CPSC 314

REVIEW

Alla Sheffer
2016

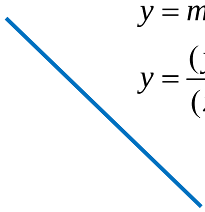
LINES AND CURVES

- Explicit - one coordinate as function of the others

$$y = f(x)$$
$$z = f(x, y)$$

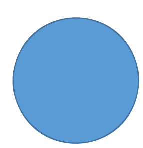
Line

$$y = mx + b$$
$$y = \frac{(y_2 - y_1)}{(x_2 - x_1)}(x - x_1) + y_1$$



Circle

$$y = \pm\sqrt{r^2 - x^2}$$

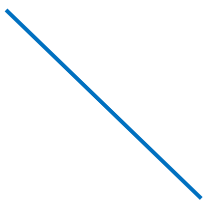


LINES AND CURVES

•Parametric – all coordinates as functions of common parameters

$$(x, y) = (f_1(t), f_2(t))$$
$$(x, y, z) = (f_1(u, v), f_2(u, v), f_3(u, v))$$

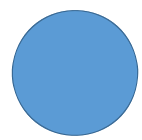
Line



$$x(t) = x_1 + t(x_2 - x_1)$$
$$y(t) = y_1 + t(y_2 - y_1)$$
$$t \in [0,1]$$

Circle

$$x(\theta) = r\cos(\theta)$$
$$y(\theta) = r\sin(\theta)$$
$$\theta \in [0, 2\pi]$$



LINES AND CURVES

•Implicit - define as "zero set" of some function

$$\{(x, y) : F(x, y) = 0\}$$
$$\{(x, y, z) : F(x, y, z) = 0\}$$

- May define meaningful partition of space

$$\{(x, y) : F(x, y) > 0\}, \{(x, y) : F(x, y) = 0\}, \{(x, y) : F(x, y) < 0\}$$

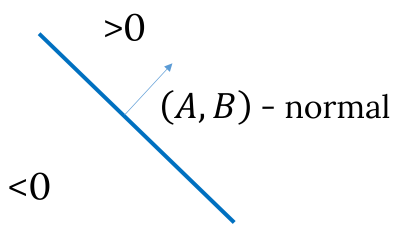
LINES AND CURVES - IMPLICITS

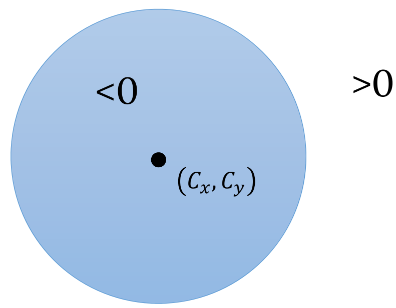
Line

$$Ax + By + C = 0$$

Circle

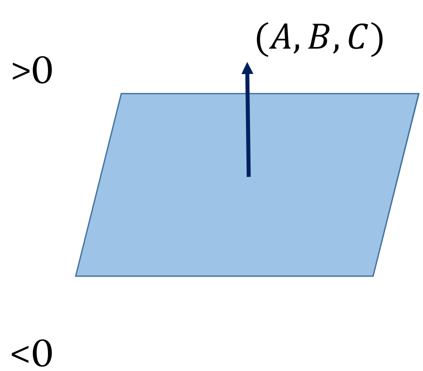
$$(x - C_x)^2 + (y - C_y)^2 - r^2 = 0$$





PLANE - IMPLICIT

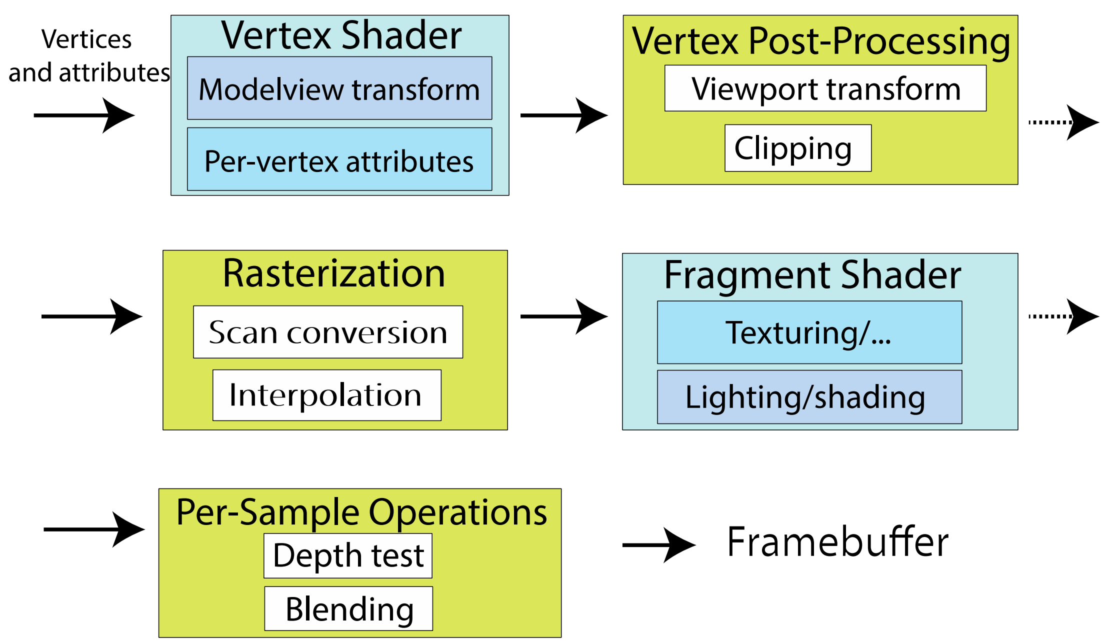
$$Ax + By + Cz + D = 0$$

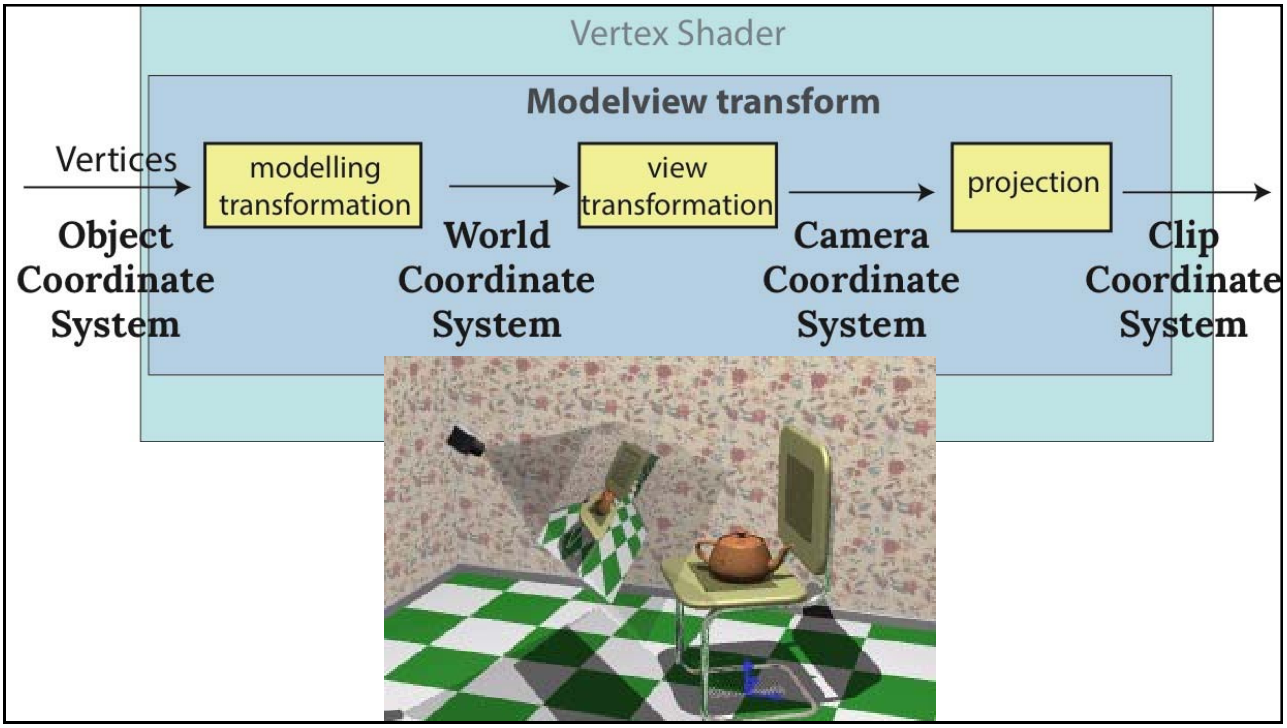


ARBITRARY IMPLICIT FUNCTION

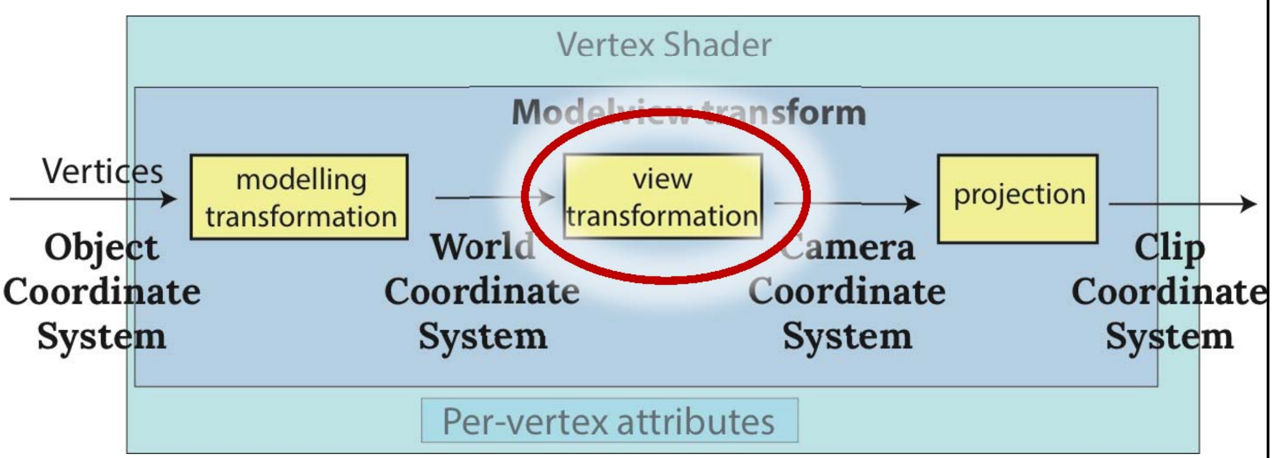
$$F(x, y, z) = 0$$

$$n(x, y, z) = \nabla F(x, y, z) = \begin{pmatrix} \partial F(x, y, z)/\partial x \\ \partial F(x, y, z)/\partial y \\ \partial F(x, y, z)/\partial z \end{pmatrix}$$



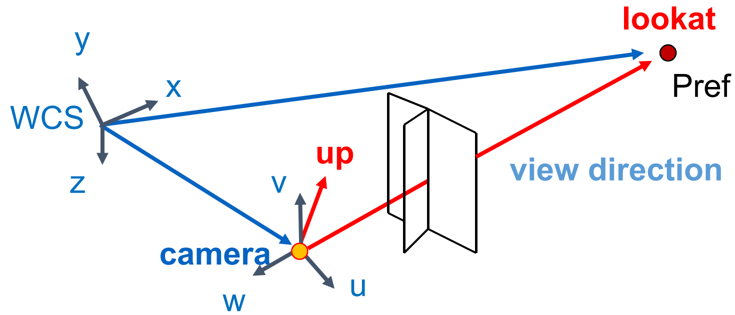


VERTEX SHADER: CLOSER LOOK



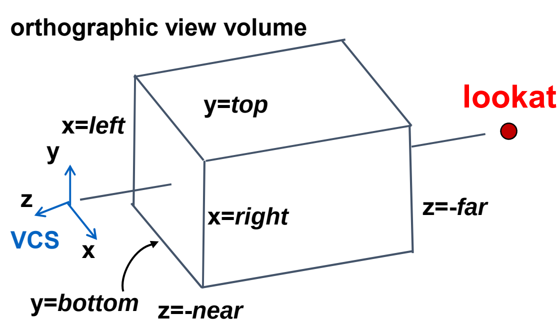
CAMERA COORDINATE SYSTEM

- $w = \frac{p^{eye} - p^{ref}}{\|p^{eye} - p^{ref}\|}$
- $u = \frac{v_{up} \times w}{\|v_{up} \times w\|}$
- $v = w \times u$



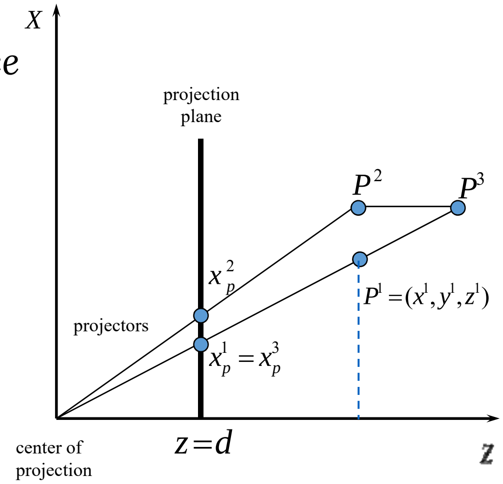
VIEW VOLUME ORTHOGRAPHIC PROJECTION

- specifies field-of-view, used for clipping
- restricts domain of z stored for visibility test



PINHOLE CAMERA (PERSPECTIVE)

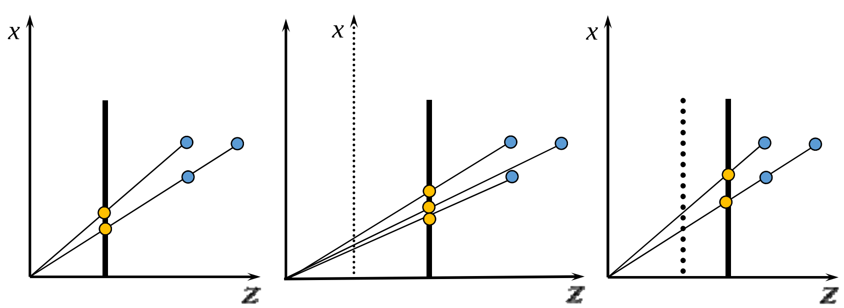
- Viewing from *point at finite distance*
- Without loss of generality:
  - Viewpoint at origin
  - Viewing plane is $z=d$
- Given P=(x,y,z) triangle similarity gives:



$$\frac{x}{z}=\frac{x_p}{d} \text{ and } \frac{y}{z}=\frac{y_p}{d} \Rightarrow x_p=\frac{xd}{z} \text{ and } y_p=\frac{yd}{z}$$

PERSPECTIVE PROJECTION (CONT'D)

- What is (if any) is the difference between:
  - Moving projection plane
  - Moving viewpoint (center of projection)?



PERSPECTIVE PROJECTION

$$P(x, y, z, 1) = \begin{pmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} xd \\ yd \\ z \\ z \end{pmatrix}$$

$$\left(\frac{xd}{z}, \frac{yd}{z}, \frac{z}{z} \right) = (z_p, y_p, 1).$$

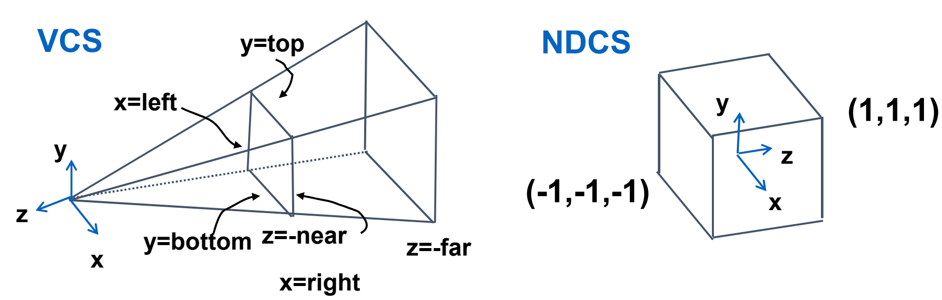
- Keeping track of z:

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

  - NEW Z is monotonic increasing function of old

OPENGL PERSPECTIVE DERIVATION

Map FRUSTUM to the NDCS cube.
Now we need to scale/translate/shear -> generic transform



PERSPECTIVE PROJECTION

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

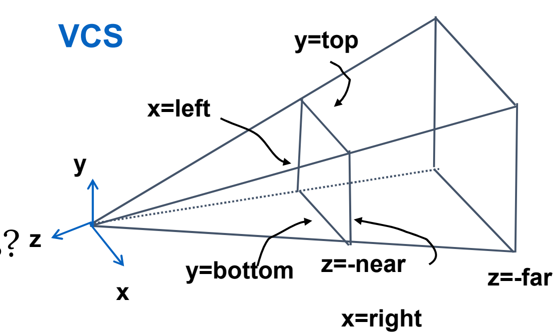
CLIPPING

What's the purpose of clipping?

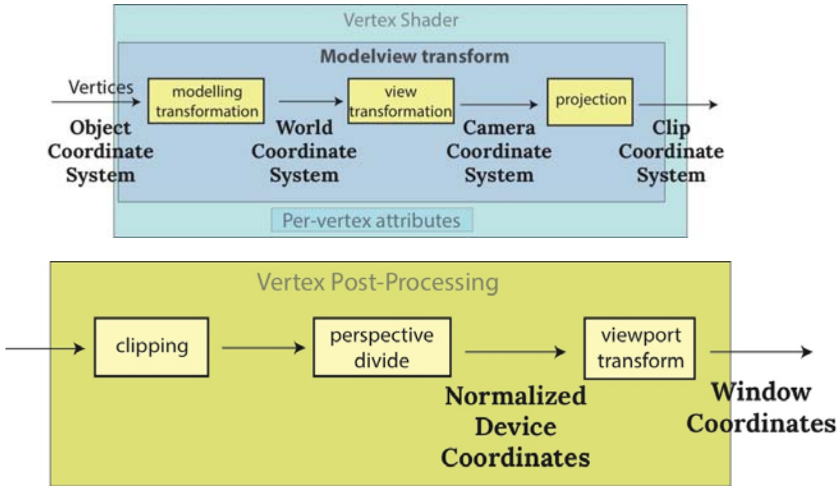
How is it done?

When is it done in pipeline?

Why do we need near/far planes?



PIPELINE EXPANDED



VIEWPORT TRANSFORM

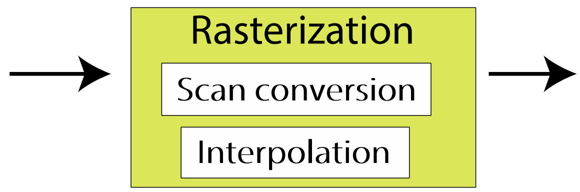
- What does it do?

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

RASTERIZATION

- This is part of the fixed function pipeline

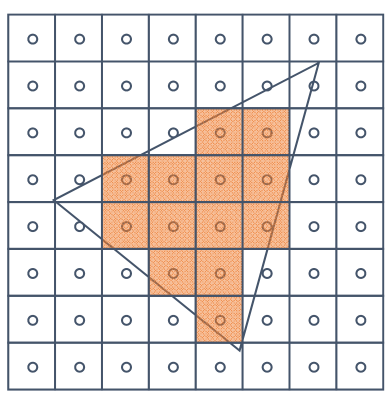
- Input: clipped polygons
- Output: fragments (with **varying variables** interpolated)



SCAN CONVERSION: IDEA

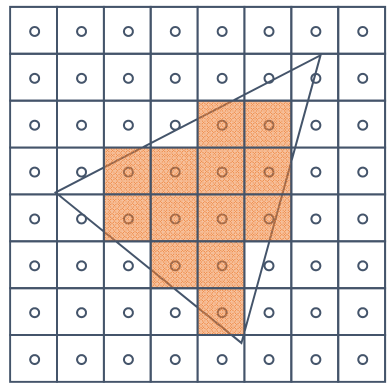
A point is inside ⇔

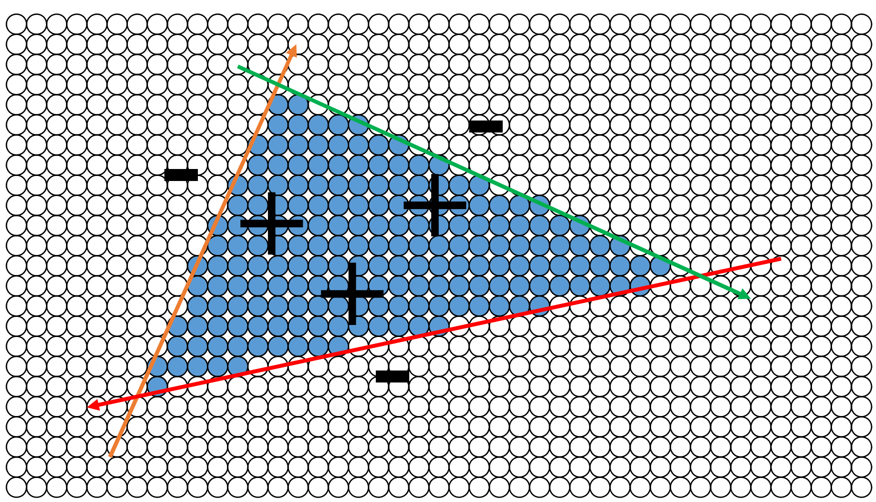
$$A_i x + B_i y + C > 0, i = 1, \ldots, 3$$



HOW TO TREAT BOUNDARY?

- If two triangles share an edge, scan conversion should be consistent
  - No pixel drawn twice
  - No gaps
- E.g. draw left edge, don't draw right one



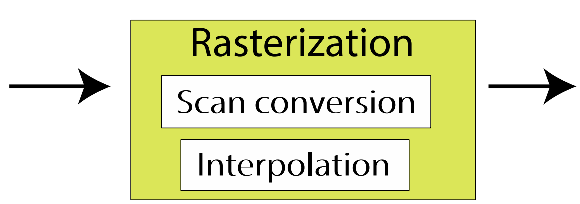


SCAN CONVERSION

- What are problems of scan conversion?
- How to find a bounding box?
- How to scan-convert an arbitrary polygon?

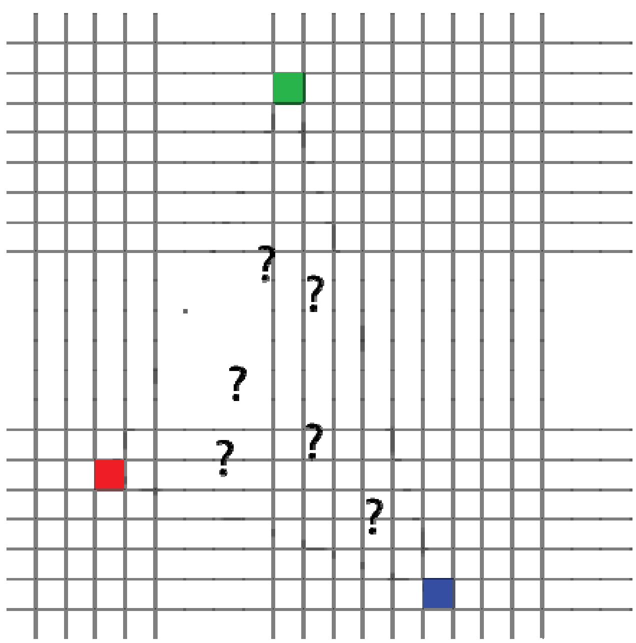
INTERPOLATION

- What does it do?



INTERPOLATION – ACCESS TRIANGLE INTERIOR

- Interpolate between vertices:
  - z
  - r,g,b - colour components
  - u,v - texture coordinates
  - N_x, N_y, N_z - surface normals
- Equivalent
  - Barycentric coordinates
  - Bilinear interpolation
  - Plane Interpolation

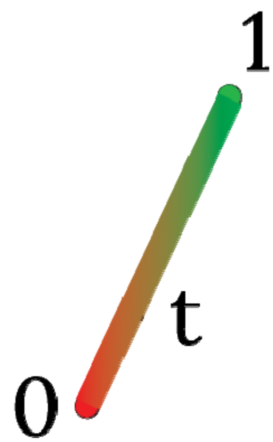


SIMPLER:

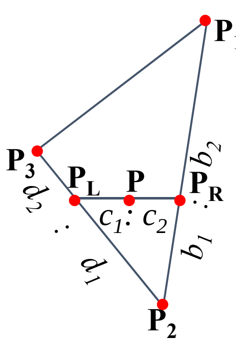
How to interpolate color between two points?

$$c(t) \approx c(0) \cdot (1-t) + c(1) \cdot t$$

Linear interpolation



BI-LINEAR INTERPOLATION



$$P = \frac{c_2}{c_1 + c_2} \cdot P_L + \frac{c_1}{c_1 + c_2} \cdot P_R$$

$$P_L = \frac{d_2}{d_1 + d_2} P_2 + \frac{d_1}{d_1 + d_2} P_3$$

$$P_R = \frac{b_2}{b_1 + b_2} P_2 + \frac{b_1}{b_1 + b_2} P_1$$

$$P = \frac{c_2}{c_1 + c_2} \left(\frac{d_2}{d_1 + d_2} P_2 + \frac{d_1}{d_1 + d_2} P_3 \right) + \frac{c_1}{c_1 + c_2} \left(\frac{b_2}{b_1 + b_2} P_2 + \frac{b_1}{b_1 + b_2} P_1 \right)$$

BARYCENTRIC COORDINATES

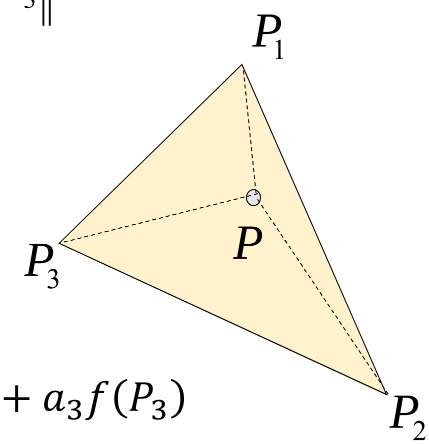
- Area

$$A = \frac{1}{2}\left\|\overrightarrow{P_1P_2} \times \overrightarrow{P_1P_3}\right\|$$

- Barycentric coordinates

$$a_1 = A_{P_2P_3P} / A, a_2 = A_{P_3P_1P} / A,$$
$$a_3 = A_{P_1P_2P} / A,$$
$$P = a_1P_1 + a_2P_2 + a_3P_3$$

$$f(P) \approx a_1 f(P_1) + a_2 f(P_2) + a_3 f(P_3)$$

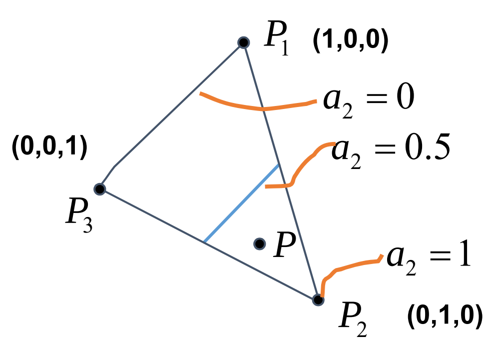


BARYCENTRIC COORDINATES

- weighted (affine) combination of vertices

$$P = a_1 \cdot P_1 + a_2 \cdot P_2 + a_3 \cdot P_3$$

$$a_1 + a_2 + a_3 = 1$$
$$0 \le a_1, a_2, a_3 \le 1$$



BARYCENTRIC COORDINATES

- Positive inside the triangle
- Always sum up to 1
- If we extend barycentric coordinates outside the triangle,

$$A = \frac{1}{2}\left\| \overrightarrow{P_1P_2} \times \overrightarrow{P_1P_3} \right\| \quad \Rightarrow \quad A = \frac{1}{2}(P_1P_2 \times P_1P_3)_z$$

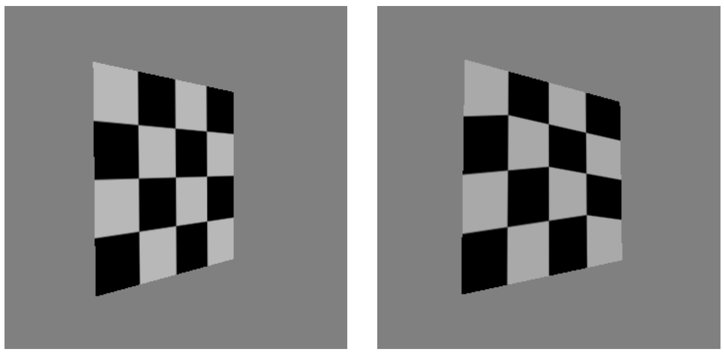
- We have signed areas
- Outside the triangle at least one coordinate < 0

ISSUE WITH INTERPOLATION UNDER PERSPECTIVE PROJECTION

TEXTURE MAPPING

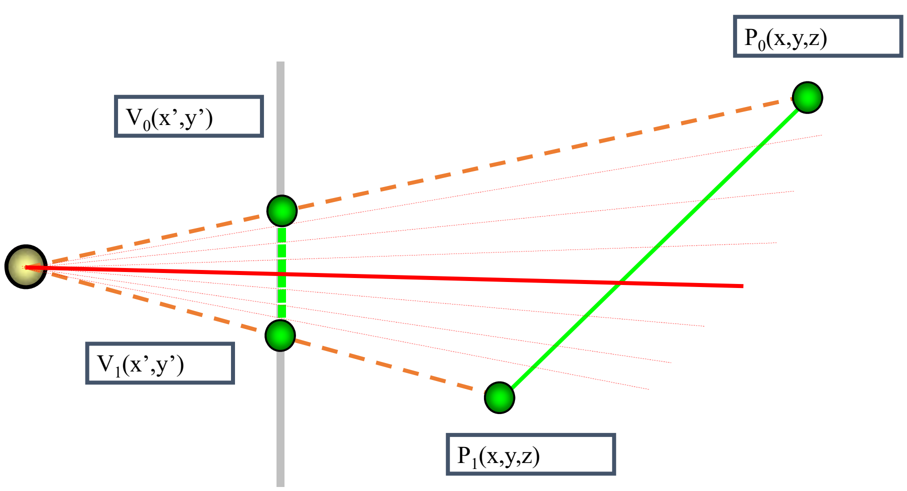
Texture coordinate interpolation

- Perspective foreshortening problem
- Also problematic for color interpolation, etc.



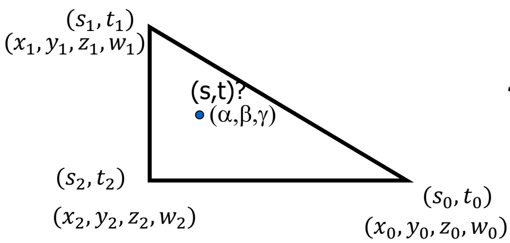
INTERPOLATION: SCREEN VS. WORLD SPACE

- Screen space interpolation incorrect under perspective
  - Problem ignored with shading, but artifacts more visible with texturing



TEXTURE COORDINATE INTERPOLATION

- Perspective Correct Interpolation
  - α, β, γ : Barycentric coordinates (2D) of point **P**
  - s_0, s_1, s_2 : texture coordinates of vertices
  - w_0, w_1, w_2 : homogenous coordinate of vertices



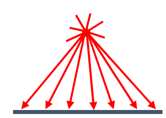
$$s = \frac{\alpha \cdot s_0 / w_0 + \beta \cdot s_1 / w_1 + \gamma \cdot s_2 / w_2}{\alpha / w_0 + \beta / w_1 + \gamma / w_2}$$

- Similarly for t

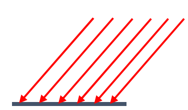
Derivation (similar triangles):
https://www.comp.nus.edu.sg/~lowkl/publications/lowk_persp_interp_techrep.pdf

LIGHT SOURCES

- Point source
  - light originates at a point
  - Rays hit planar surface at different angles



- Parallel source
  - light rays are parallel
  - Rays hit a planar surface at identical angles
  - May be modeled as point source at infinity
  - *Directional light*



LIGHT

- Light has color
- Interacts with object color (r,g,b)

$$I = I_a k_a$$
$$I_a = (I_{ar}, I_{ag}, I_{ab})$$
$$k_a = (k_{ar}, k_{ag}, k_{ab})$$
$$I = (I_r, I_g, I_b) = (I_{ar}k_{ar}, I_{ag}k_{ag}, I_{ab}k_{ab})$$

- Blue light on white surface?
- Blue light on red surface?

DIFFUSE REFLECTION

- Illumination equation is now:

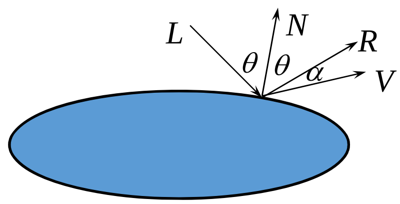
$$I = I_a k_a + I_p k_d (N \cdot L) = I_a k_a + I_p k_d \cos\theta$$

- I_p - point/parallel source's intensity
- k_d - surface diffuse reflection coefficient

- Can we locate light source from shading?

SPECULAR REFLECTION

- Shiny objects (e.g. metallic) reflect light in preferred direction R determined by surface normal N.



- Most objects are not ideal mirrors - reflect in the immediate vicinity of R

ILLUMINATION EQUATION

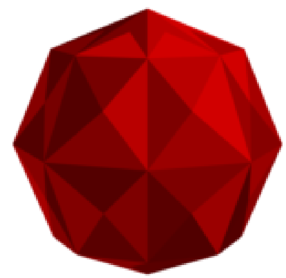
- For multiple light sources:

$$I = I_a k_a + \sum_p \frac{I_p}{d_p{}^2}(k_d(N \cdot L_p) + k_s(R_p \cdot V)^n)$$

- d_p- distance between surface and light source + distance between surface and viewer (Heuristic atmospheric attenuation)

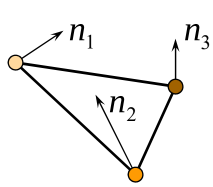
FLAT SHADING

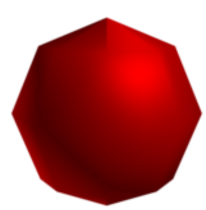
- Illumination value depends only on polygon normal
  - each polygon colored with uniform intensity
- Not adequate for polygons approximating smooth surface
- Looks non-smooth
  - worsened by Mach bands effect



GOURARD SHADING

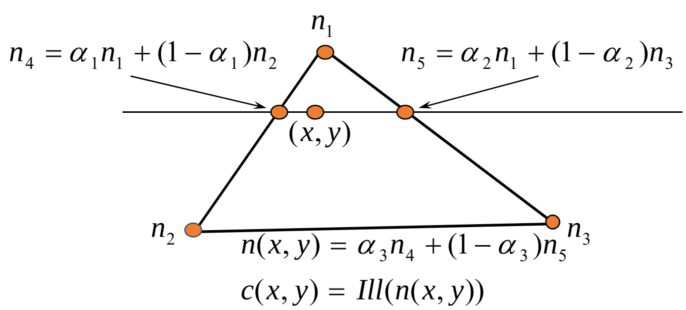
- Polyhedron - approximation of smooth surface
  - Assign to each vertex normal of original surface at point
  - If surface not available use estimate normal
- Compute illumination intensity at vertices using those normals
- Linearly interpolate vertex intensities over interior pixels of polygon projection



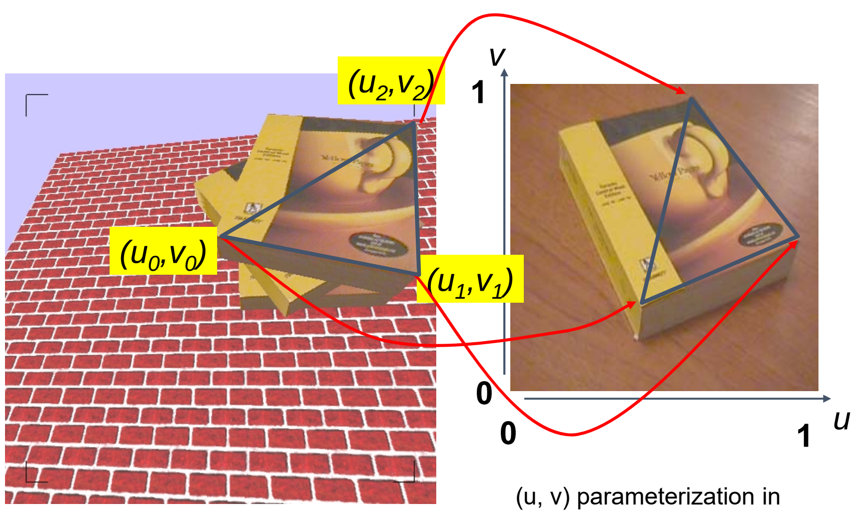


PHONG SHADING

- Interpolate (in image space) normal vectors instead of intensities
- Apply illumination equation for each interior pixel with its own normal



Texture Mapping

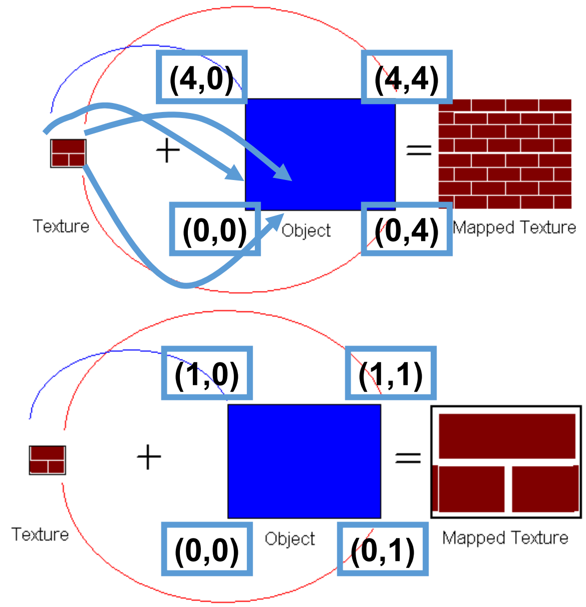


(u, v) parameterization in OpenGL

EXAMPLE TEXTURE MAP

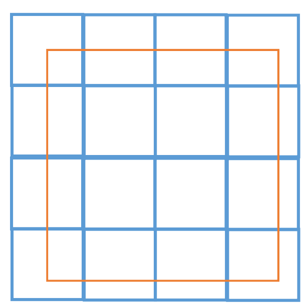
glTexCoord2d(4, 4);
glVertex3d (x, y, z);

glTexCoord2d(1, 1);
glVertex3d (x, y, z);

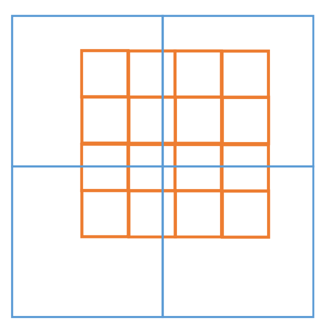


RECONSTRUCTION

- how to deal with:
  - pixels that are much larger than texels?
    - minification

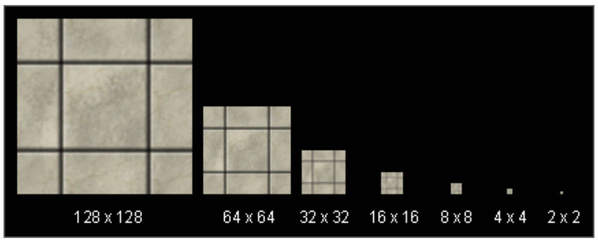


  - pixels that are much smaller than texels ?
    - magnification

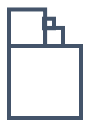


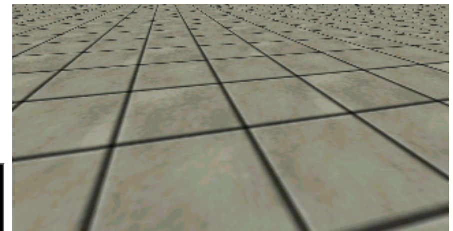
MIPMAPPING

use “image pyramid” to precompute averaged versions of the texture

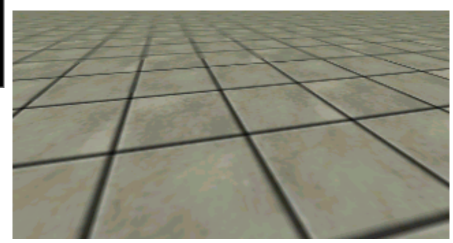


store whole pyramid in single block of memory





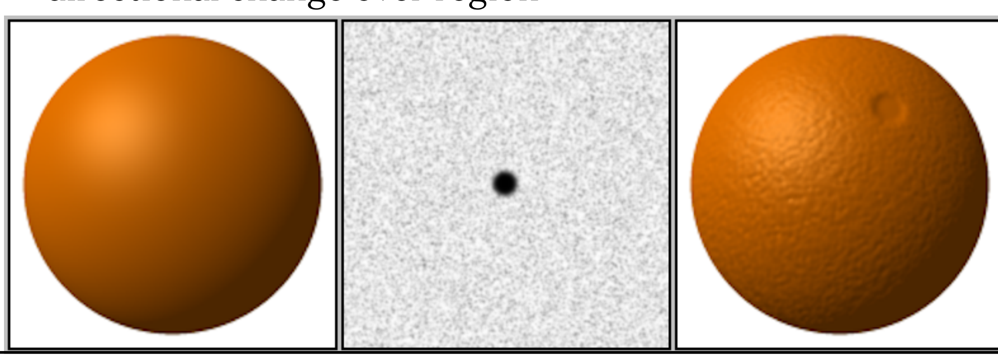
Without MIP-mapping



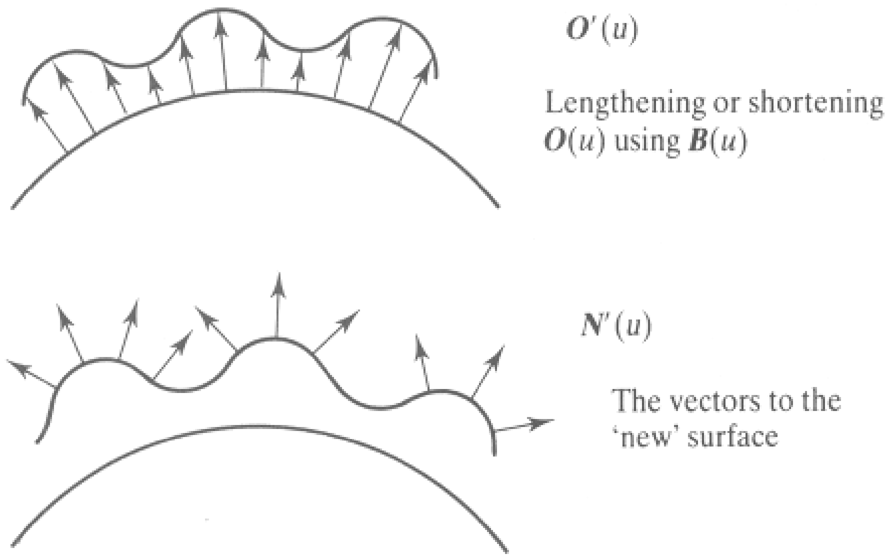
With MIP-mapping

BUMP MAPPING: NORMALS AS TEXTURE

- object surface often not smooth – to recreate correctly need complex geometry model
- can control shape “effect” by locally perturbing surface normal
  - random perturbation
  - directional change over region

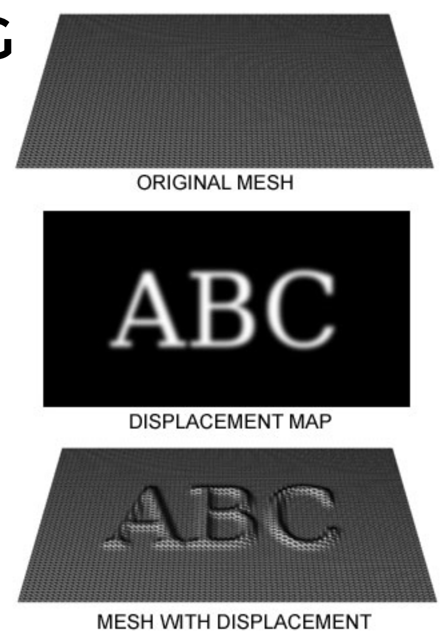


BUMP MAPPING



DISPLACEMENT MAPPING

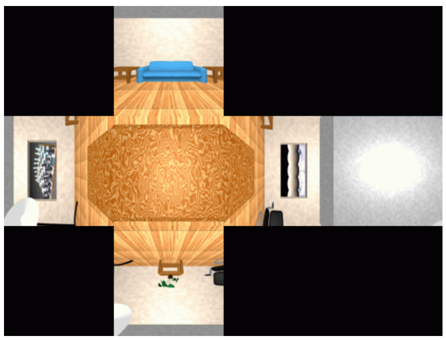
- bump mapping gets silhouettes wrong
  - shadows wrong too
- change surface geometry instead
  - only recently available with realtime graphics
  - need to subdivide surface



https://en.wikipedia.org/wiki/Displacement_mapping#/media/File:Displacement.jpg

CUBE MAPPING

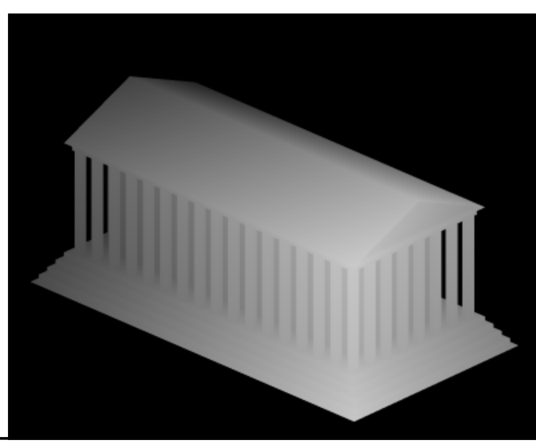
- 6 planar textures, sides of cube
  - point camera in 6 different directions, facing out from origin



SHADOWS

Need at least 2 shader passes:

1. Draw everything as it's viewed from the LIGHT SOURCE
   Depth per pixel ('depth map')

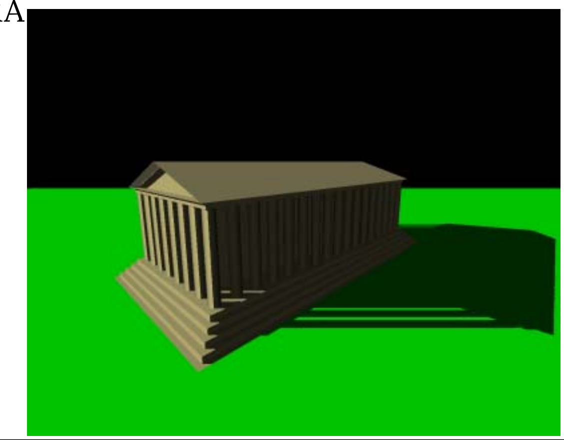


SHADOW MAPS

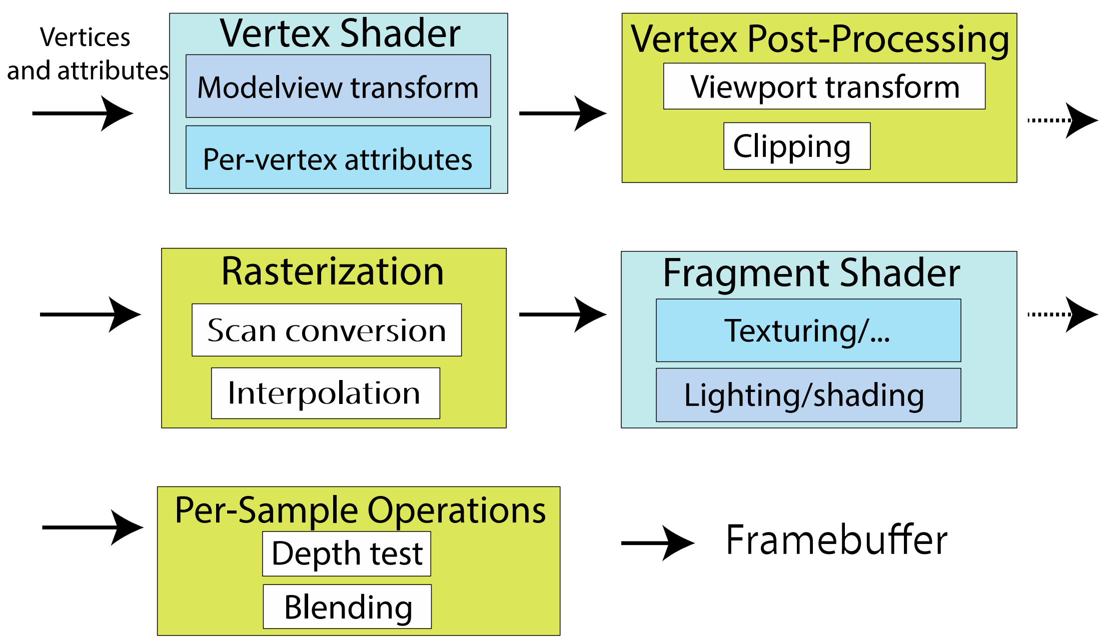
Need at least 2 shader passes:

1. Draw everything as it's viewed from the LIGHT SOURCE
   Depth per pixel ('depth map').
2. Now draw everything from CAMERA
   When computing color per pixel:

   - Find corresponding depth map pixel:
     D - distance from light source
   - Is distance from me to the camera > D?
     - Yes: I am occluded! I'm in SHADOW.
     - No: I'm LIT!



THE RENDERING PIPELINE



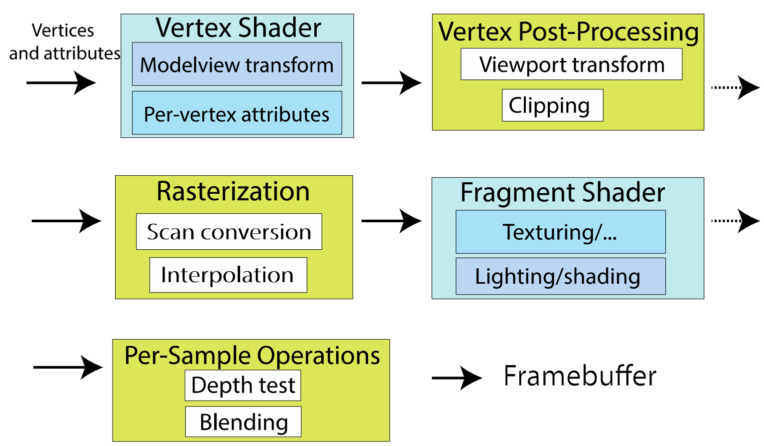
Z-BUFFER

- Store (r,g,b,z) for each pixel
  - typically 8+8+8+24 bits, can be more

```
for all i,j {
 Depth[i,j] = MAX_DEPTH
 Image[i,j] = BACKGROUND_COLOUR
}
for all polygons P {
  for all pixels in P {
    if (Z_pixel < Depth[i,j]) {
      Image[i,j] = C_pixel
      Depth[i,j] = Z_pixel
    }
  }
}
```

DEPTH TEST

- Why is it after FS?



DEPTH TEST PRECISION

- Reminder: projective transformation maps eye-space z to generic z-range (NDC)
- Simple example:

$$T\left(\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

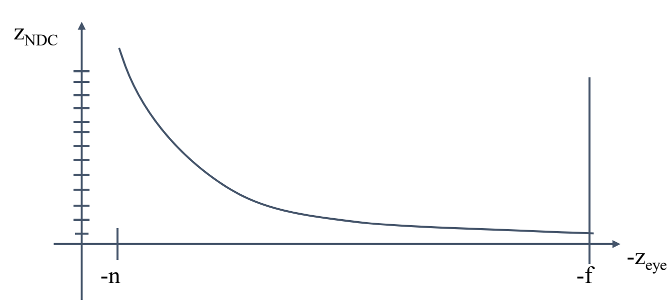
- Thus:

$$z_{NDC} = \frac{a z_{eye} + b}{-z_{eye}} = -a - \frac{b}{z_{eye}}$$

DEPTH TEST PRECISION

Therefore, depth-buffer essentially stores -1/z, rather than z!

- Issue with **integer** depth buffers
  - High precision for near objects
  - Low precision for far objects



DEPTH TEST PRECISION

- Low precision can lead to depth fighting for far objects
  - Two different depths in eye space get mapped to same depth in framebuffer
  - Which object “wins” depends on drawing order and scan-conversion
- Gets worse for larger ratios *f:n*
  - Rule of thumb: f:n < 1000 for 24 bit depth buffer

$$\frac{dz_{NDC}}{dz_{eye}} = \frac{-2fn}{(f-n)z_{eye}^2} = -\frac{2f}{\left(\frac{f}{n}-1\right)z_{eye}^2}$$