Rasterization and Texture Mapping

Lecturer: Erick Fredj

This presentation is largely inspired by Pfister and Chan Computer Graphics Course from MIT 2007.

Reading

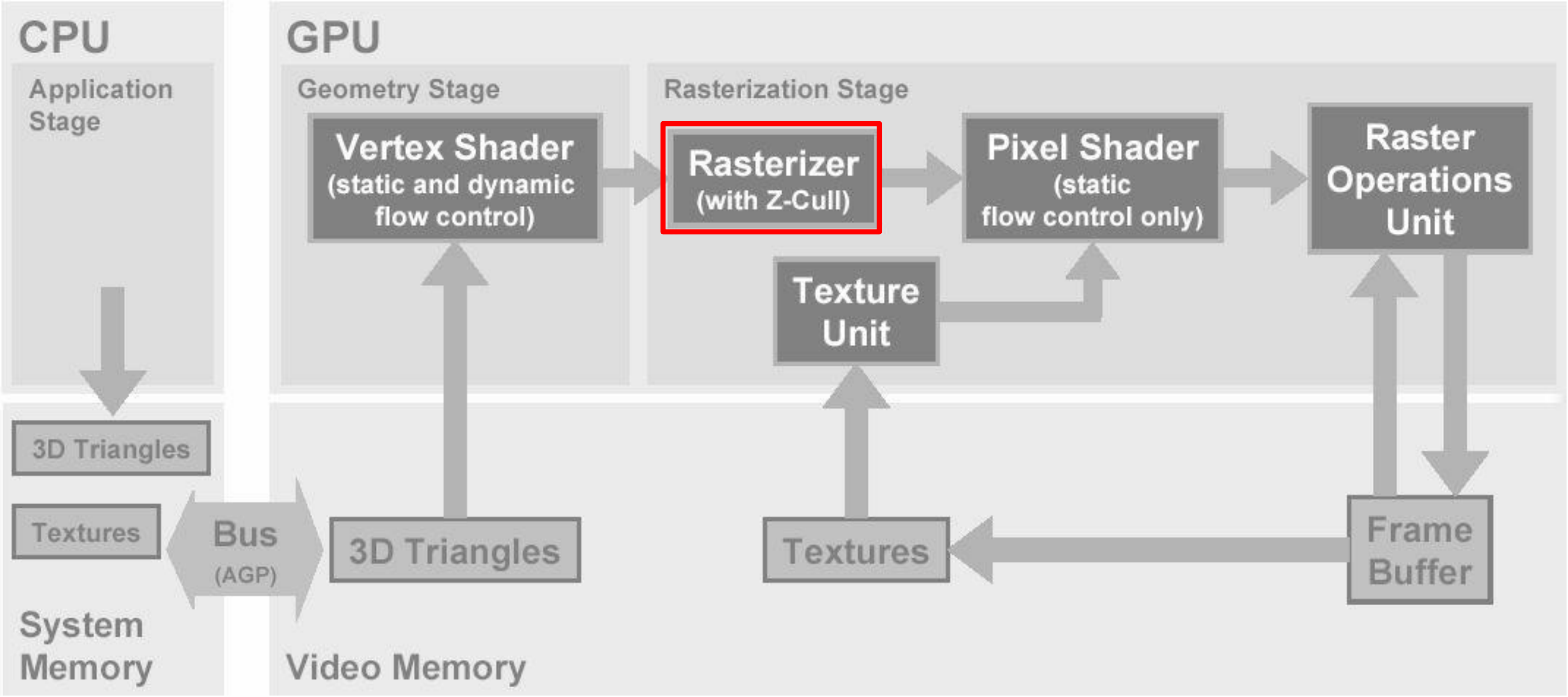
- Hill, Chapter 10

Outline

- Triangle rasterization using barycentric coordinates
- Hidden surface removal (z-buffer)
- Texture mapping

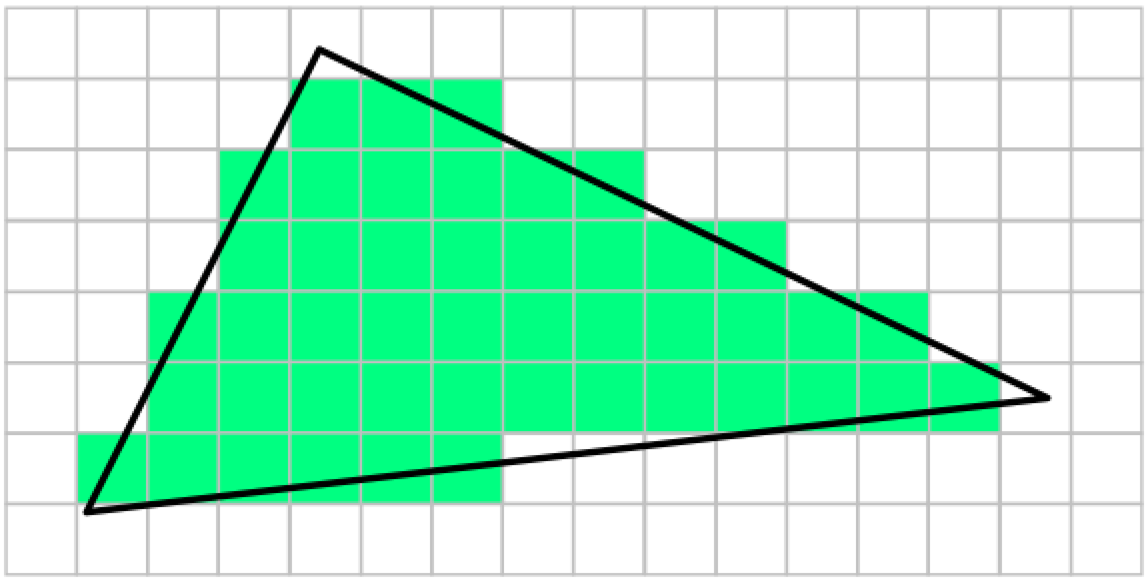
3D Graphics Pipeline

- The rasterization step scan converts the object into pixels



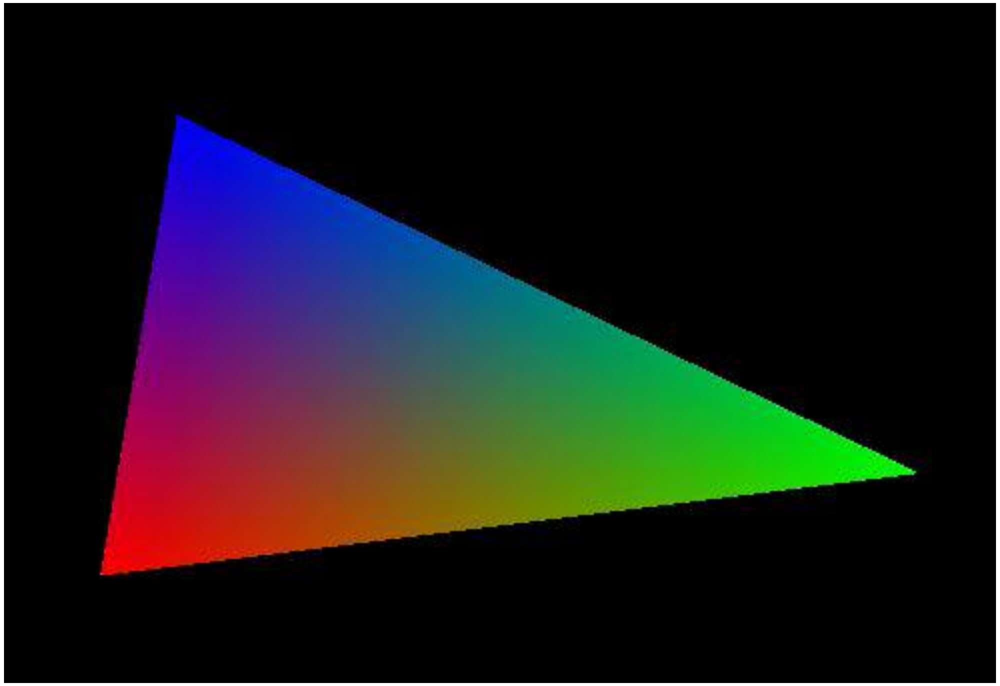
Rasterization (scan conversion)

- Determine which fragments get generated
- Interpolate parameters (colors, texture coordinates, etc.)



Parameter interpolation

- What does “interpolation” mean?
- Example: colors

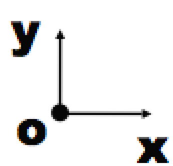


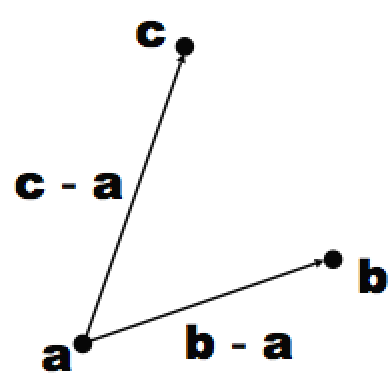
Game Plan

- Theory (using triangles):
  - vector representation of a triangle
  - introduction to barycentric coordinates
  - implicit lines
- Application (still using triangles):
  - rasterization and interpolation using implicit lines and barycentric coordinates

A triangle in terms of vectors

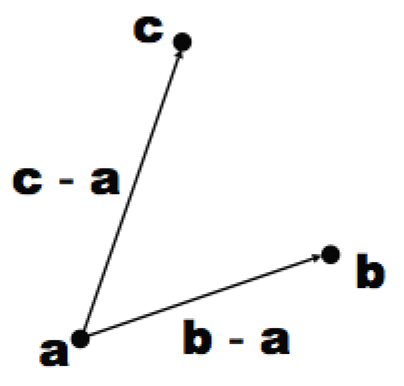
- We can use vertices a, b, and c to specify the three points of a triangle.
- We can also compute the edge vectors.





Points and planes

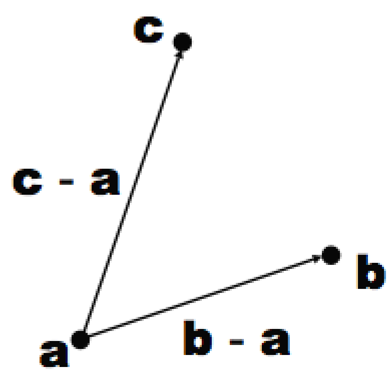
- Three non-collinear points determine a plane



- Example: the vertices a, b, and c determine a plane
- The vectors b-a and c-a form a basis for this plane

Basis vectors

- This (non-orthogonal) basis can be used to specify the location of any point **p in the plane**



$$p = a + \beta(b - a) + \gamma(c - a)$$

Barycentric Coordinates

- We can reorder the terms of the equation:

$$p = a + \beta(b - a) + \gamma(c - a)$$

$$p = (1 - \beta - \gamma)a + \beta b + \gamma c$$

$$p = \alpha a + \beta b + \gamma c$$

- This yields the equation: $p = \alpha a + \beta b + \gamma c$

- with: $1 = \alpha + \beta + \gamma$

- This coordinate system is called barycentric coordinates
  - α, β, γ are the "coordinates"

Barycentric Coordinates

- Barycentric coordinates describe a point p as an affine combination of the triangle vertices

$$p = \alpha a + \beta b + \gamma c \qquad 1 = \alpha + \beta + \gamma$$

- For any point P inside the triangle (a, b, c):

$$0 < \alpha < 1$$

$$0 < \beta < 1$$

$$0 < \gamma < 1$$

- Point on an edge: one coefficient is 0
- Vertex: two coefficients are 0, remaining one is 1

Recap so far

- We need to interpolate parameters during rasterization
- If a triangle is defined by (a, b, c) then any point p inside can be written as

$$p = \alpha a + \beta b + \gamma c$$

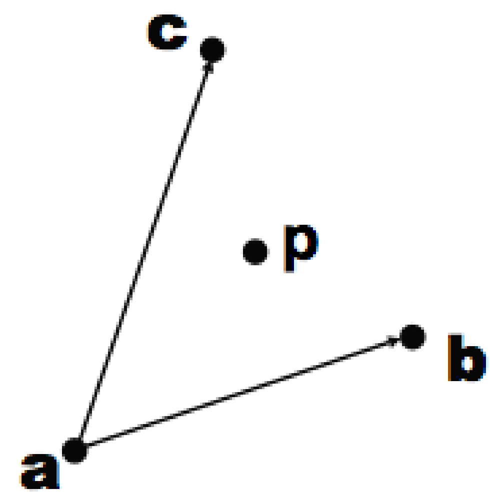
- with these properties:

$$0 < \alpha < 1$$

$$0 < \beta < 1$$

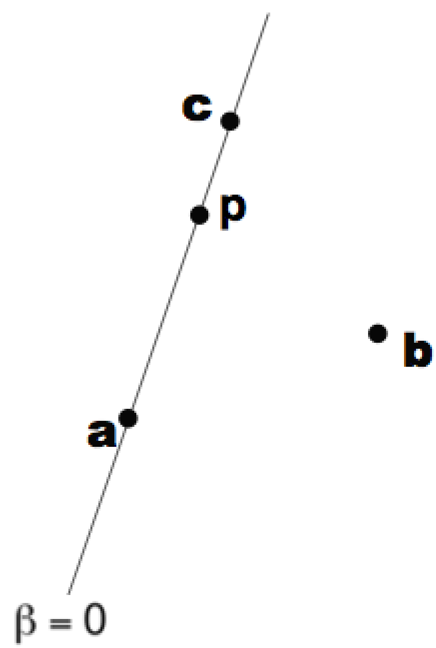
$$0 < \gamma < 1$$

$$1 = \alpha + \beta + \gamma$$



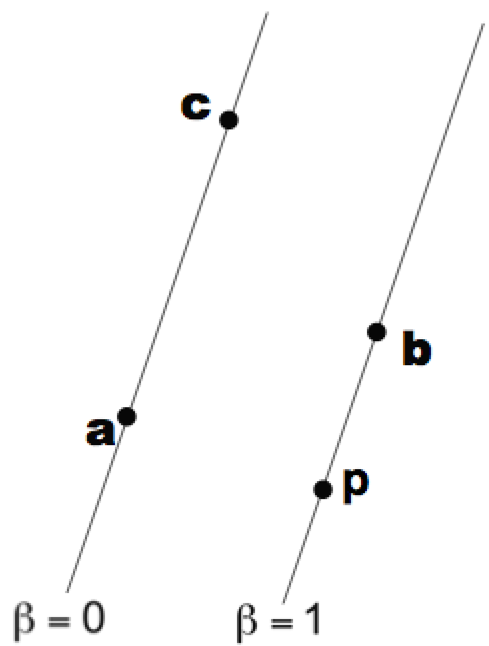
Signed distances

- Let **p = αa + βb + γc. Each coordinate (e.g. β) is the** signed distance from **p to the line through a triangle** edge (e.g. **ac)**



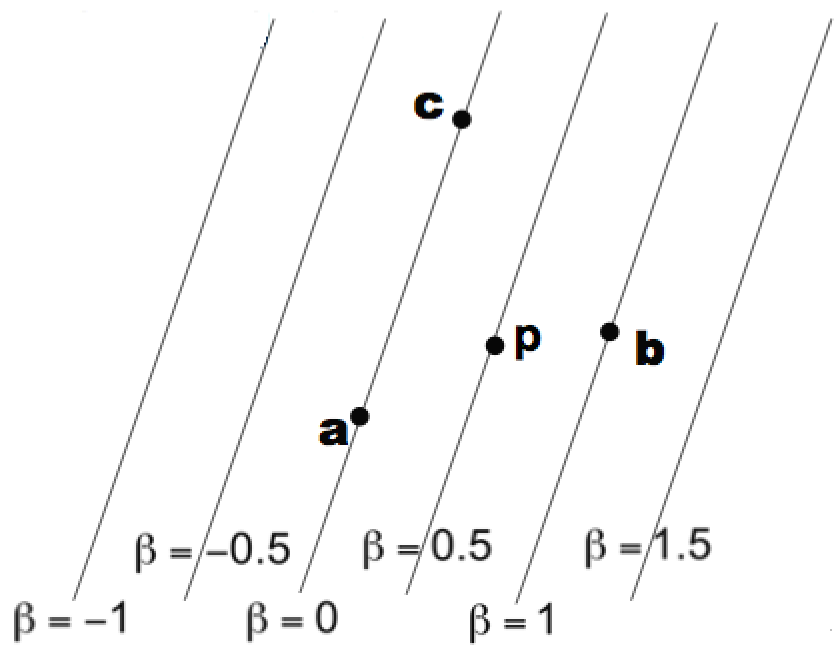
Signed distances

- Let **p = αa + βb + γc. Each coordinate (e.g. β) is the** signed distance from **p to the line through a triangle** edge (e.g. **ac)**



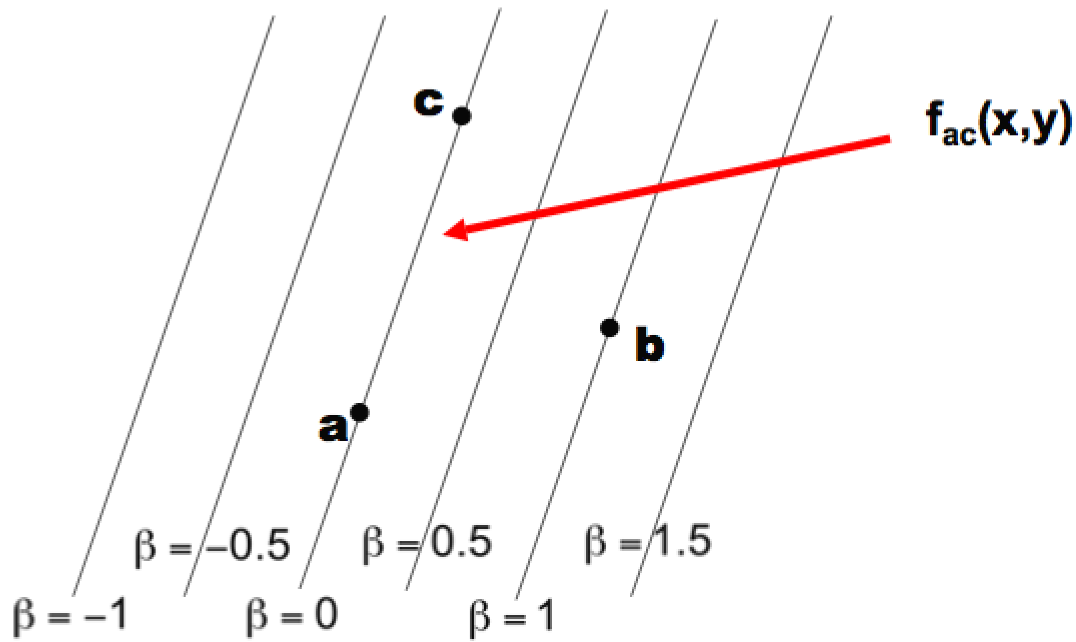
Signed distances

- Let **p = αa + βb + γc. Each coordinate (e.g. β) is the** signed distance from **p to the line through a triangle** edge (e.g. **ac)**



Signed distances

- The signed distance can be computed by evaluating implicit line equations, e.g., **$f_{ac}(x,y)$ of edge ac**



Implicit Lines

- Implicit equation in two dimensions:

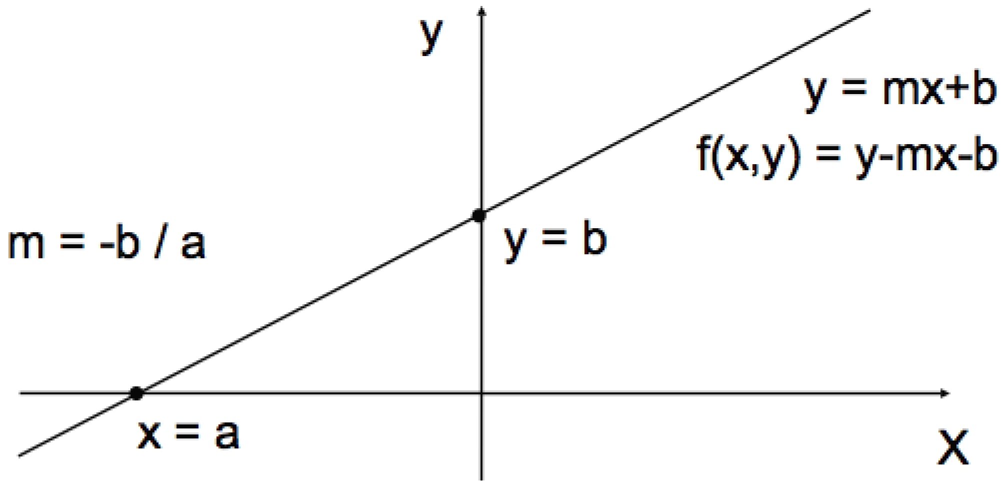
$$f(x,y)=0$$

  - Points with $f(x,y) = 0$ are on the line
  - Points with $f(x,y) \neq 0$ are not on the line

Implicit Lines

- The implicit form of the slope-intercept equation:

$$\boxed{y - mx - b = 0}$$



Implicit Lines

- The slope-intercept form can not represent some lines, such as x = 0.
- A more general implicit form is more useful:

$$Ax+By+C=0$$

- The implic
- it line through two points (x_0,y_0) and (x_1,y_1):

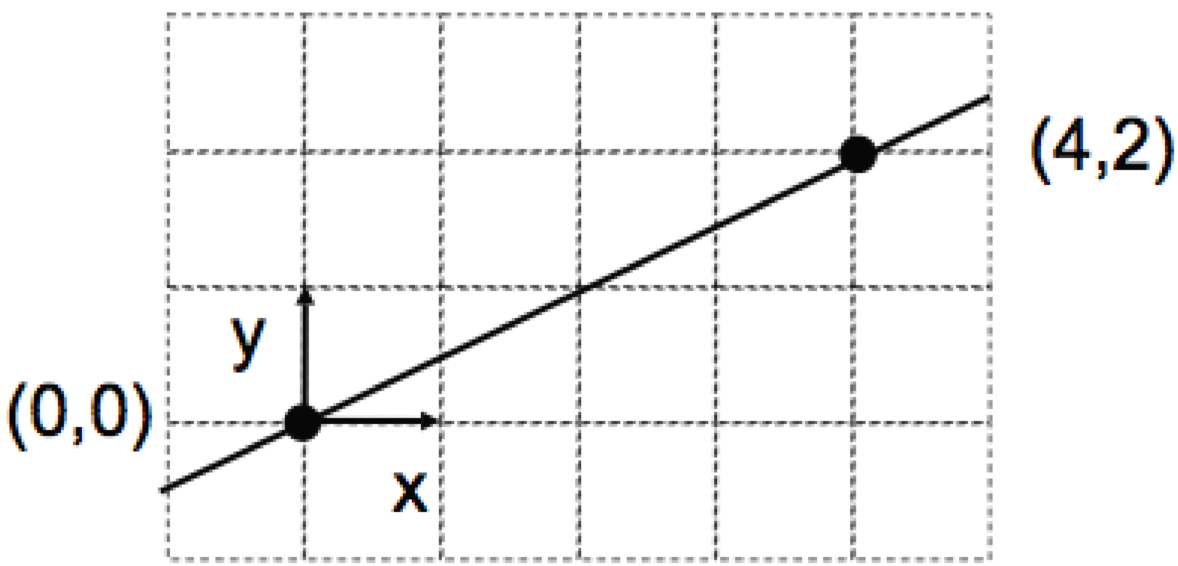
$$(y_0 - y_1)x + (x_1 - x_0)y + x_0 y_1 - x_1 y_0 = 0$$

Example

- What is the implicit equation of this line?

$$(y_0 - y_1)x + (x_1 - x_0)y + x_0 y_1 - x_1 y_0 = 0$$

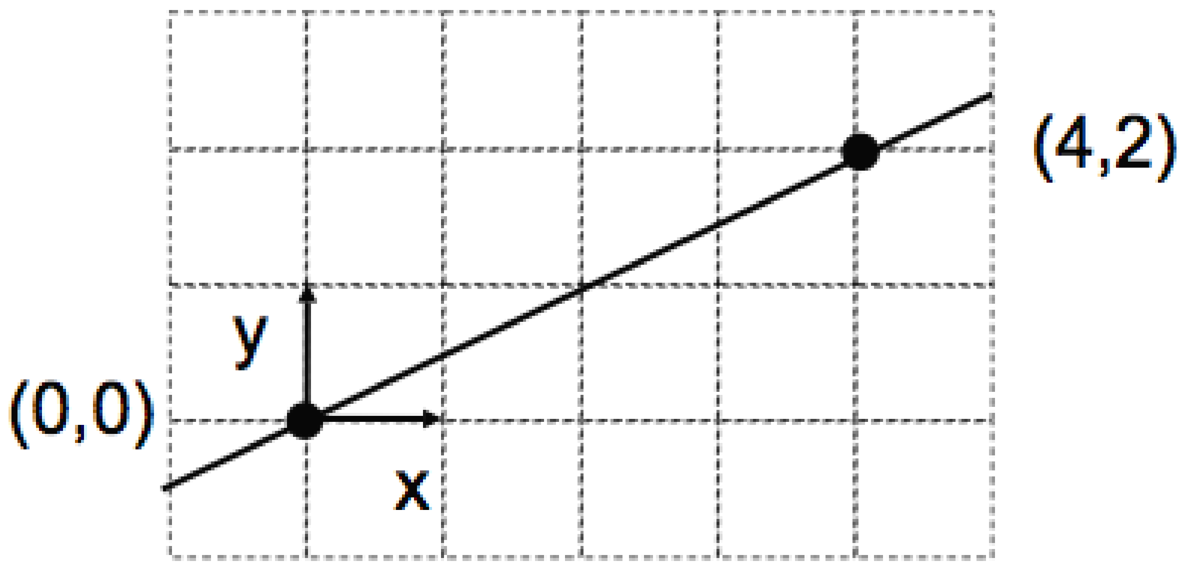
$$\_\,x + \_\,y = 0$$



Example

- Solution 1: -2x + 4y = 0
- Solution 2: 2x - 4y = 0
- What's the lesson here?

k f(x,y) = 0 is the same line, for any value of k

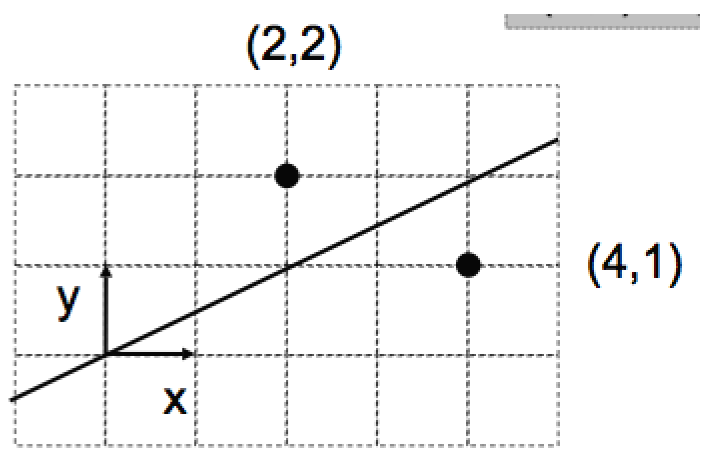


Example

- The value of $f(x,y) = -2x + 4y$ tells us which side of the line a point (x,y) is on

$$f(2,2)=\_$$

$$f(4,1)=\_$$

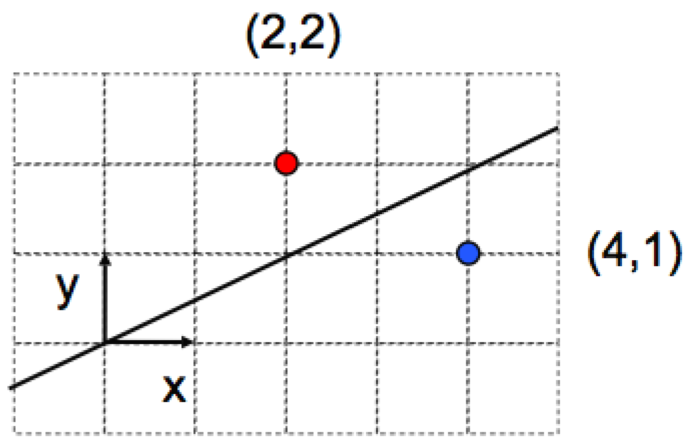


Example

- The value of $f(x,y) = -2x + 4y$ tells us which side of the line a point (x,y) is on

$f(2,2) = +4$ (+=above)

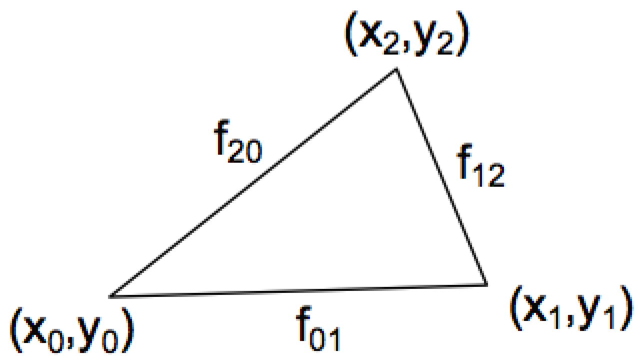
$f(4,1) = -4$ (-=below)



Edge Equations

- Given a triangle with vertices (x_0,y_0), (x_1,y_1), and (x_2,y_2).

$$f_{01}(x,y) = (y_0 - y_1)x + (x_1 - x_0)y + x_0 y_1 - x_1 y_0$$
$$f_{12}(x,y) = (y_1 - y_2)x + (x_2 - x_1)y + x_1 y_2 - x_2 y_1$$
$$f_{20}(x,y) = (y_2 - y_0)x + (x_0 - x_2)y + x_2 y_0 - x_0 y_2$$



Barycentric Coordinates

- Remember that: $f(x,y)=0 \leftrightarrow kf(x,y)=0$

Barycentric Coordinates

- Remember that: $f(x,y)=0 \leftrightarrow kf(x,y)=0$
- A barycentric coordinate (e.g. β) is a signed distance from a line (e.g. the line that goes through **ac)**
- For a given point **p, we would like to compute its** barycentric coordinate β using an implicit edge equation.
- We need to choose **k such that $kf_{ac}(x,y)=\beta$**

Barycentric Coordinates

- We would like to choose k such that: $\mathbf{k f_{ac}(x,y)=\beta}$
- We know that $\boldsymbol{\beta}$ = 1 at point b:

$$k f_{ac}\left(x_b, y_b\right) = 1 \Leftrightarrow k = \frac{1}{f_{ac}\left(x_b, y_b\right)}$$

- The barycentric coordinate $\boldsymbol{\beta}$ for point **p is:**

cartesian coordinates of **p**

$$\beta = \frac{f_{ac}(x, y)}{f_{ac}\left(x_b, y_b\right)}$$

cartesian coordinates of **b**

Barycentric Coordinates

- In general, the barycentric coordinates for point **p are:**

cartesian coordinates of **p**

$$\alpha = \frac{f_{bc}(x, y)}{f_{bc}(x_c, y_c)} \qquad \beta = \frac{f_{ac}(x, y)}{f_{ac}(x_b, y_b)}$$

cartesian coordinates of **a**

cartesian coordinates of **b**

- Given a point **p with cartesian coordinates (x,y), we can** compute its barycentric coordinates (α,β,γ) as above.

GPU Triangle Rasterization

- Many different ways to generate fragments for a triangle
- Checking ($0<\alpha<1$ && $0<\beta<1$ && $0<\gamma<1$) is one method
- In practice, GPUs use optimized methods:
  - fixed point precision (not floating-point)
  - incremental (use results from previous pixel)

Triangle Rasterization

- We can use barycentric coordinates to rasterize and color triangles

```
for all x do
    for all y do
            compute (alpha, beta, gamma) for (x,y)
            if ( 0 < alpha < 1 and
            0 < beta < 1 and
            0 < gamma < 1 ) then
            c = alpha c0 + beta c1 + gamma c2
            drawpixel(x,y) with color c
```

- The color c varies smoothly within the triangle
- This is called Gouraud interpolation after its inventor Henri Gouraud (French, born 1944)

Triangle Rasterization

- Instead of looping over the whole image, we can loop over pixels inside the bounding rectangle of the triangle

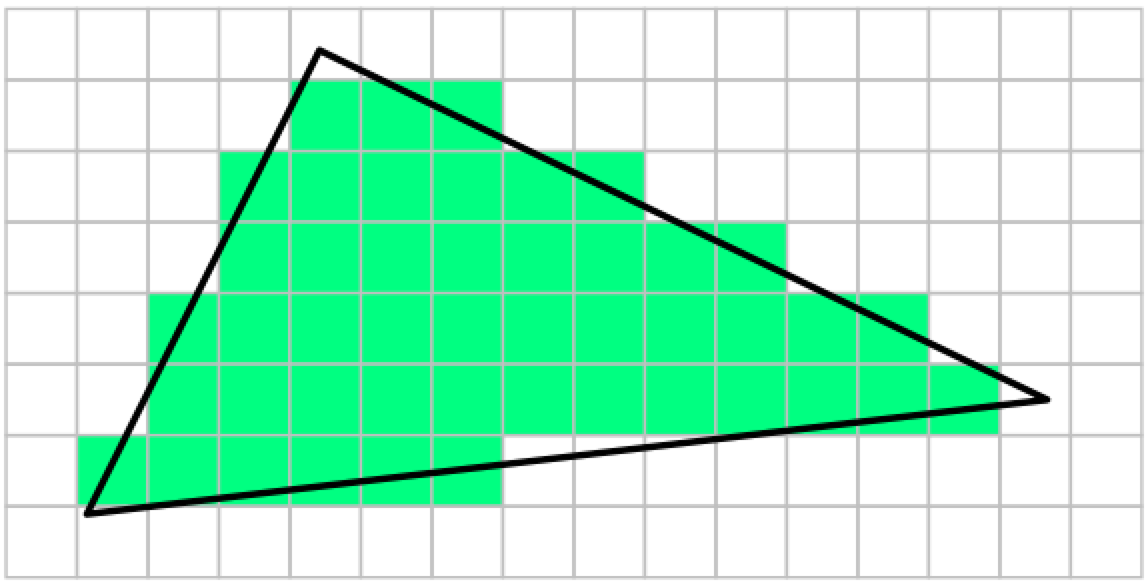
```
xmin = floor(xi)
xmax = ceiling(xi)
ymin = floor(yi)
ymax = ceiling(yi)
for all y = ymin to ymax do
    for all x = xmin to xmax do
        alpha = f12(x,y) / f12(x0,y0)
        beta = f20(x,y) / f20(x1,y1)
        gamma = f01(x,y) / f01(x2,y2)
        if ( alpha > 0 and beta > 0 and gamma > 0 ) then
        c = alpha c0 + beta c1 + gamma c2
        drawpixel(x,y) with color c
```

Outline

- Triangle rasterization using barycentric coordinates
- Hidden surface removal (z-buffer)
- Texture mapping

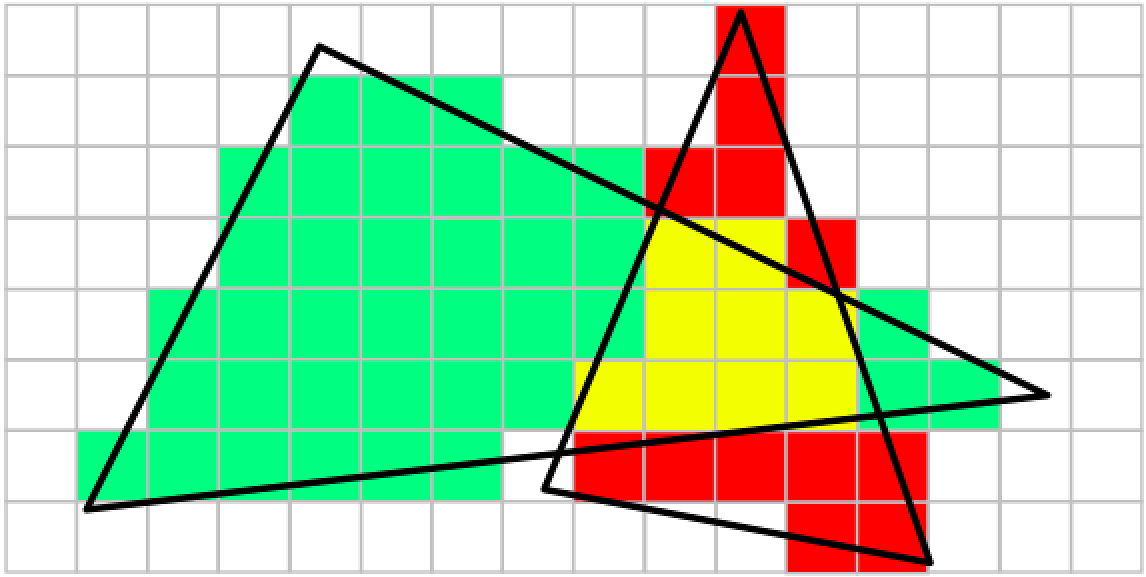
One Triangle

- With one triangle, things are simple
- Fragments never overlap!



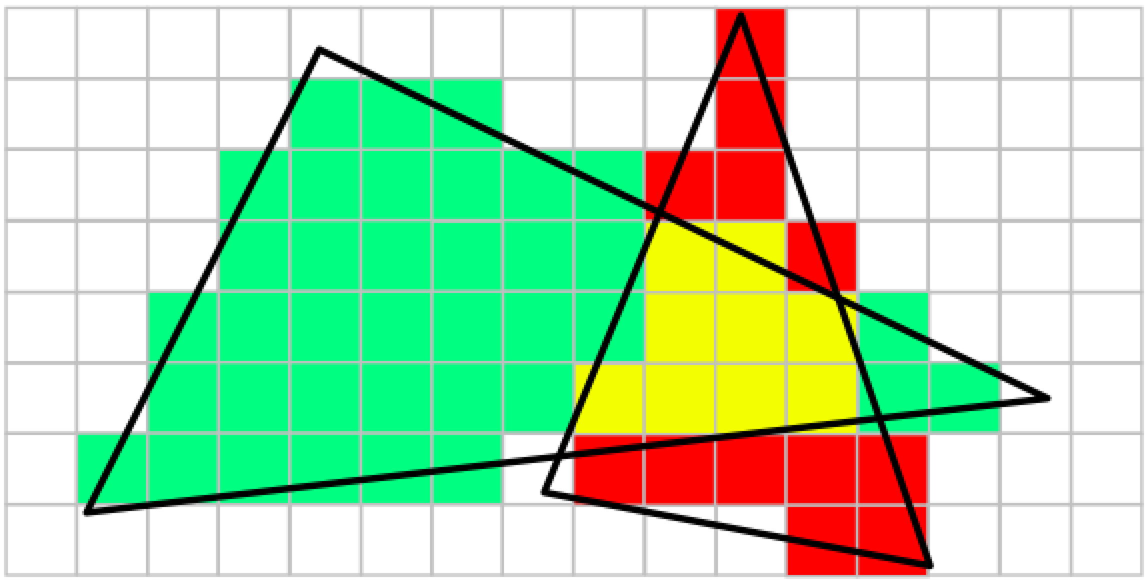
Two Triangles

- Things get more complicated with multiple triangles
- Fragments might overlap in screen space!



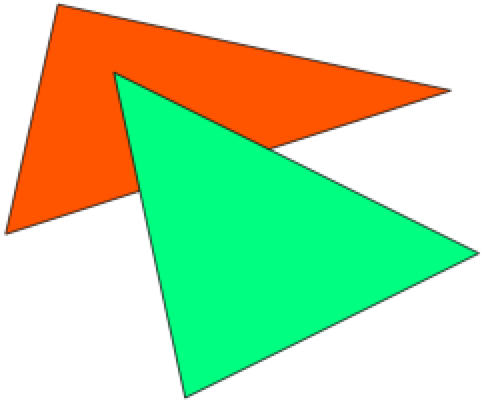
Fragments vs. Pixels

- Each pixel has a unique frame buffer (image) location
- But multiple fragments may end up at same address

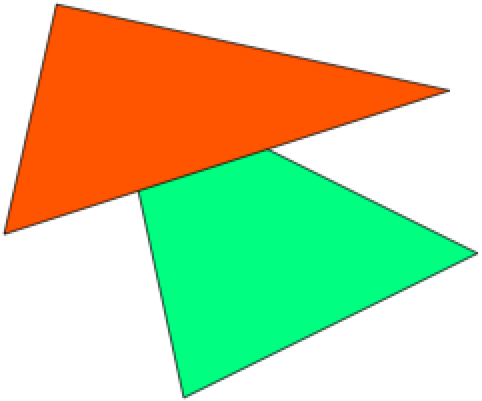


Which triangle wins?

- Two possible cases:



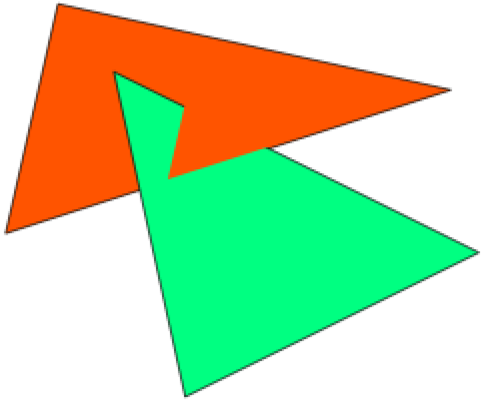
green triangle on top



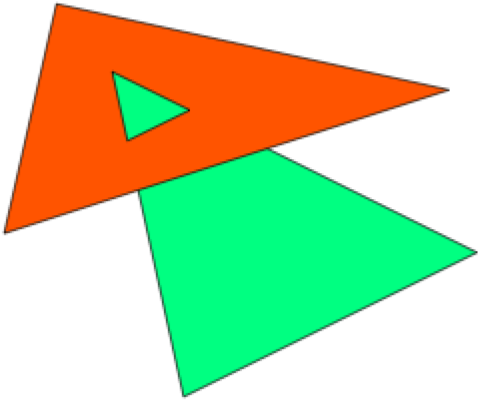
orange triangle on top

Which (partial) triangle wins?

- Many other cases possible!



intersection #1



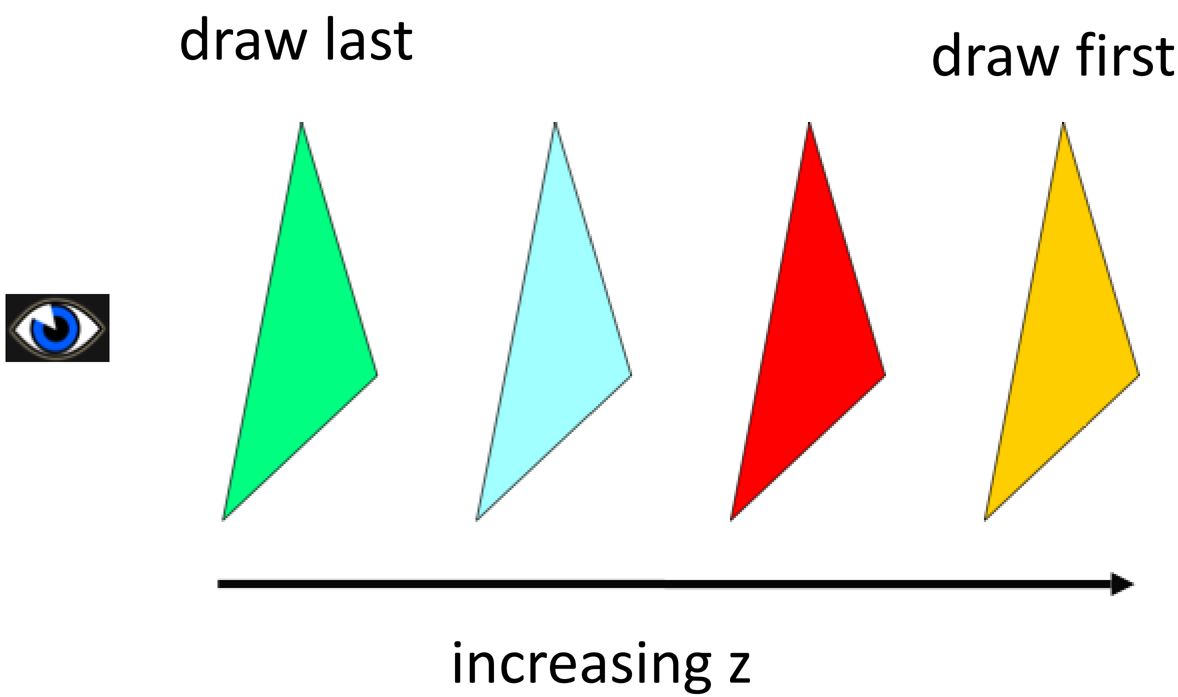
intersection #2

Hidden Surface Removal

- Idea: keep track of visible surfaces
- Typically, we see only the front-most surface
- Exception: transparency

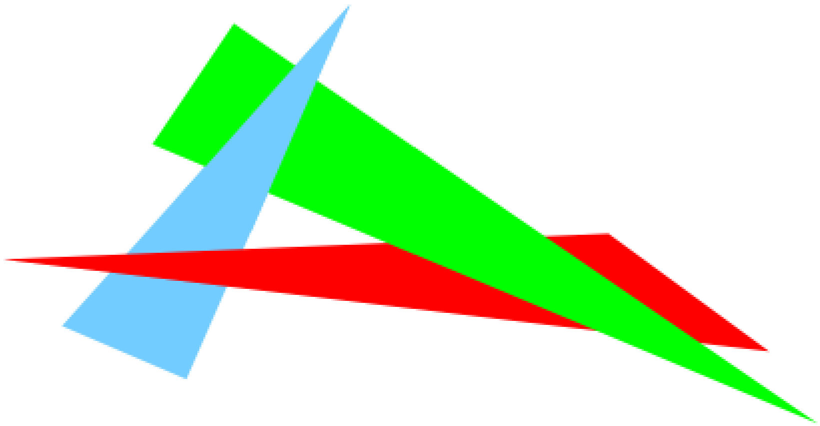
First Attempt: Painter's Algorithm

- Sort triangles (using z values in eye space)
- Draw triangles from back to front



Problems?

- Correctness issues:
  - Intersections
  - Cycles
  - Solve by splitting triangles, but ugly and expensive
- Efficiency (sorting)

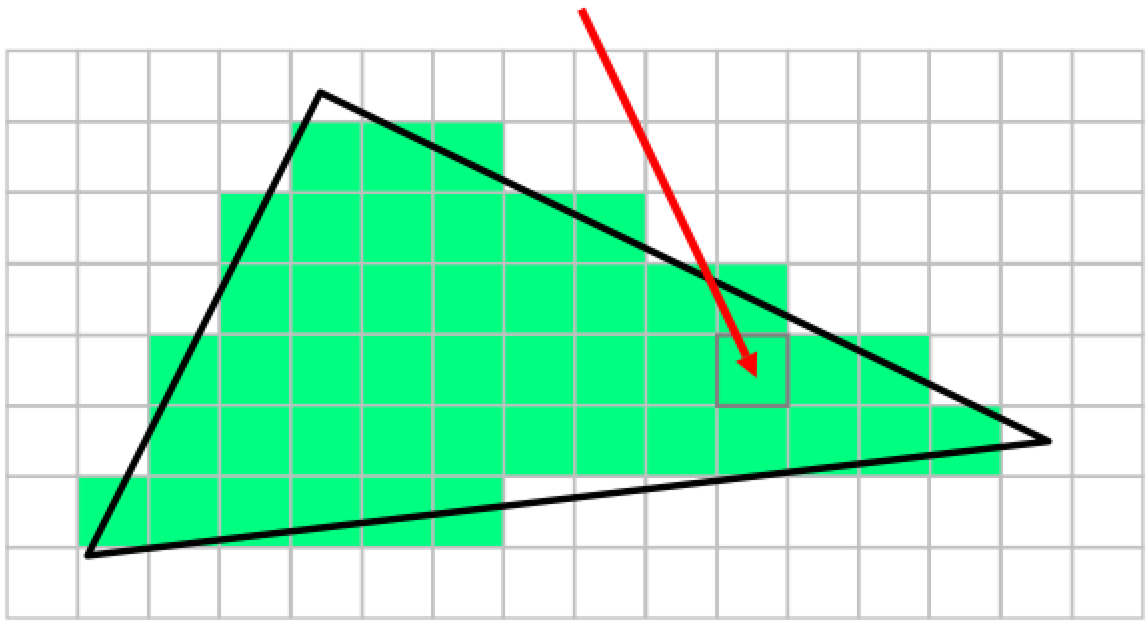


The Depth Buffer (Z-buffer)

- Perform hidden surface removal per-fragment
- Idea:
  - Each fragment gets a z value in screen space
  - Keep only the fragment with the smallest z value

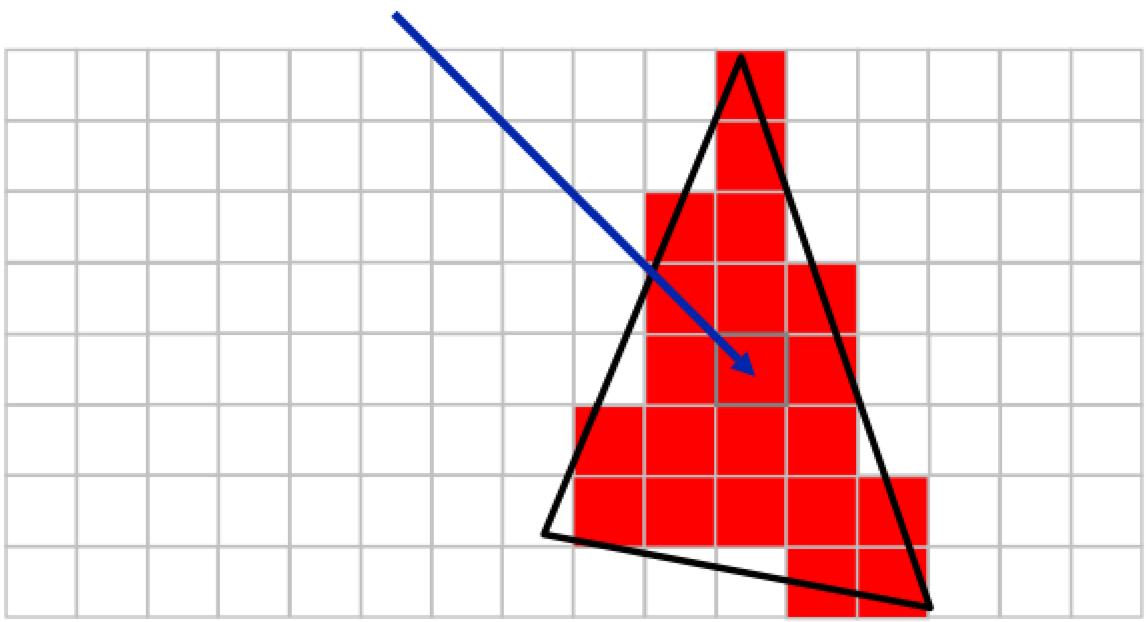
The Depth Buffer (Z-buffer)

- Example:
  - fragment from green triangle has z value of 0.7



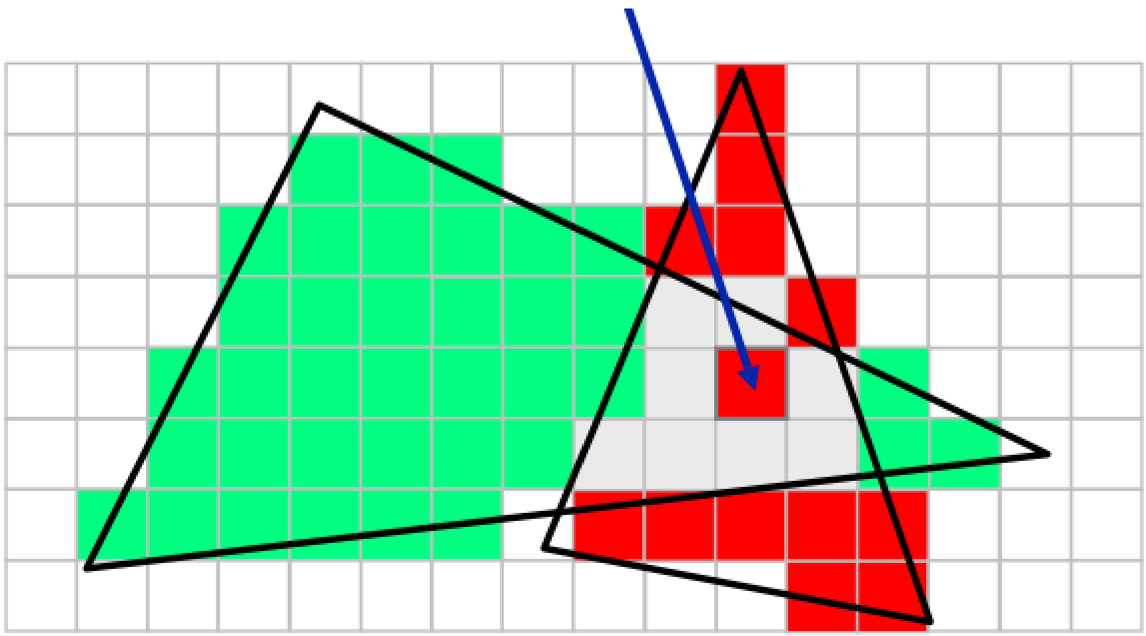
The Depth Buffer (Z-buffer)

- Example:
  - fragment from red triangle has z value of 0.3



The Depth Buffer (Z-buffer)

- Since 0.3 < 0.7, the red fragment wins



The Z-buffer

- Lots of fragments might map to the same pixel location
- How to track their z-values?
- Solution: z-buffer (2D buffer, same size as image)

1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.1	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.1	0.1	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.2	0.2	0.3	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.3	0.3	0.4	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.3	0.4	0.4	0.5	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.4	0.4	0.5	0.5	0.5	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.4	0.5	1.0	1.0	1.0

Z-buffer Algorithm

- Let CB be color buffer, ZB be z-buffer
- Initialize z-buffer contents to 1.0 (far away)

Z-buffer Algorithm

- Let CB be color buffer, ZB be z-buffer
- Initialize z-buffer contents to 1.0 (far away)
- For each triangle T
  - Rasterize T to generate fragments
    - For each fragment F with screen position (x,y,z) and color value C
      - If (z < ZB[x,y]) then
        - » Update color: CB[x,y] = C
        - » Update depth: ZB[x,y] = z

Z-buffer Algorithm Properties

- What makes this method nice?

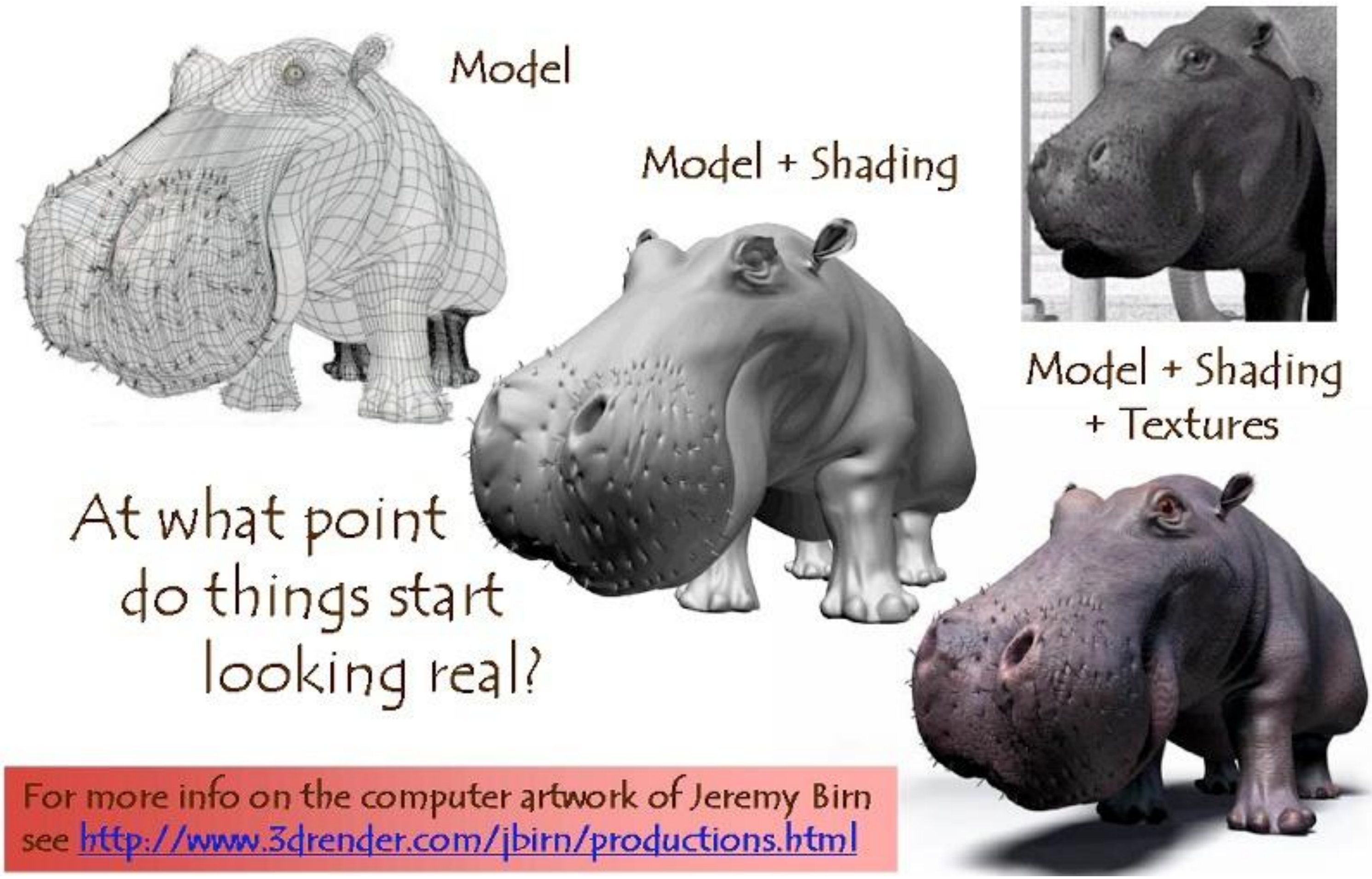
Z-buffer Algorithm Properties

- What makes this method nice?
  - simple (facilitates hardware implementation)
  - handles intersections
  - handles cycles
  - draw opaque polygons in any order

Outline

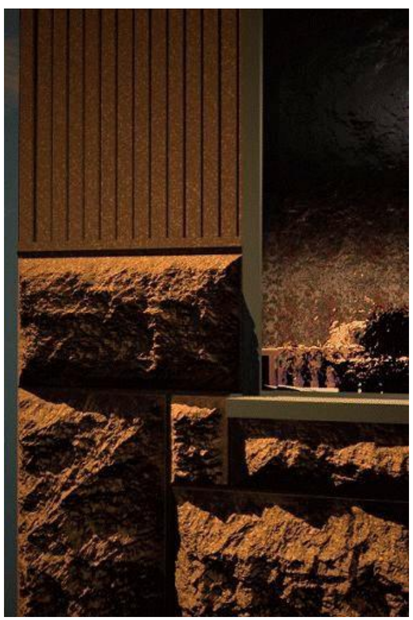
- Triangle rasterization using barycentric coordinates
- Hidden surface removal (z-buffer)
- Texture mapping

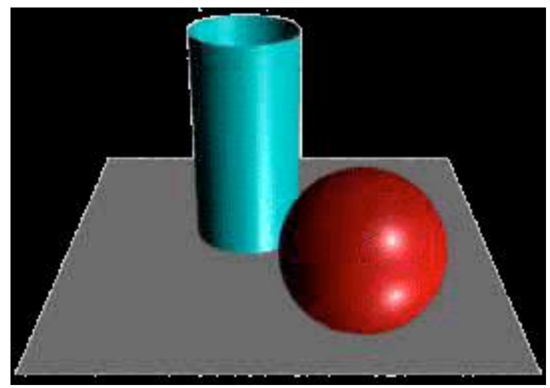
The Quest for Visual Realism

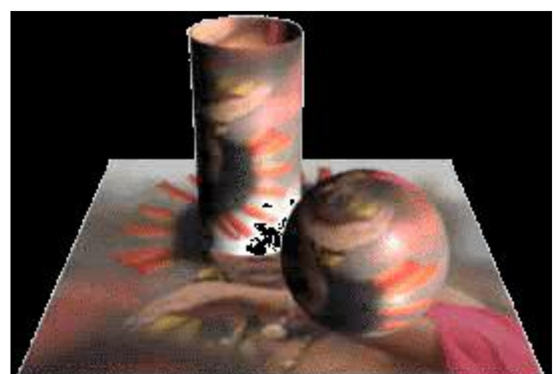


Idea

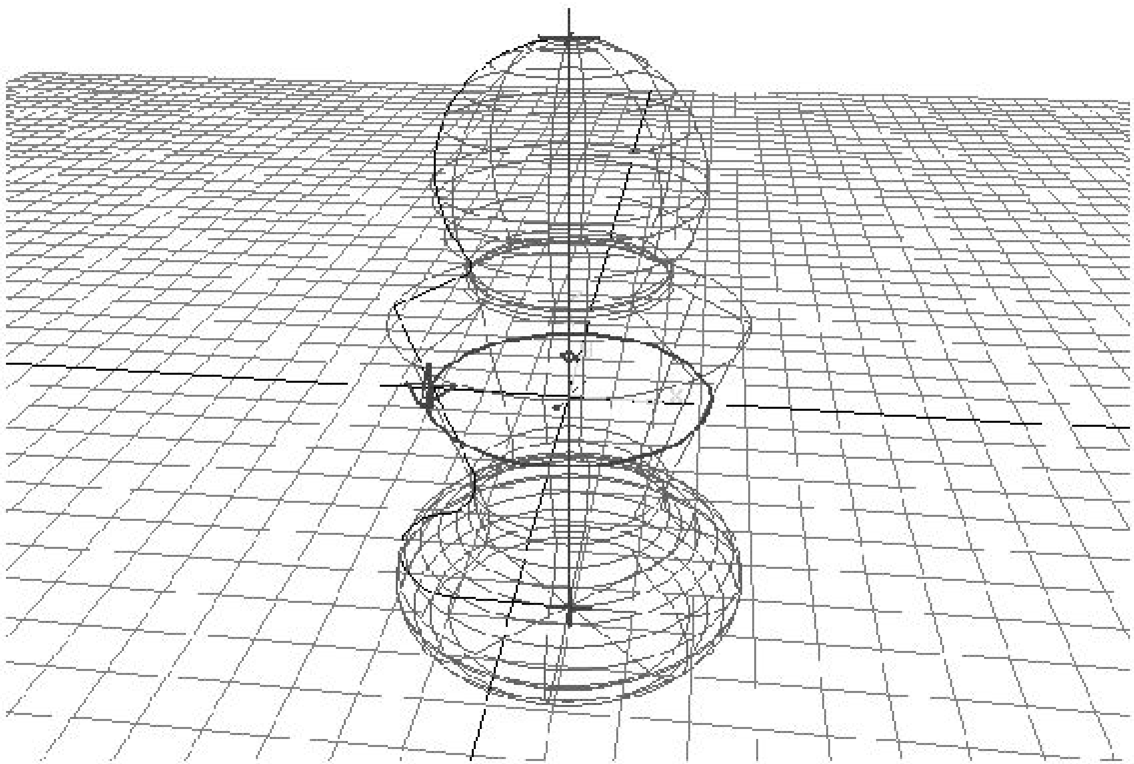
- Add surface detail without raising geometric complexity.
- Texture mapping is a shading trick that makes a surface look textured even though, geometrically, it is not.



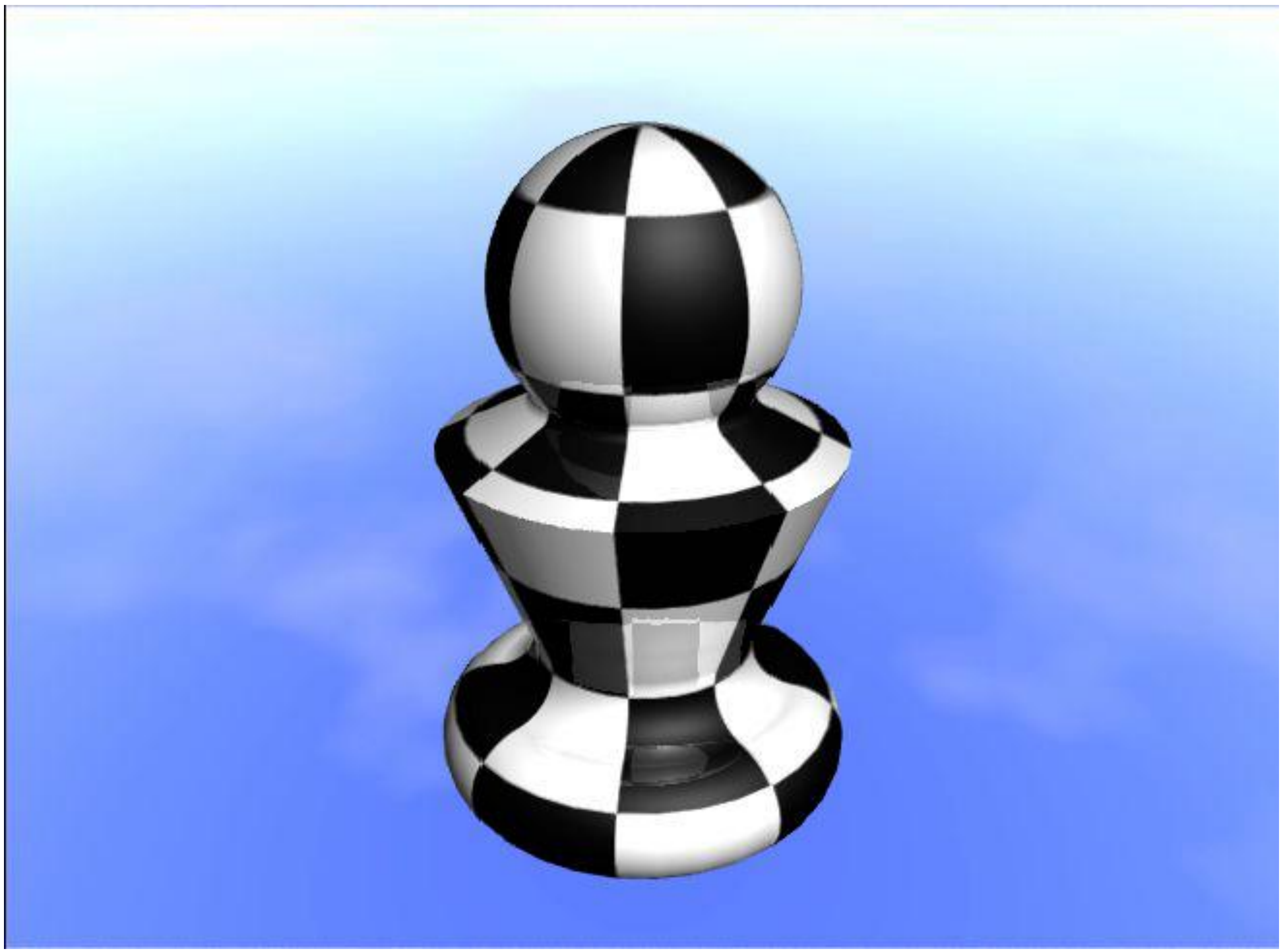




Example

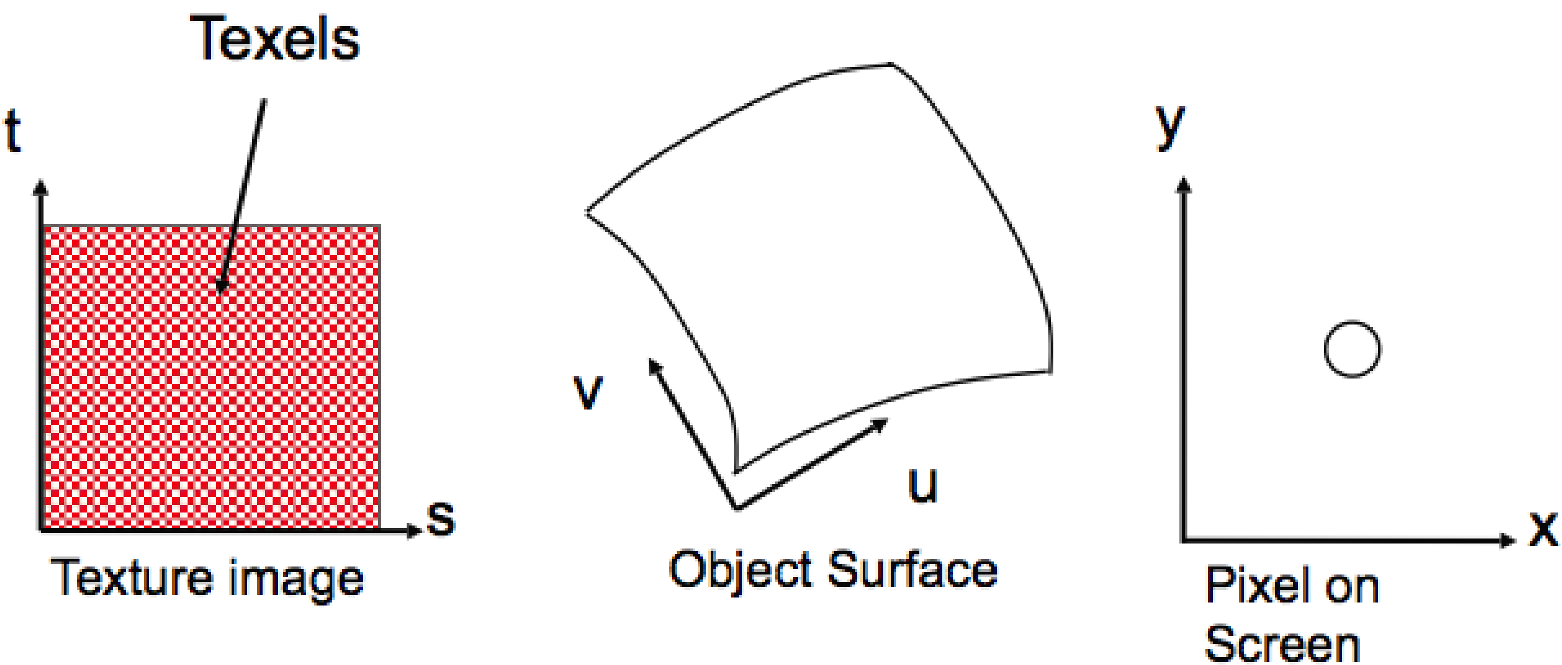


Example



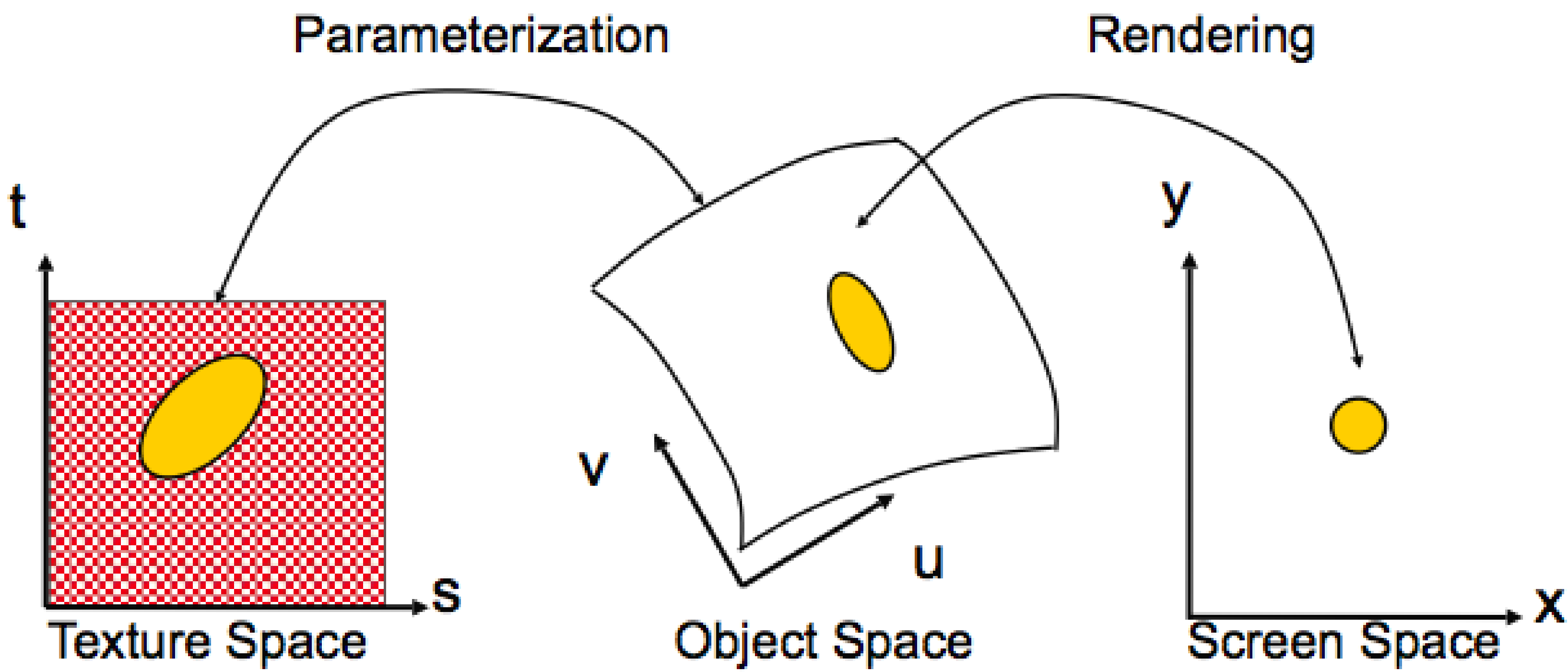
Concept of Texture Mapping

- Find mappings between different coordinate systems.



Mapping Terminology

- There are two main steps in texture mapping:



3D Graphics Pipeline

- !Texture mapping happens in the fragment (pixel) shader

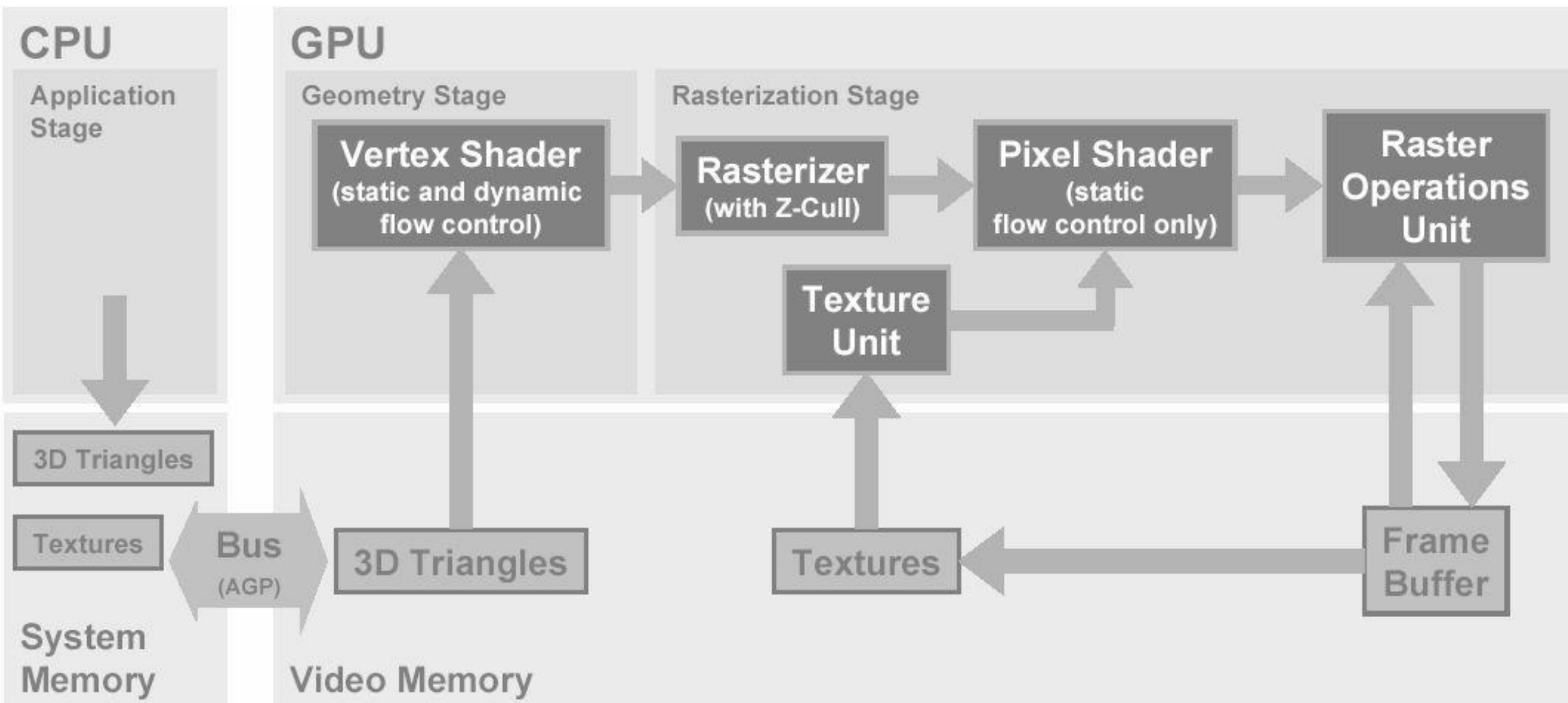


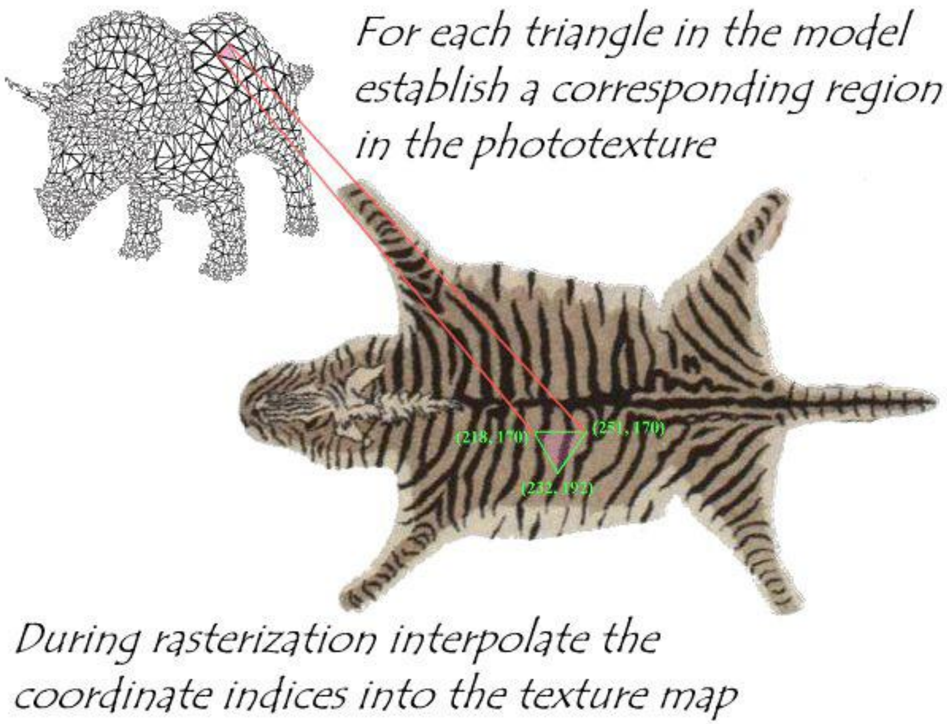
Photo Textures

- There are many free
- textures available online.
- Type “free textures” in
- Google.
- Check out Paul Bourke’s
- web page.

Texture Synthesis

Parameterization

- The concept is very simple!



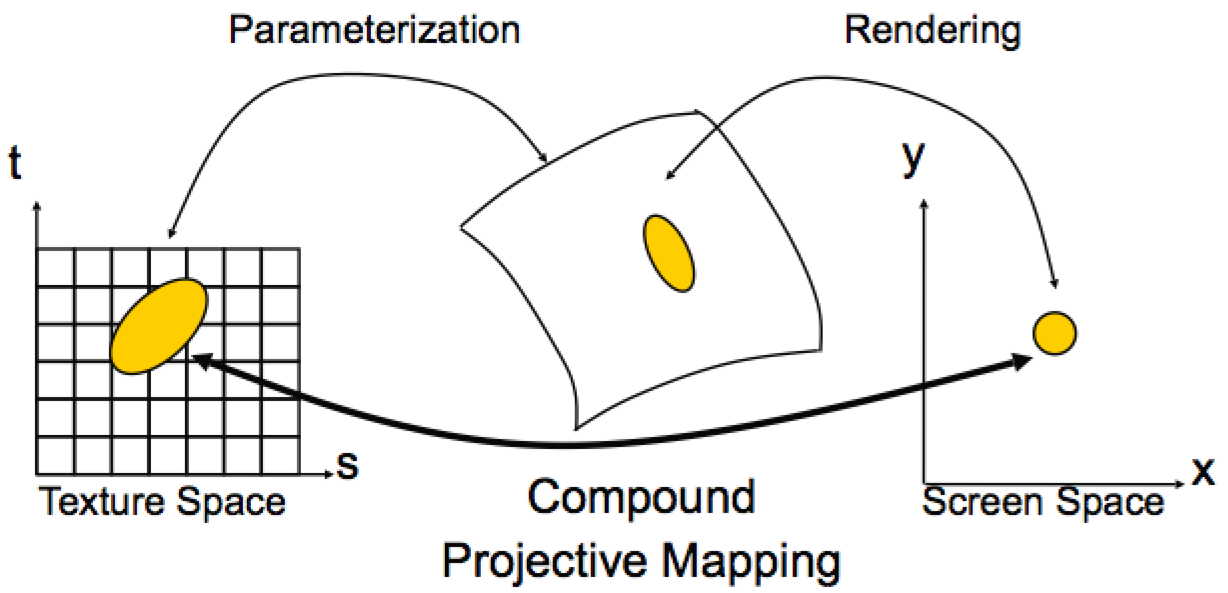
Parameterization in Practice

- Texture creation and parameterization is an art form
- You can even go to “Texture Mapping School”

- All 3D design programs (e.g., 3D Studio Max) provide tools for texture mapping

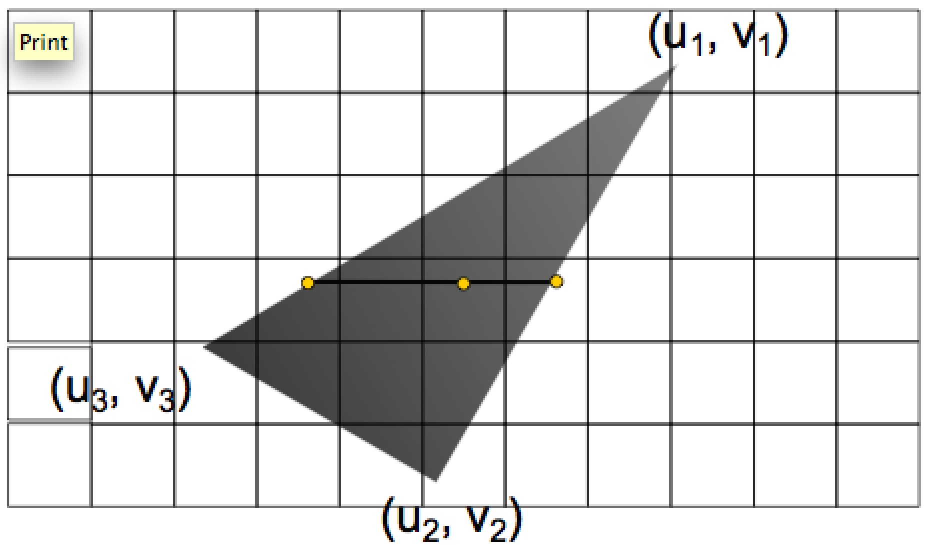
Texture Mapping

- Texture mapping is a 2D projective transformation
  - Texture coordinates (s, t) to screen coordinates (x, y)



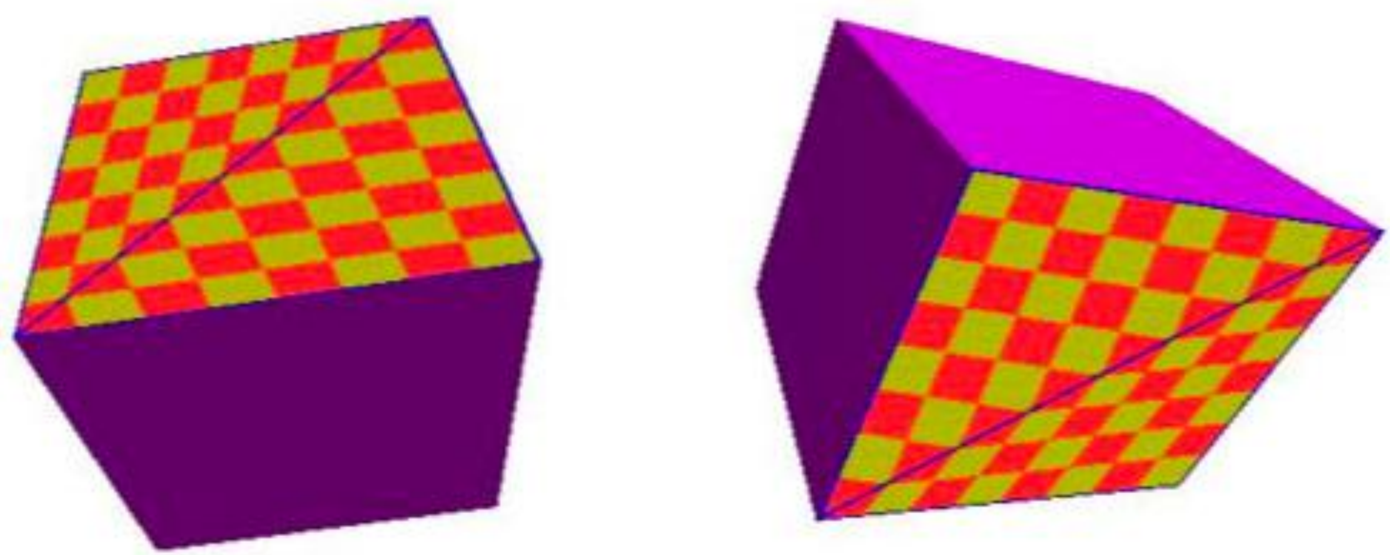
Triangle Rasterization

- Interpolate texture coordinates across scanlines
- The first idea is to use basic Gouraud shading for texture coordinates



But wait...

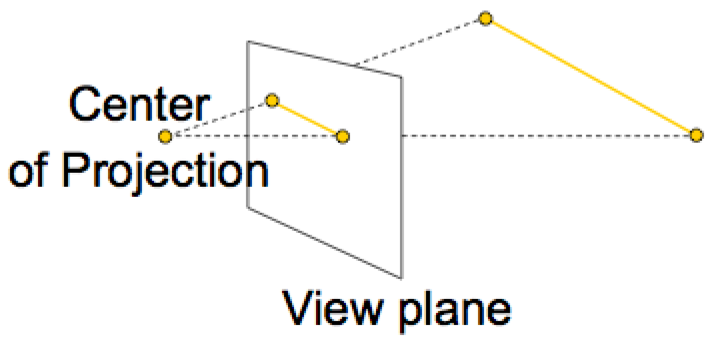
- Something is terribly wrong...

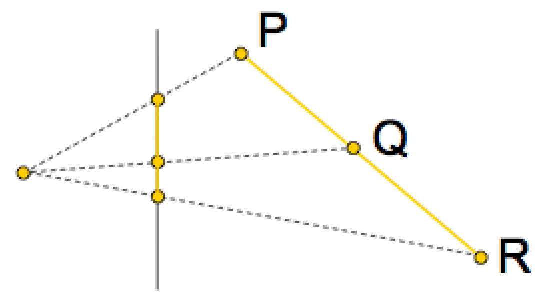


- Notice the distortion along the diagonal triangle edge of the cube face

Perspective Projection

- The problem is that perspective projection does not
- preserve affine combinations of points
- In particular, equal distances along a line in eye space do not map to equal distances in screen space

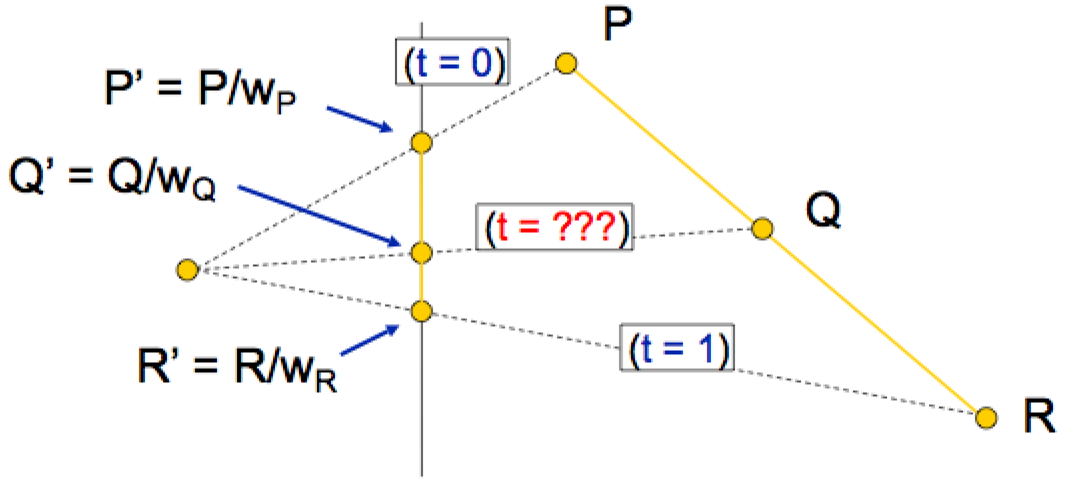




- Linear interpolation in screen space is not equal to linear interpolation in eye space!

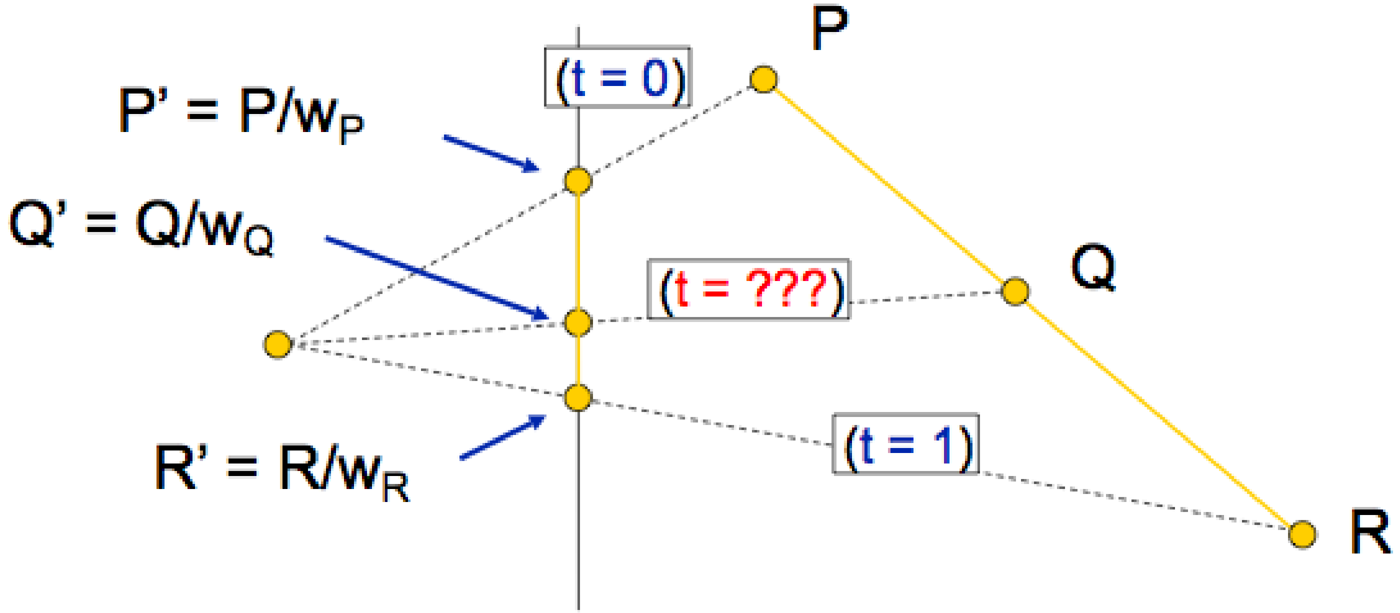
How to fix?

- Suppose we assign parameter **t to vertices P and R**
- Suppose **t = 0 at P, and t = 1 at R**
- **P projects to P' and R projects to R' (divide by w)**
- *What value should **t have at location Q'?***



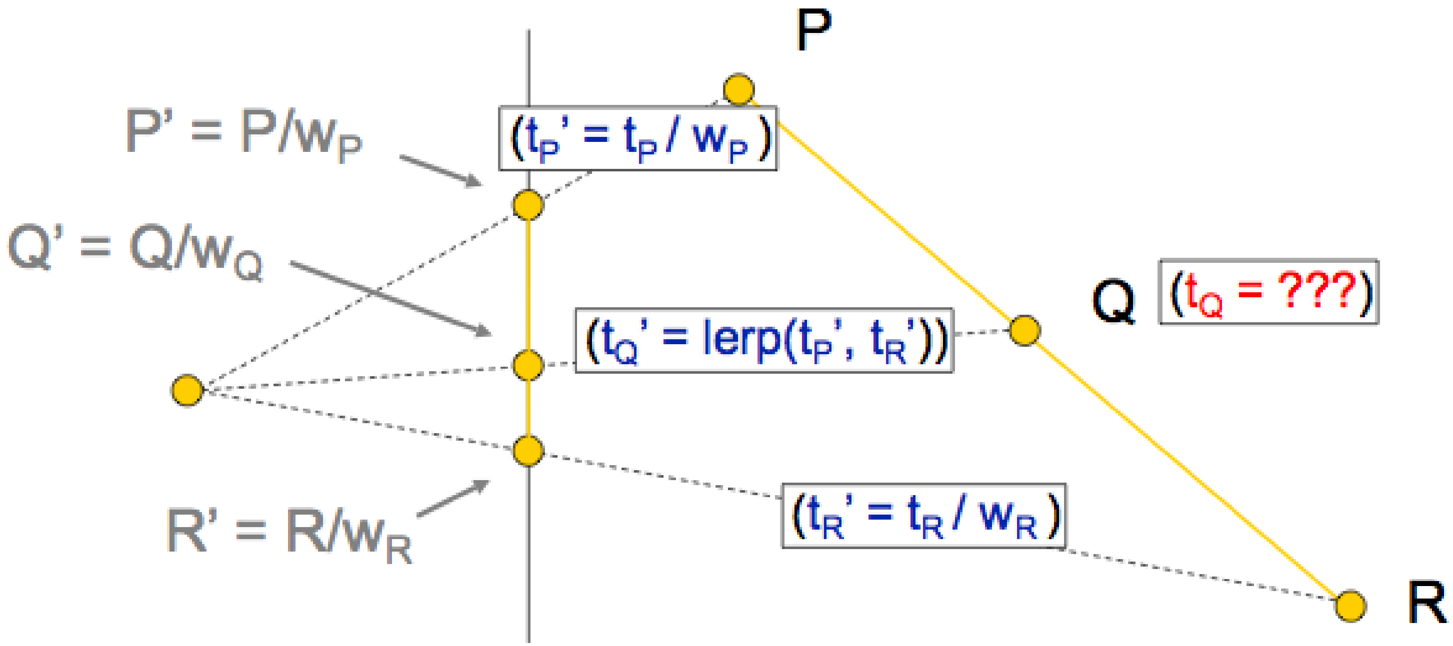
How to fix?

- We cannot linearly interpolate **t between P' and R'**
- Only projected values can be linearly interpolated in screen space
- Solution: perspective-correct interpolation



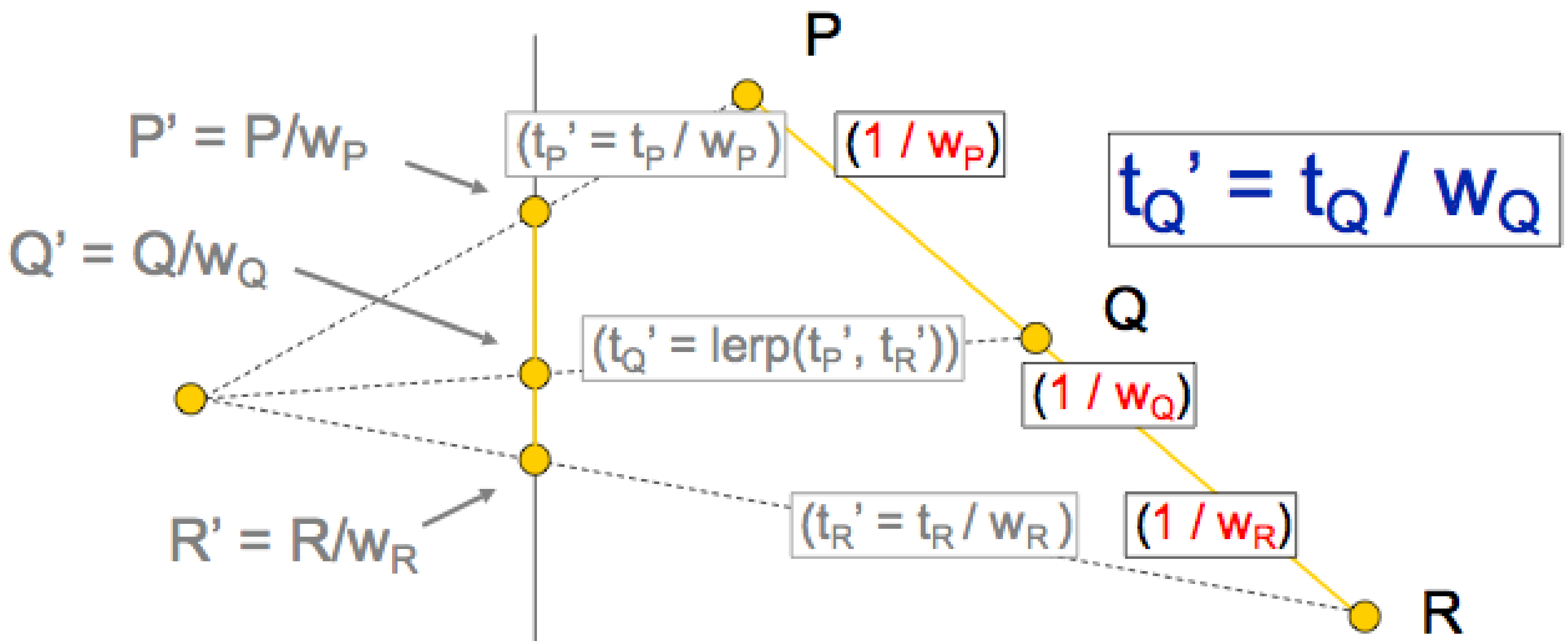
Perspective Correct Interpolation

- Linearly interpolate **t / w (not t) between P' and R'.**
  - Compute $\mathbf{t_{P'} = t_P / w_P}$ **and** $\mathbf{t_{R'} = t_R / w_R}$
  - Lerp $\mathbf{t_{P'}}$ **and** $\mathbf{t_{R'}}$ **to get** $\mathbf{t_{Q'}}$ **at location Q'**
- But, we want the (unprojected) parameter $\mathbf{t_Q}$**, not** $\mathbf{t_{Q'}}$**.**



Perspective Correct Interpolation

- The parameter $\mathbf{t_{Q'}}$ **is related to** $\mathbf{t_Q}$ **by a factor of 1 / w:**
  - Lerp $\mathbf{1 / w_P}$ **and** $\mathbf{1 / w_R}$ **to obtain** $\mathbf{1 / w_Q}$ **at point Q'.**
  - Divide $\mathbf{t_{Q'}}$ **by** $\mathbf{1 / w_Q}$ **to get** $\mathbf{t_Q}$

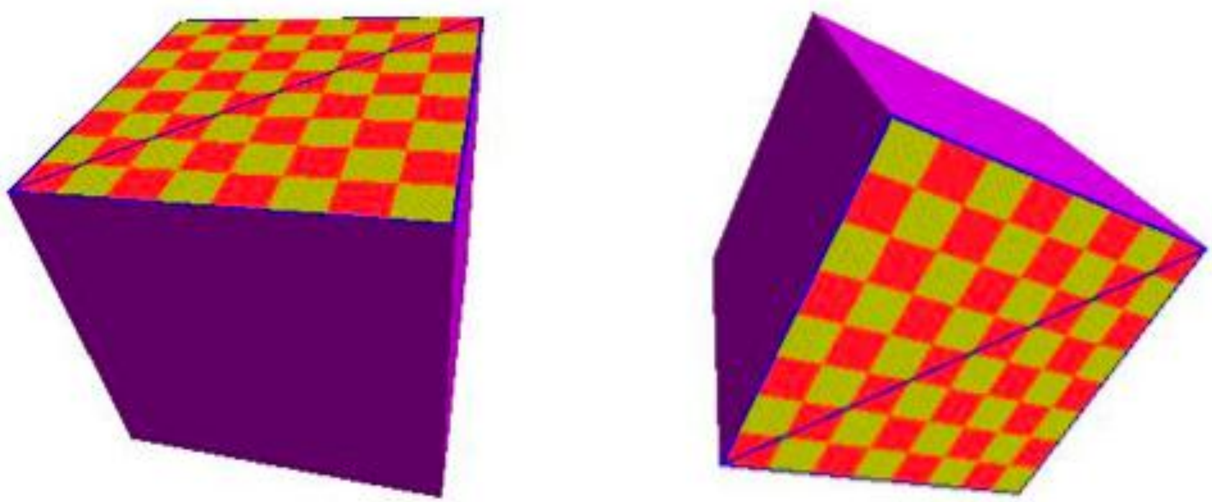


Perspective Correct Interpolation

- Summary:
  - Given parameter **t at vertices:**
    - Compute **1 / w for each vertex (per-vertex)**
    - Lerp **1 / w across the triangle (per-fragment)**
    - Lerp **t / w across the triangle (per-fragment)**
    - Do perspective division (per-fragment):
      - Divide **t/w by 1/w to obtain interpolated parameter t**

Perspective Correct Interpolation

- This looks correct now:

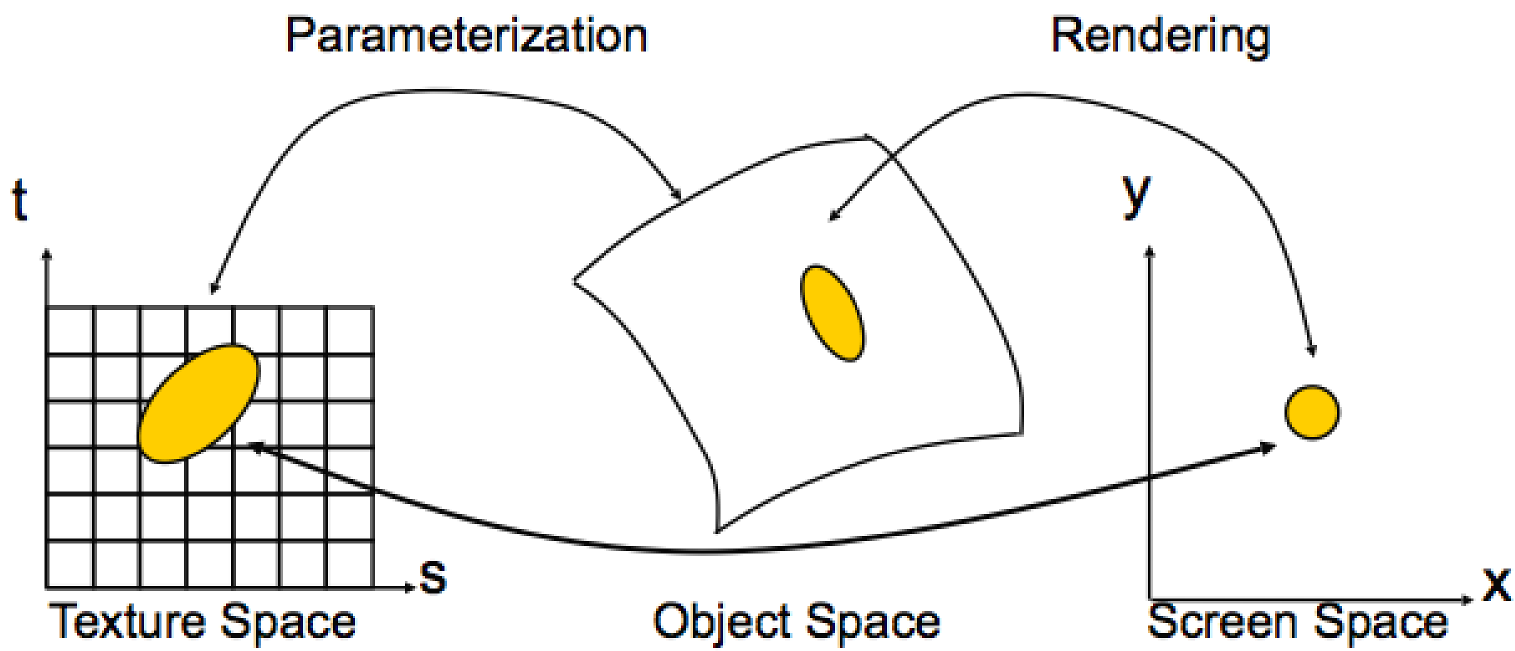


Perspective Correction Hint

- Texture coordinate and color interpolation
  - Either linearly in screen space (wrong)
  - Or using perspective correct interpolation (slower)
- **glHint(GL_PERSPECTIVE_CORRECTION_HINT, hint)**
- where **hint is one of:**
  - **GL_NICEST: Perspective**
  - **GL_FASTEST: Linear**
  - **GL_DONT_CARE: Linear**

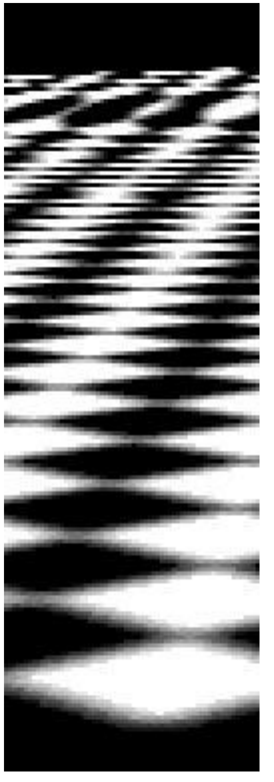
Screen Pixels in Texture Space

- Let's look at what can happen to pixels mapped into texture space:



Sampling Texture Maps

- When texture mapping it is rare that the screen space pixel sampling density matches the sampling density of the texture
- Typically one of two things can occur:
  – Minification
  – Magnification

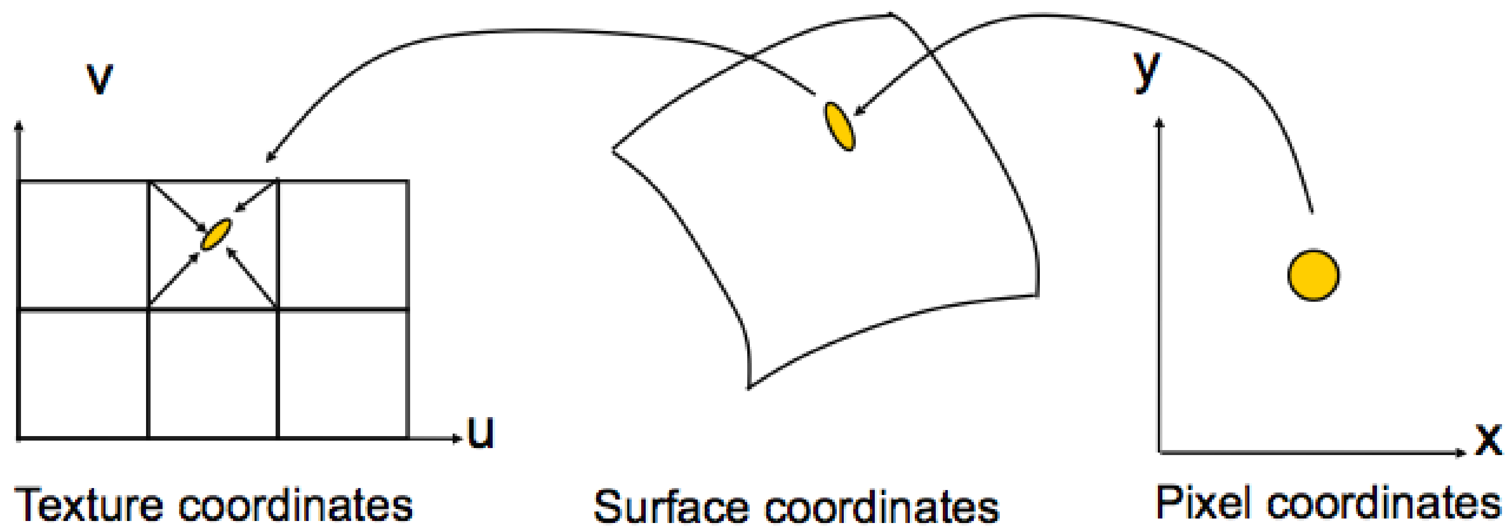


Minification

Magnification

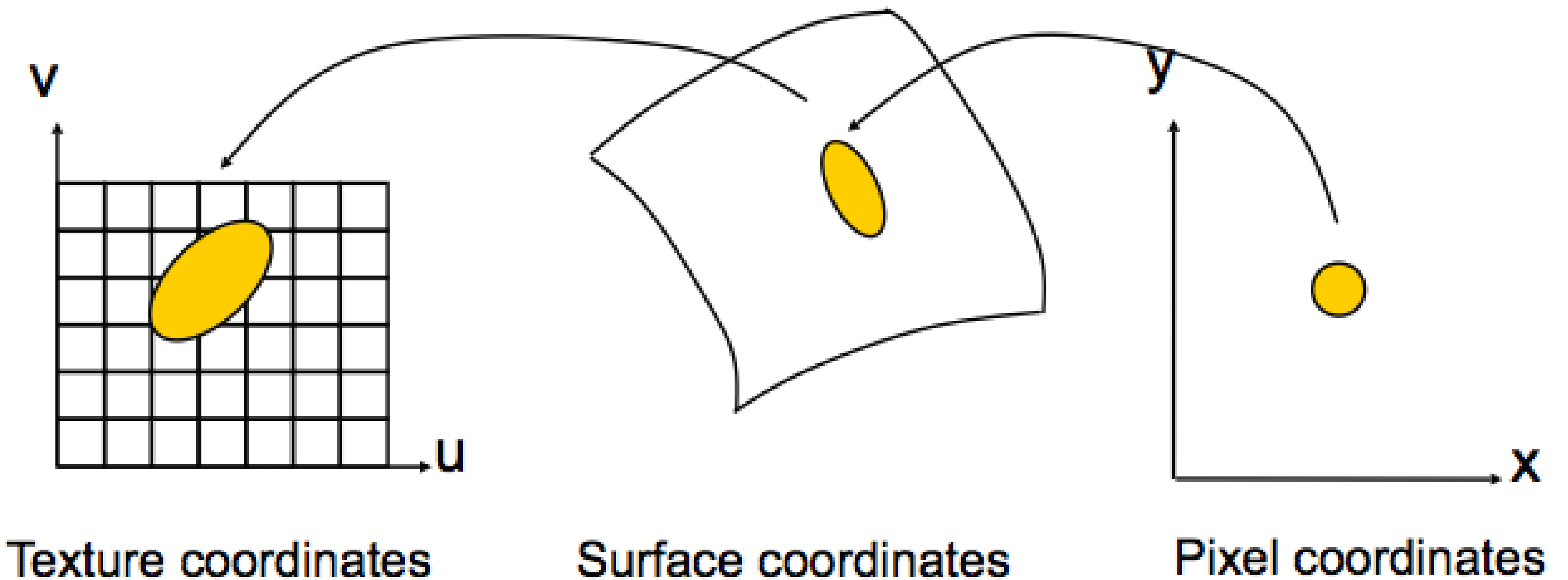
Magnification

- This happens when you zoom really close into a texture mapped polygon or due to perspective projection
- A pixel projects to something smaller than a texel in texture space



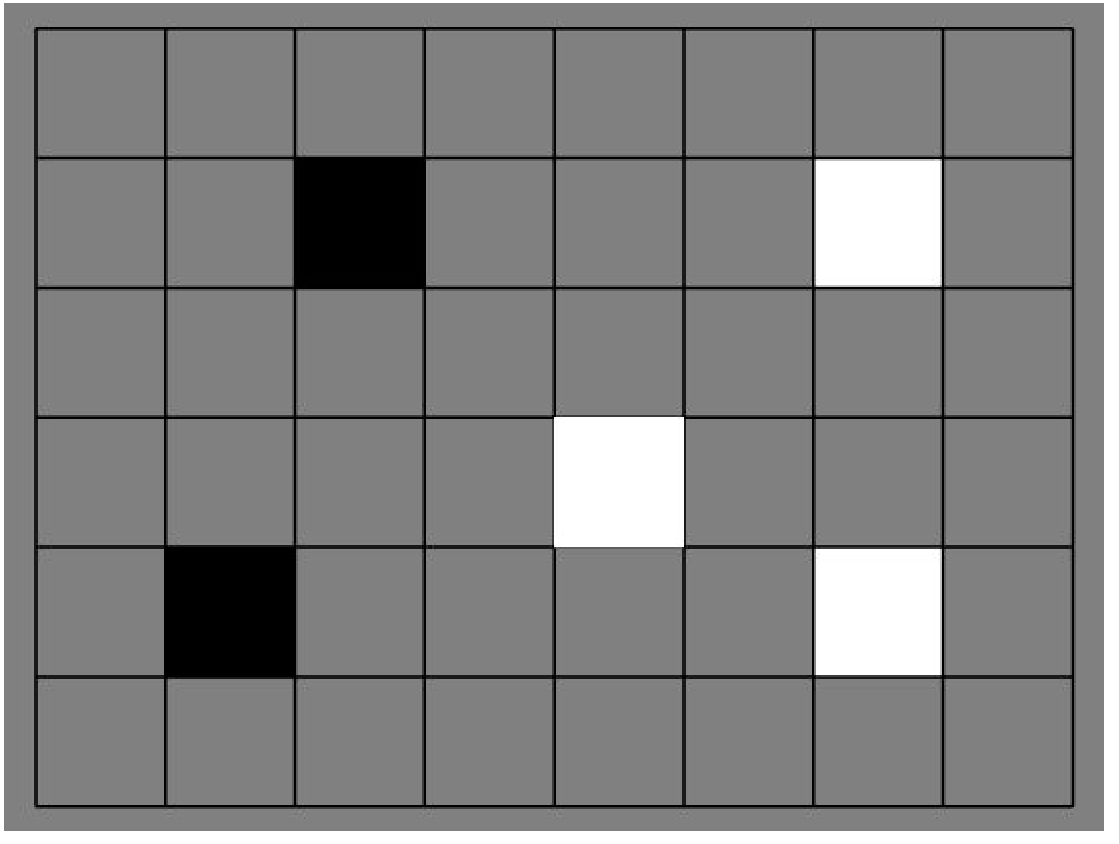
Minification

- This happens when you zoom out or due to perspective foreshortening.
- A pixel projects onto several texels in texture space.



Nearest Neighbor Interpolation

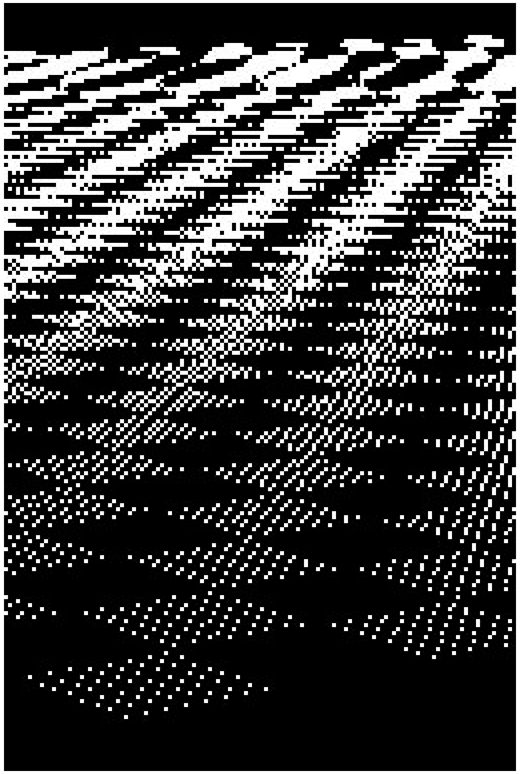
Texture Space



Nearest Neighbor Interpolation

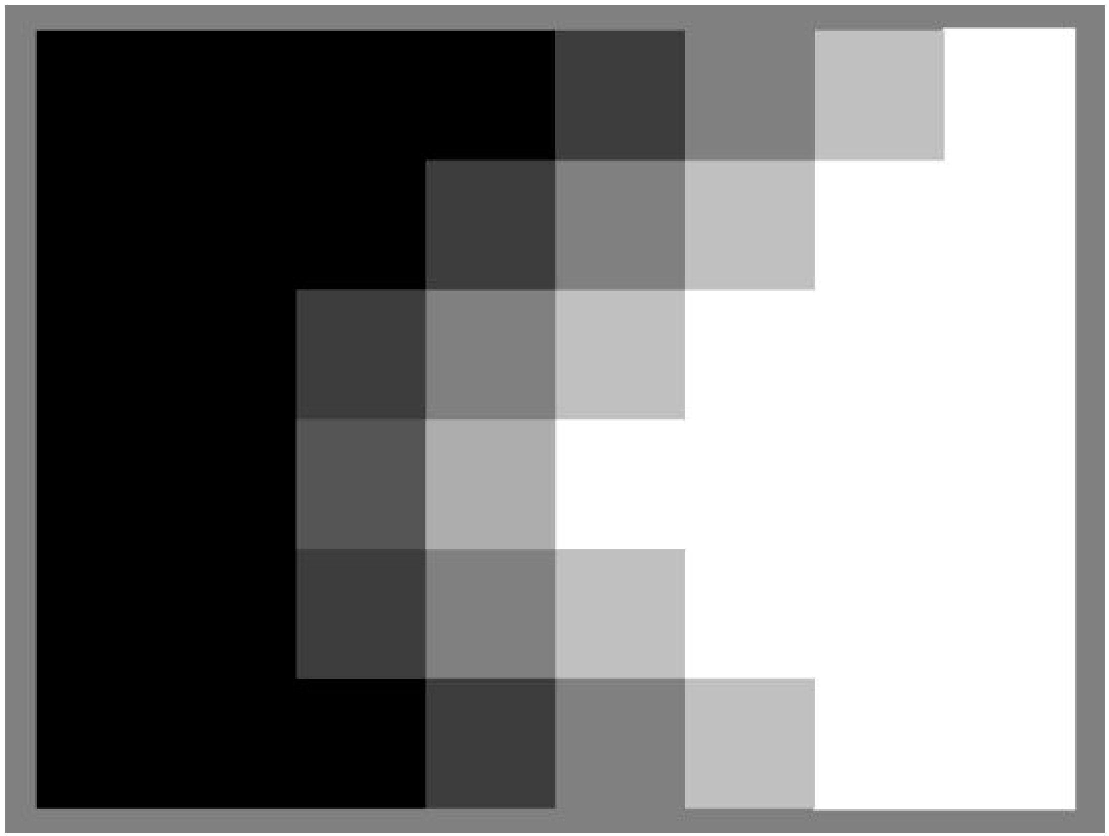
Minification

Magnification

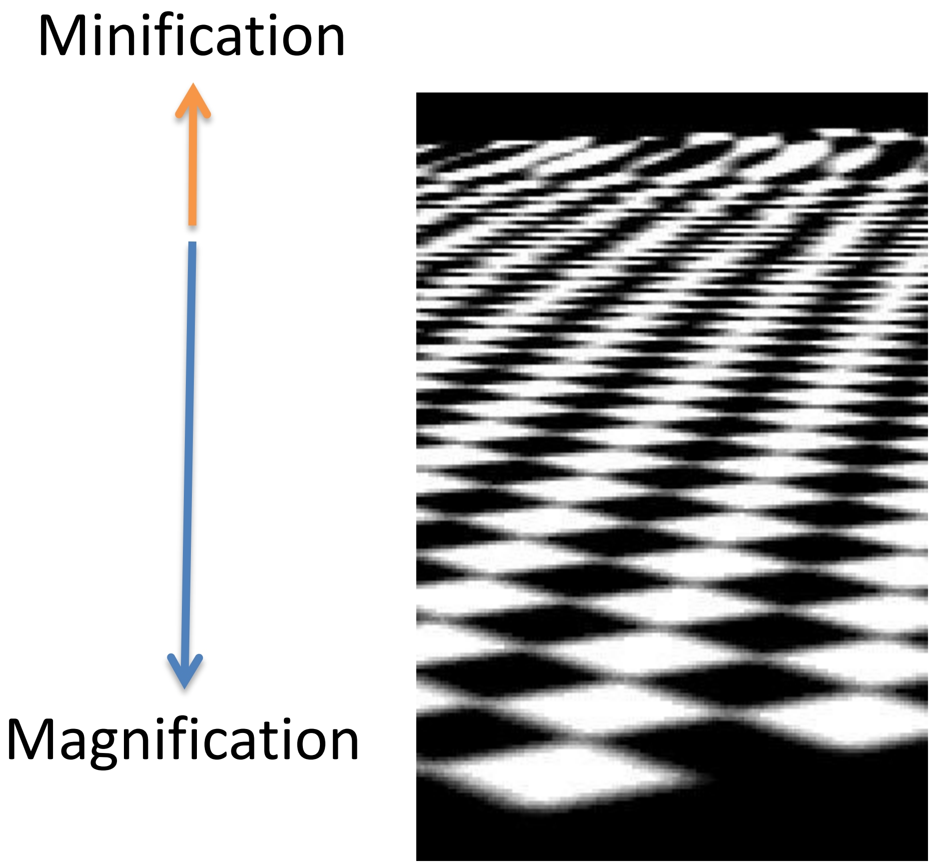


Better Filters

Texture Space



Better Filters

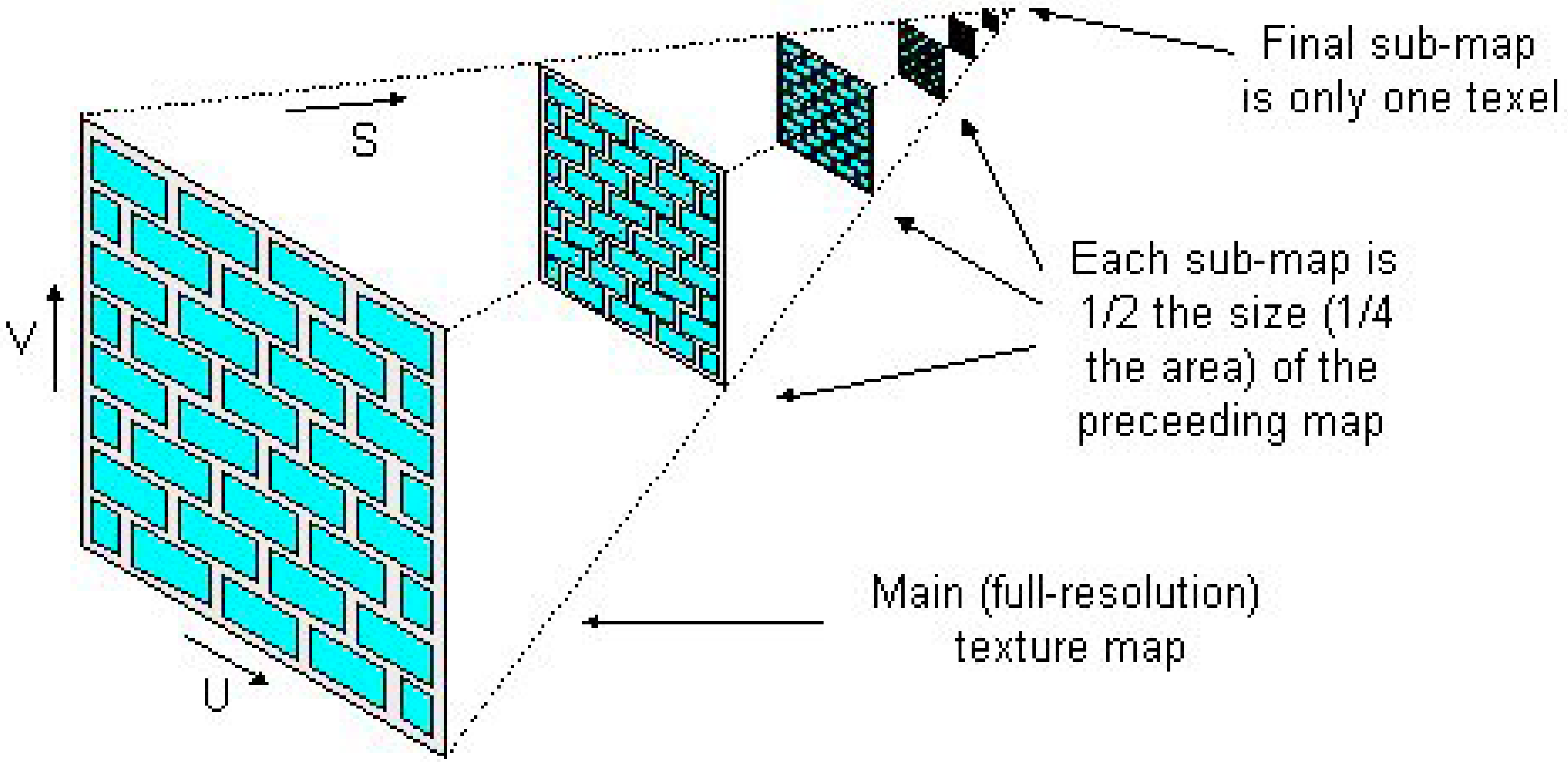


Pre-Filters

- We could perform this filtering during texture mapping
- However, even this is very expensive
- Instead, we can use pre-filtering of the texture prior to rendering

MipMapping

- The basic idea is to construct a pyramid of images that are pre-filtered and down-sampled



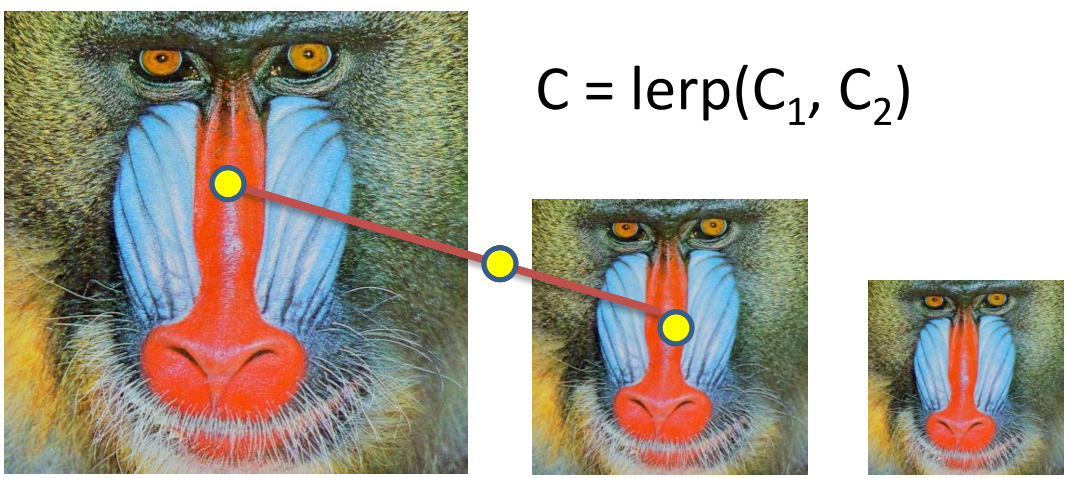
Generate Mipmaps

- You can use **glTexImage2D() directly, but it's much** simpler to use **gluBuild2DMipmaps()**
  - **gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB,**
    - **width, height, GL_RGB,**

    GL_UNSIGNED_BYTE, mytextureimage);
- This is a great function! The 2D texture does not even need to be a power of 2!

MipMap Interpolation

- Sometimes, a pixel really should be filtered with a filter size that is not part of the mipmap
- We can use linear interpolation between mipmap levels to compute the texel color
- If we use a bi-linear filter inside the mip-mapped texture, this is called tri-linear interpolation



$C = \mathrm{lerp}(C_1, C_2)$