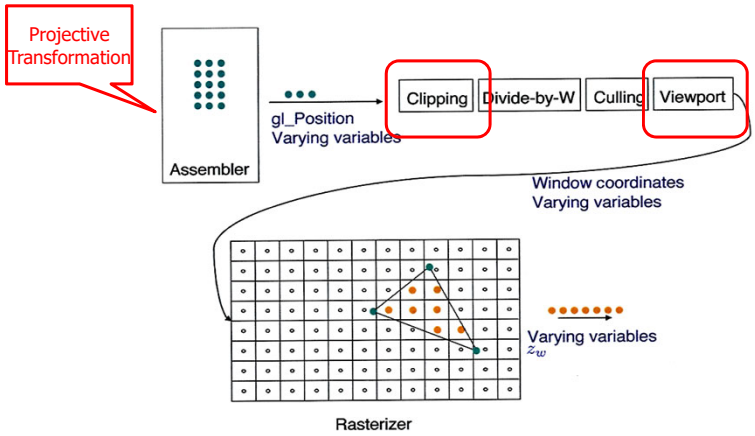


- In clip coordinates this is:

$$\begin{bmatrix} x_n w_n \\ y_n w_n \\ z_n w_n \\ w_n \end{bmatrix} = \begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} \qquad \begin{aligned} -w_c < x_c < w_c \\ -w_c < y_c < w_c \\ -w_c < z_c < w_c \end{aligned}$$

13

Path from vertex to pixel



14

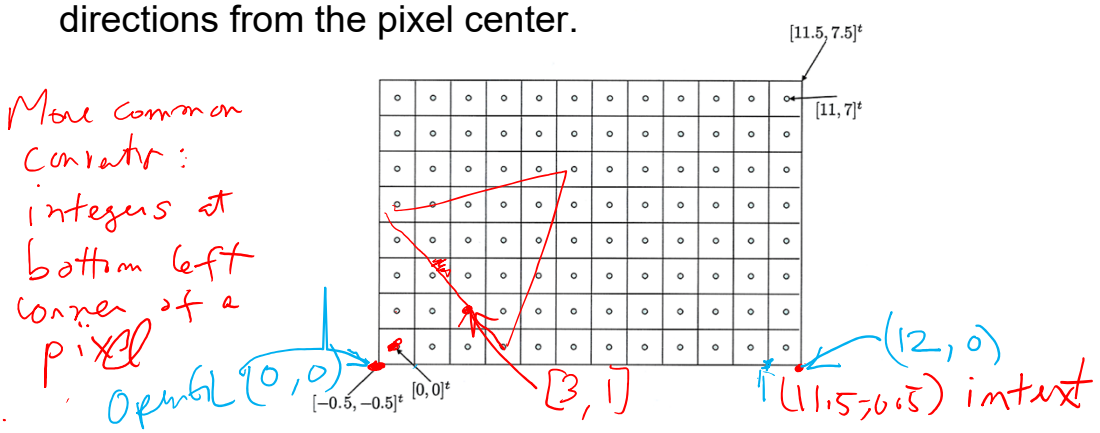
Viewport

- Now we want to position the vertices in the window. So it is time to move the NDCs to window coordinates.
- Each pixel center has an integer coordinate.
  - This will make subsequent pixel computations more natural.
- We want the lower left pixel center to have 2D window coordinates of $[0,0]^t$ and the upper right pixel center to have coordinates $[W-1, H-1]^t$

15

Viewport

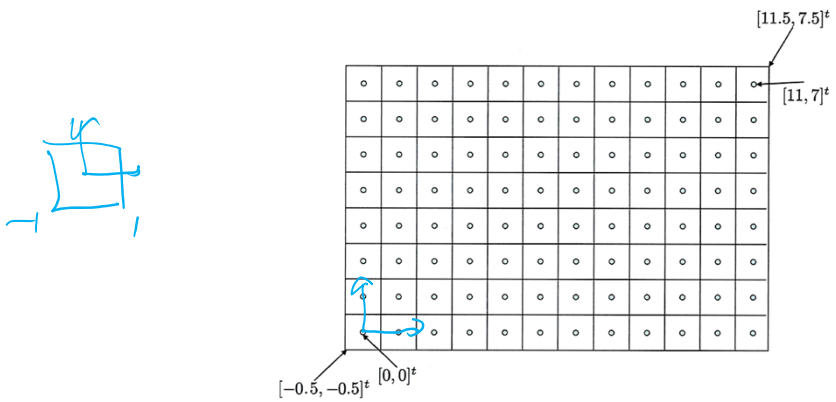
- We think of each pixel as owning the real estate which extends 0.5 pixel units in the positive and negative, horizontal and vertical directions from the pixel center.



16

Viewport

- Thus the extent of 2D window rectangle covered by the union of all our pixels is the rectangle in window coordinates with lower left corner $[-0.5,-0.5]^t$ and upper right corner $[W-0.5,H-0.5]^t$



17

Viewport matrix

- We need a transform that maps the lower left corner to $[-0.5,-0.5]^t$ and upper right corner to $[W-0.5,H-0.5]^t$
- The appropriate scale and shift can be done using the viewport matrix:

$$\begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = \begin{bmatrix} W/2 & 0 & 0 & (W-1)/2 \\ 0 & H/2 & 0 & (H-1)/2 \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \\ z_n \\ 1 \end{bmatrix}$$

18

Next Class

- Basic Rendering
  - Review Chapter 14