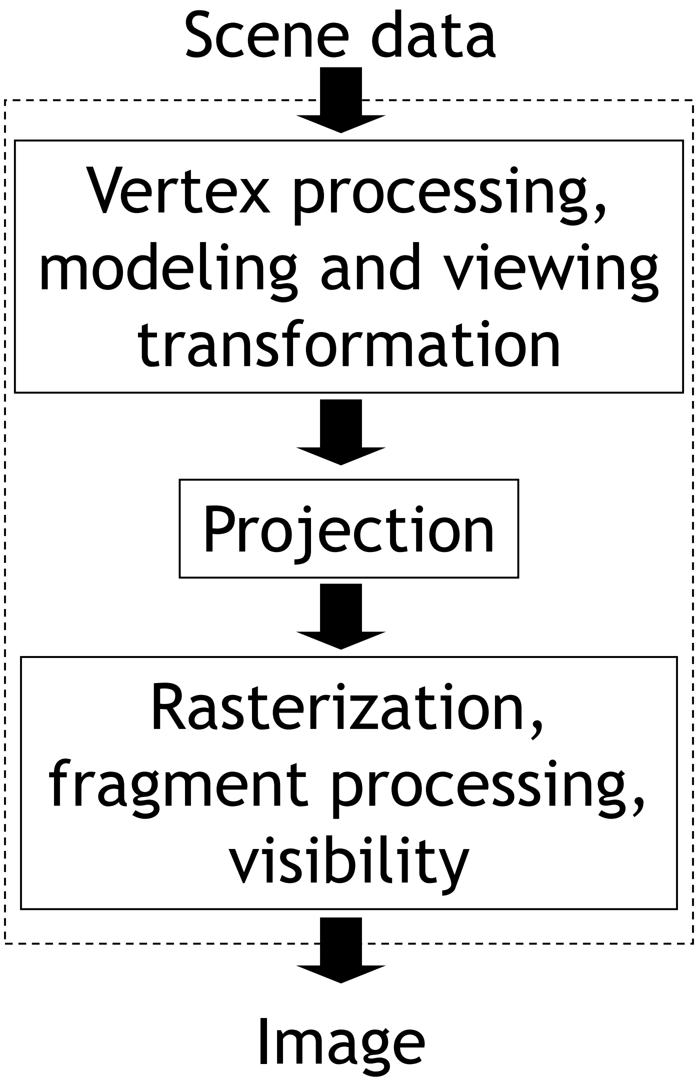
Computergrafik

Matthias Zwicker
Universität Bern
Herbst 2016

Rendering pipeline



Lectures 1 and 2

Lecture 3(last time)

Lecture 4 (today):
rasterization, visibility

Lecture 5-7: shading

Base code architecture

jrtr

Scene data

↓

Vertex processing, modeling and viewing transformation

↓

Projection

↓

Rasterization, fragment processing, visibility

↓

Image

Java library

↔

simple

Application program

- No OpenGL/jogl calls
- Independent of „rendering backend“ (low level graphics API)
- Can easily change rendering backend (OpenGL/jogl, software renderer)

Java executable

The complete vertex transform

- Mapping a 3D point in object coordinates to pixel coordinates

- Object-to-world matrix $\mathbf{M}$, camera matrix $\mathbf{C}$ projection matrix $\mathbf{P}$, viewport matrix $\mathbf{D}$

$$\mathbf{p}' = \mathbf{D}\mathbf{P}\mathbf{C}^{-1}\mathbf{M}\mathbf{p}$$

Object space

World space

Camera space

Canonic view volume

Image space

The complete vertex transform

- Mapping a 3D point in object coordinates to pixel coordinates
- Object-to-world matrix M, camera matrix C projection matrix P, viewport matrix D

$$\mathbf{p}' = \mathbf{DPC}^{-1}\mathbf{Mp}$$

$$\mathbf{p}' = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

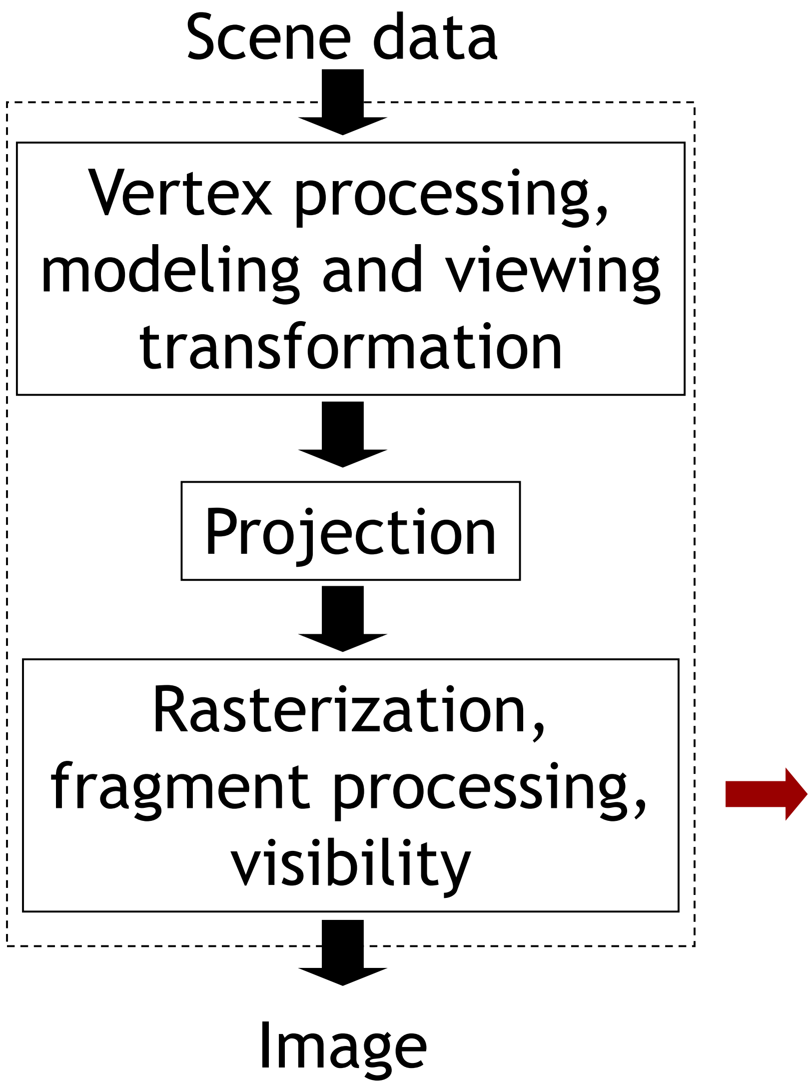
Pixel coordinates $\begin{matrix} x/w \\ y/w \end{matrix}$

Today

Drawing triangles

- Homogeneous rasterization
- Texture mapping
- Perspective correct interpolation
- Visibility

Rendering pipeline



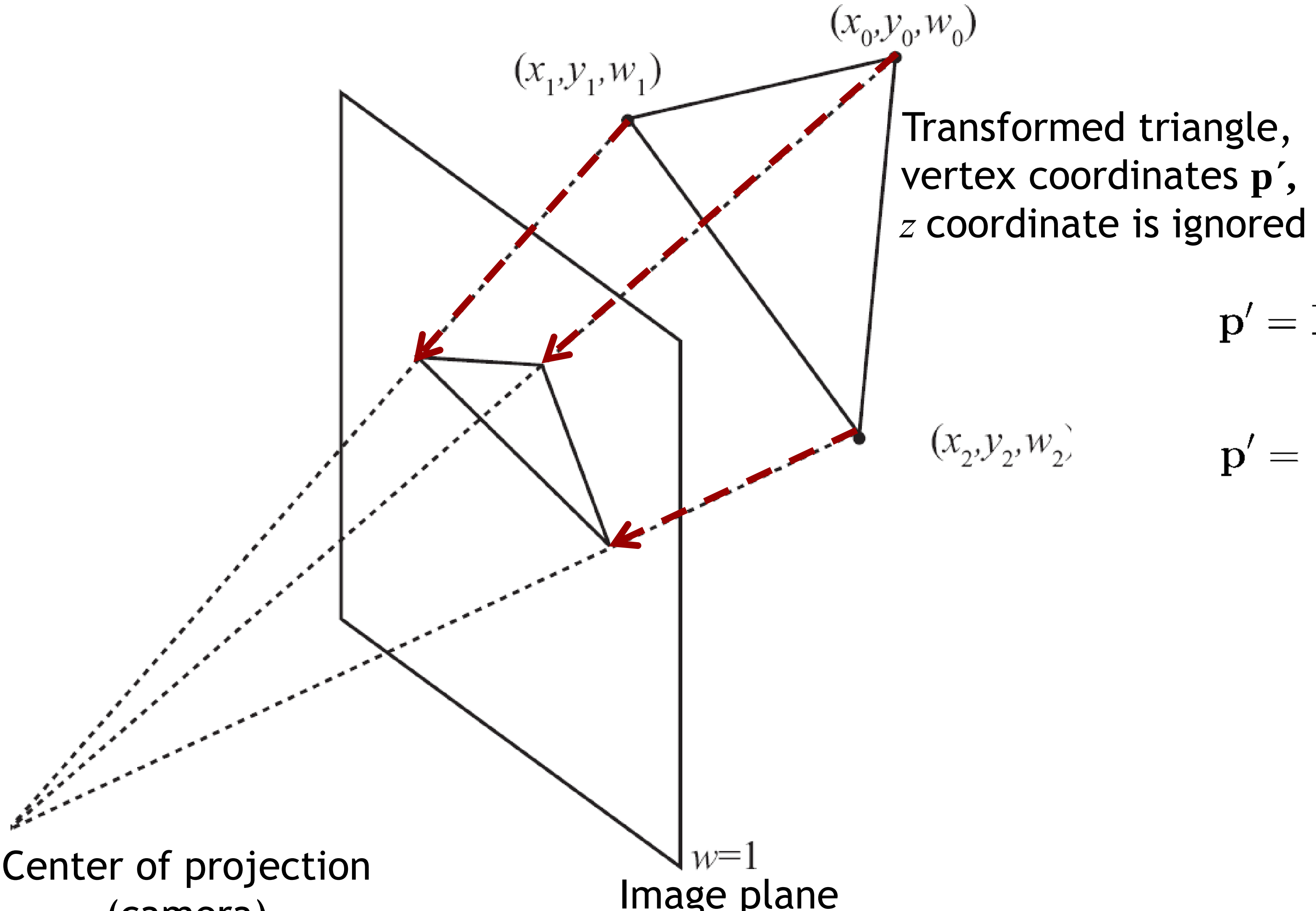
- Scan conversion and rasterization are synonyms
- One of the main operations performed by GPU
- Draw triangles, lines, points (squares)
- Focus on triangles in this lecture

Rasterization

- How many pixels can a modern graphics processor draw per second?
- See for example
  http://en.wikipedia.org/wiki/Comparison_of_Nvidia_graphics_processing_units

Rasterization

- Ideas?

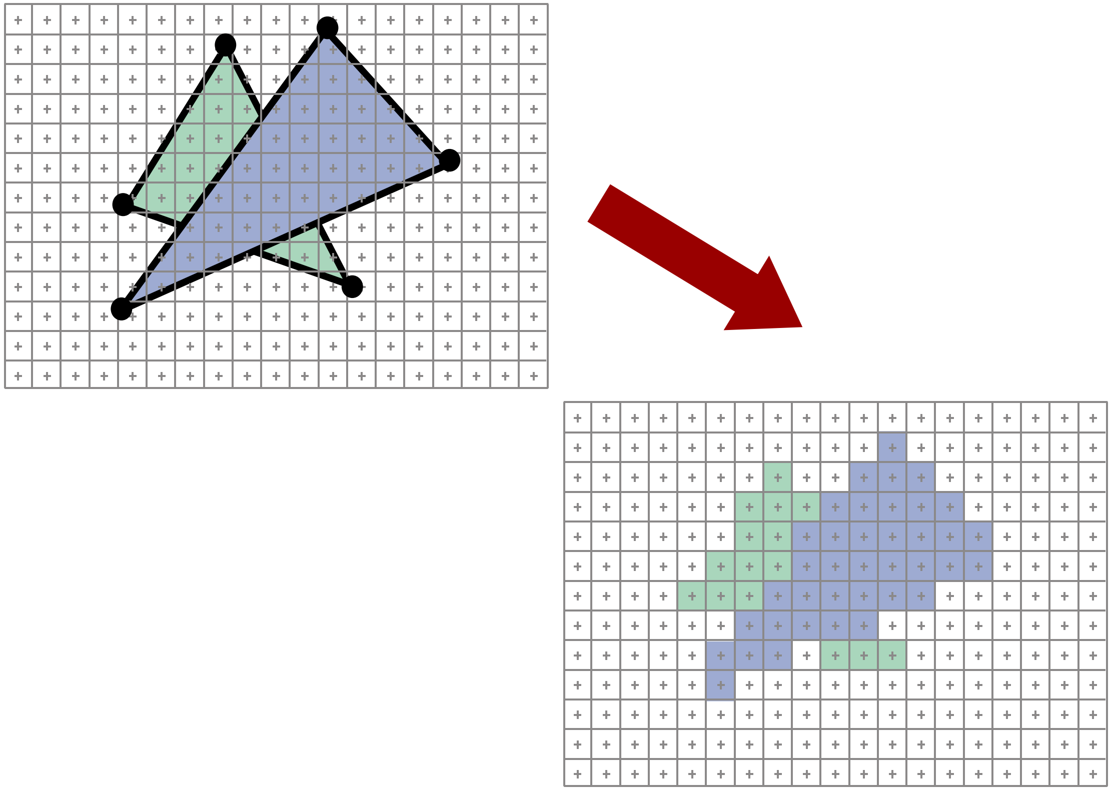


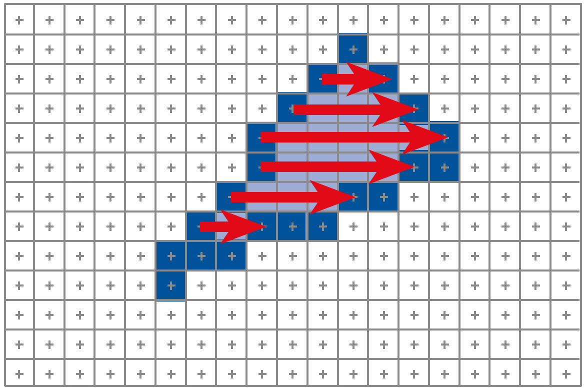
$$\mathbf{p}' = \mathbf{DPC}^{-1}\mathbf{Mp}$$

$$\mathbf{p}' = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Rasterization

- Idea
  - Project vertices by dividing by w
  - Fill triangle given by projected vertices





„scan conversion“

Rasterization

- Idea
  - Project vertices by dividing by w
  - Fill triangle given by projected vertices
- Problems
  - What happens if $w=0$ for some vertices?
  - What happens if some vertices have $w>0$, others $w<0$?

Clipping

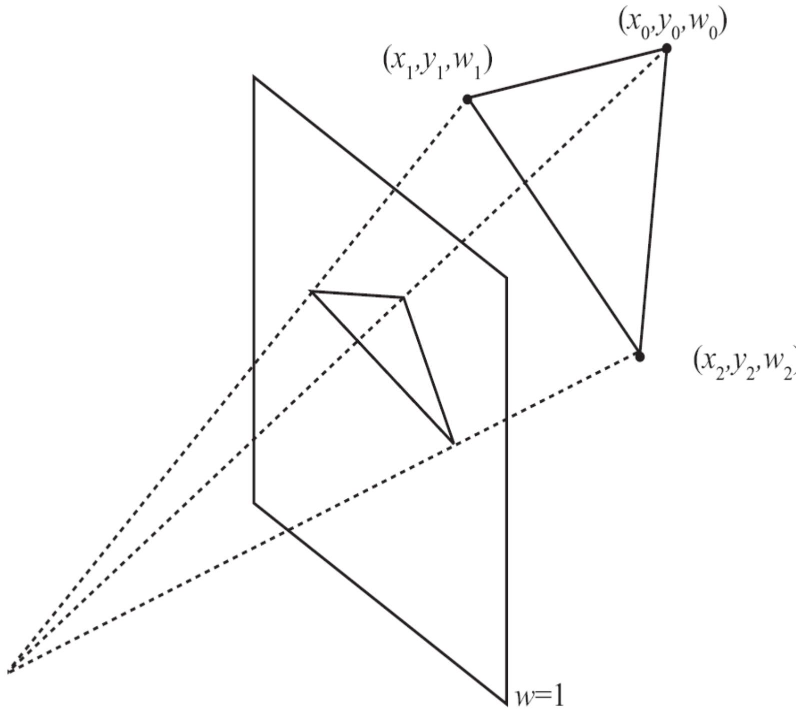
- Split (subdivide) triangles along view volume boundary into smaller ones
- Draw only triangles completely within view volume
- Many sophisticated algorithms, but still complicated and slow
  - Sutherland-Hodgman
    http://en.wikipedia.org/wiki/Sutherland%E2%80%93Hodgman
  - Weiler-Atherton
    http://en.wikipedia.org/wiki/Weiler%E2%80%93Atherton
- Try to avoid clipping!

Homogeneous rasterization

- Based on not-so-old research (1995)
  http://www.cs.unc.edu/~olano/papers/2dh-tri/
- Method of choice for GPU rasteriazation
  – Patent (NVidia) http://www.patentstorm.us/patents/6765575.html
- Does not require homogeneous division at vertices
  – Does not require costly clipping
- Caution
  – Different algorithm than in Shirley’s book (Sec. 3.6)
  – Read for comparison

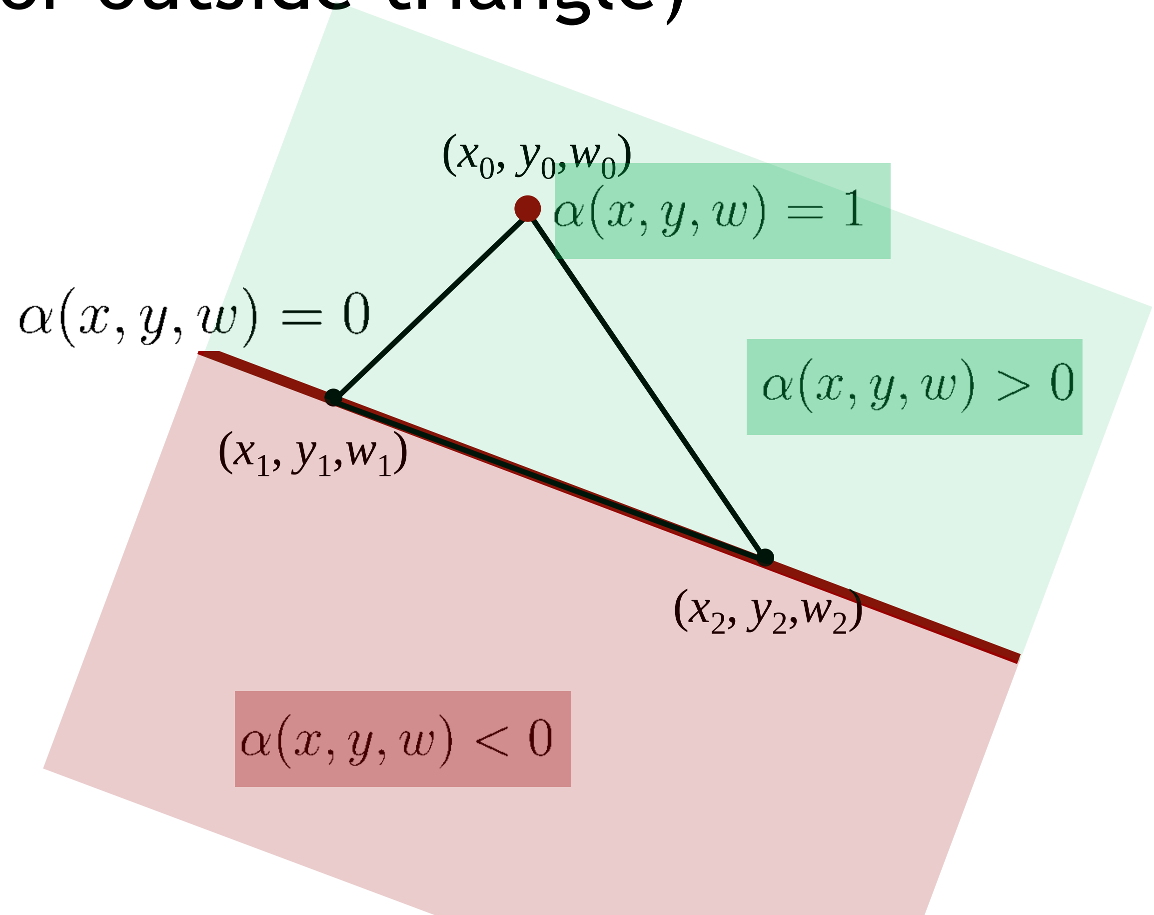
Homogeneous rasterization

- Idea: define linear edge functions on triangles
  - Three functions, one for each edge
  - In x,y,w coordinates (2D homogeneous coordinates), before projecton (i.e., homogeneous division)
  - Functions denoted $\alpha(x,y,w)$, $\beta(x,y,w)$, $\gamma(x,y,w)$



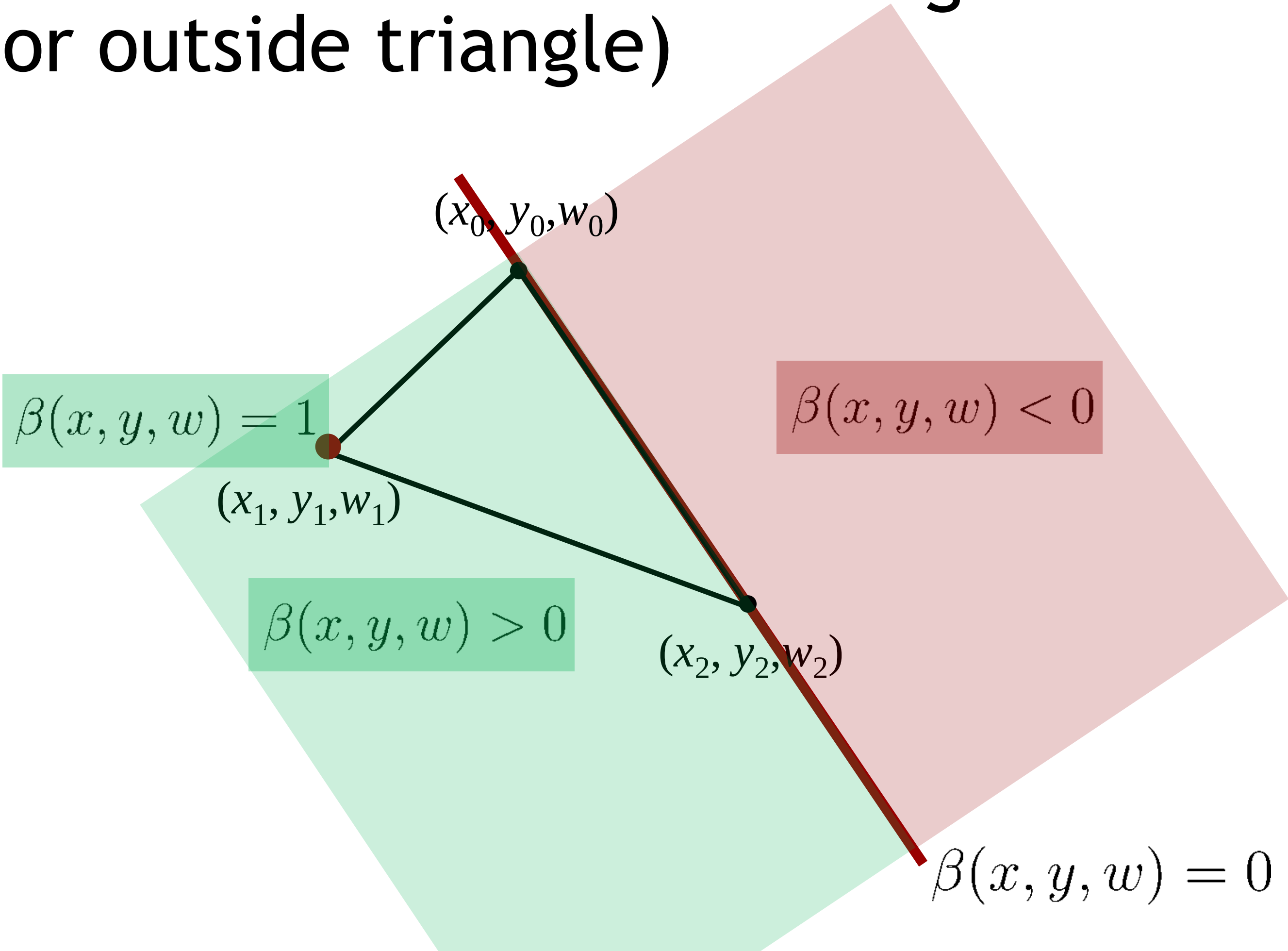
Edge functions

- Edge functions are zero on one edge, one at opposite vertex
- Sign indicates on which side of edge we are (inside or outside triangle)



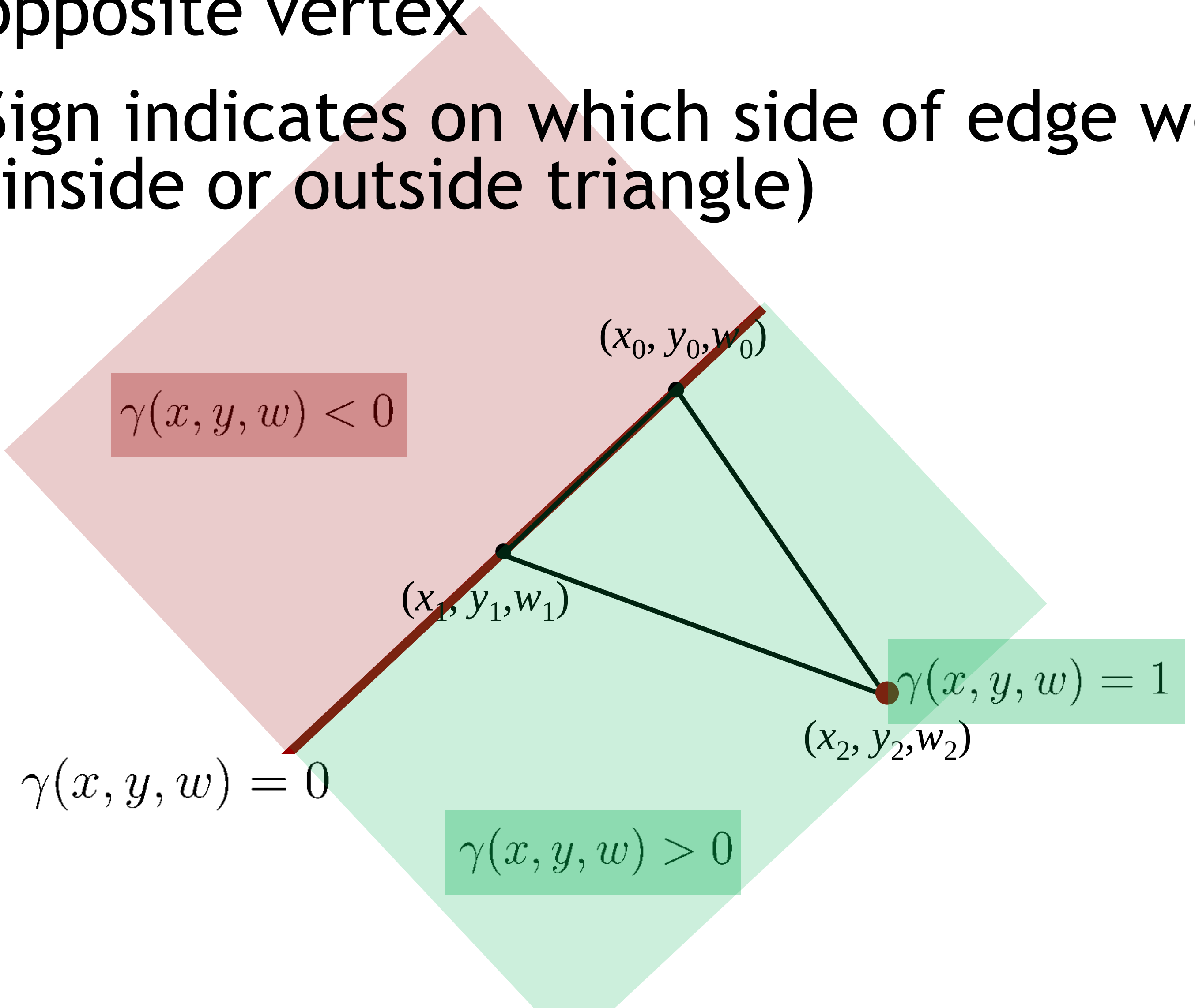
Edge functions

- Edge functions are zero on one edge, one at opposite vertex
- Sign indicates on which side of edge we are (inside or outside triangle)



Edge functions

- Edge functions are zero on one edge, one at opposite vertex
- Sign indicates on which side of edge we are (inside or outside triangle)

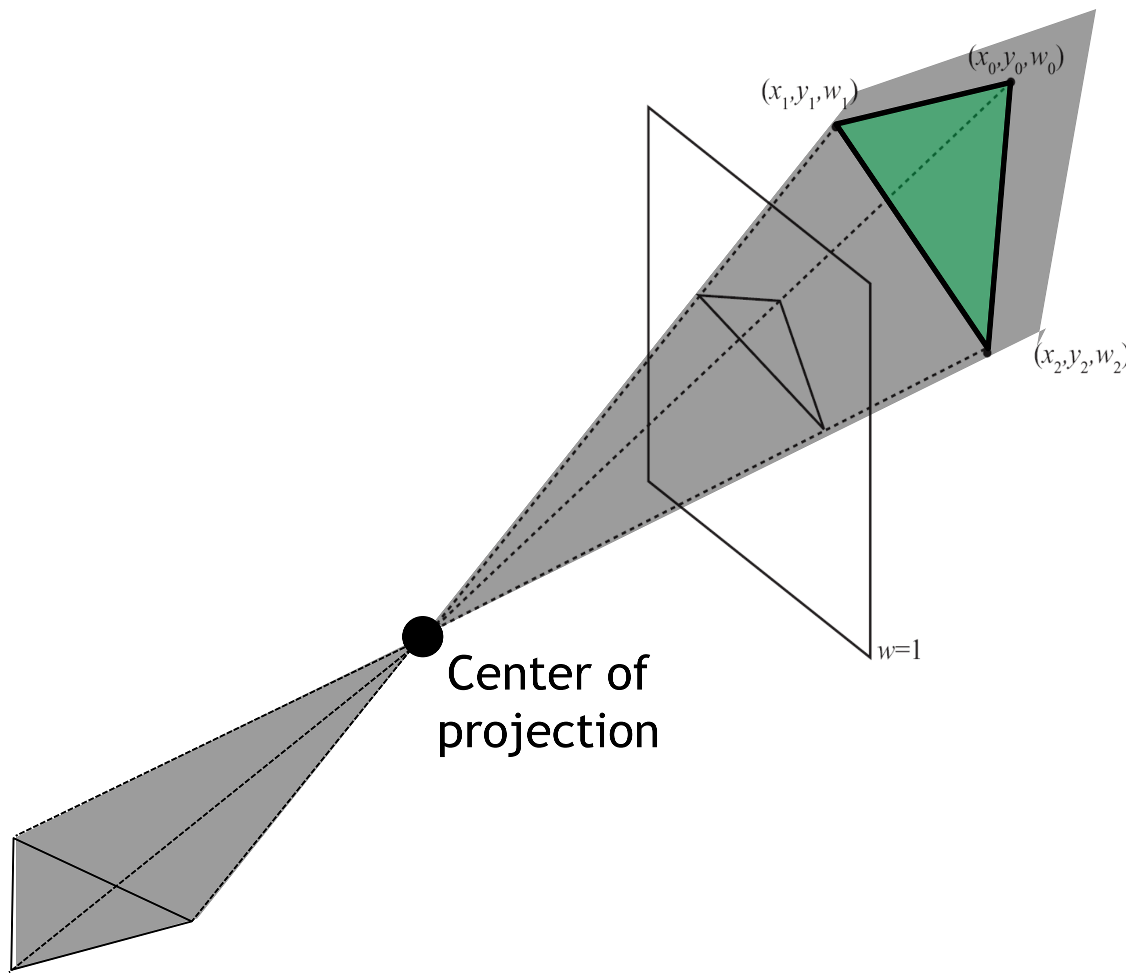


Edge functions

- Functions α, β, γ are also called barycentric coordinates
- Functions are defined for any point *x*,*y*,*w*, not only on plane of triangle!
- Points *x*,*y*,*w* on plane defined by triangle have $\alpha(x, y, w) + \beta(x, y, w) + \gamma(x, y, w) = 1$
- Points inside the triangle have $0 < \alpha, \beta, \gamma < 1$

Edge functions

- Points inside double pyramid spanned by triangle and center of projection: $0 < \alpha, \beta, \gamma$



Edge functions

- Linear functions have form

$$\begin{aligned}\alpha(x,y,w) &= a_\alpha x + b_\alpha y + c_\alpha w\\ \beta(x,y,w) &= a_\beta x + b_\beta y + c_\beta w\\ \gamma(x,y,w) &= a_\gamma x + b_\gamma y + c_\gamma w\end{aligned}$$

- Need to determine coefficients $a_\alpha, b_\alpha, c_\alpha, \ldots$
- Using interpolation constraints
  (zero on one edge, one at opposite vertex)

$$\begin{bmatrix} x_0 & y_0 & w_0 \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{bmatrix} \begin{bmatrix} a_\alpha & a_\beta & a_\gamma \\ b_\alpha & b_\beta & b_\gamma \\ c_\alpha & c_\beta & c_\gamma \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Finding coefficients

- Determine coefficients using interpolation constraints

$$
\begin{array}{ccc}
\text{Known} & \text{Unknown} & \\
\begin{bmatrix} x_0 & y_0 & w_0 \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{bmatrix} &
\begin{bmatrix} a_\alpha & a_\beta & a_\gamma \\ b_\alpha & b_\beta & b_\gamma \\ c_\alpha & c_\beta & c_\gamma \end{bmatrix} &
= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}
\end{array}
$$

$$\alpha(x_0, y_0, w_0) = 1$$

α needs to be 1 on vertex 0

Finding coefficients

- Determine coefficients using interpolation constraints

$$\begin{array}{c} \text{Known} \\ \begin{bmatrix} x_0 & y_0 & w_0 \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{bmatrix} \end{array} \begin{array}{c} \text{Unknown} \\ \begin{bmatrix} a_\alpha & a_\beta & a_\gamma \\ b_\alpha & b_\beta & b_\gamma \\ c_\alpha & c_\beta & c_\gamma \end{bmatrix} \end{array} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\alpha(x_1, y_1, w_1) = 0$$

α needs to be 0 on vertex 1

Finding coefficients

- Determine coefficients using interpolation constraints

$$\begin{array}{ccc} \text{Known} & \text{Unknown} & \\ \begin{bmatrix} x_0 & y_0 & w_0 \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{bmatrix} & \begin{bmatrix} a_\alpha & a_\beta & a_\gamma \\ b_\alpha & b_\beta & b_\gamma \\ c_\alpha & c_\beta & c_\gamma \end{bmatrix} & = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{array}$$

$$\gamma(x_1, y_1, w_1) = 0$$

γ needs to be 0 on vertex 1

Etc., matrix equation encodes 9 constraints necessary to determine coefficients

Finding coefficients

- Determine coefficients using interpolation constraints

$$\begin{array}{cc} \text{Known} & \text{Unknown} \\ \begin{bmatrix} x_0 & y_0 & w_0 \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{bmatrix} & \begin{bmatrix} a_\alpha & a_\beta & a_\gamma \\ b_\alpha & b_\beta & b_\gamma \\ c_\alpha & c_\beta & c_\gamma \end{bmatrix} \end{array} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

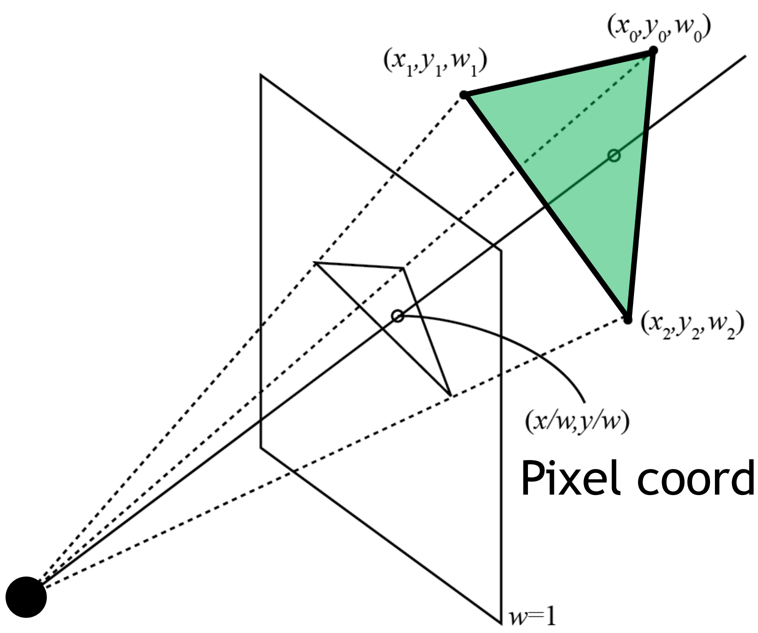
- Matrix inversion to solve for coefficients

$$\begin{bmatrix} a_\alpha & a_\beta & a_\gamma \\ b_\alpha & b_\beta & b_\gamma \\ c_\alpha & c_\beta & c_\gamma \end{bmatrix} = \begin{bmatrix} x_0 & y_0 & w_0 \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{bmatrix}^{-1}$$

Pixel inside/outside test

- Our question: Are pixel coordinates $(x/w, y/w)$ inside or outside projected triangle?
- Homogeneous division applied to edge functions

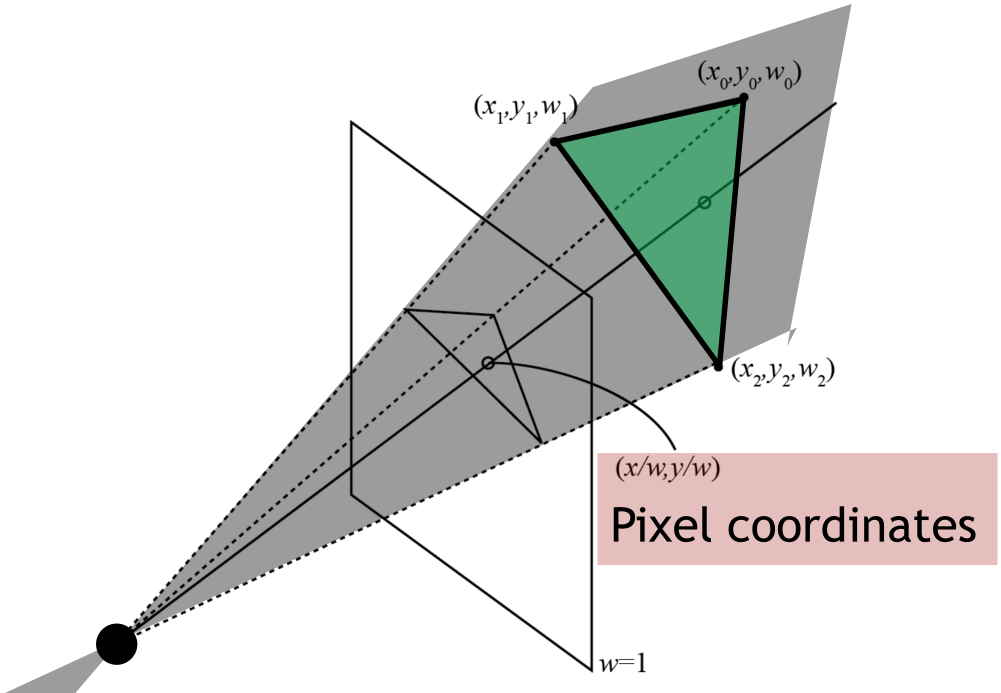
$$\alpha/w = a_\alpha(x/w) + b_\alpha(y/w) + c_\alpha$$
$$\beta/w = a_\beta(x/w) + b_\beta(y/w) + c_\beta$$
$$\gamma/w = a_\gamma(x/w) + b_\gamma(y/w) + c_\gamma$$



Functions of pixel coordinates! $(x/w, y/w)$!

Pixel inside/outside test

- Pixel is inside if $0 < \alpha/w, \beta/w, \gamma/w$
- Pixel is inside, but behind eye (w negative) if $0 > \alpha/w, \beta/w, \gamma/w$
- Intuitively, test if pixel in double pyramid



Pixel inside/outside test

- Trick
  - Evaluate edge equations using pixel coordinates $(x/w, y/w)$
  - Result we get is $\alpha/w,\ \beta/w,\ \gamma/w$
  - Can still determine inside outside based on signs of $\alpha/w,\ \beta/w,\ \gamma/w$
- Main benefits
  - Division by w is not actually computed, no division by 0 problem
  - No need for clipping

Summary

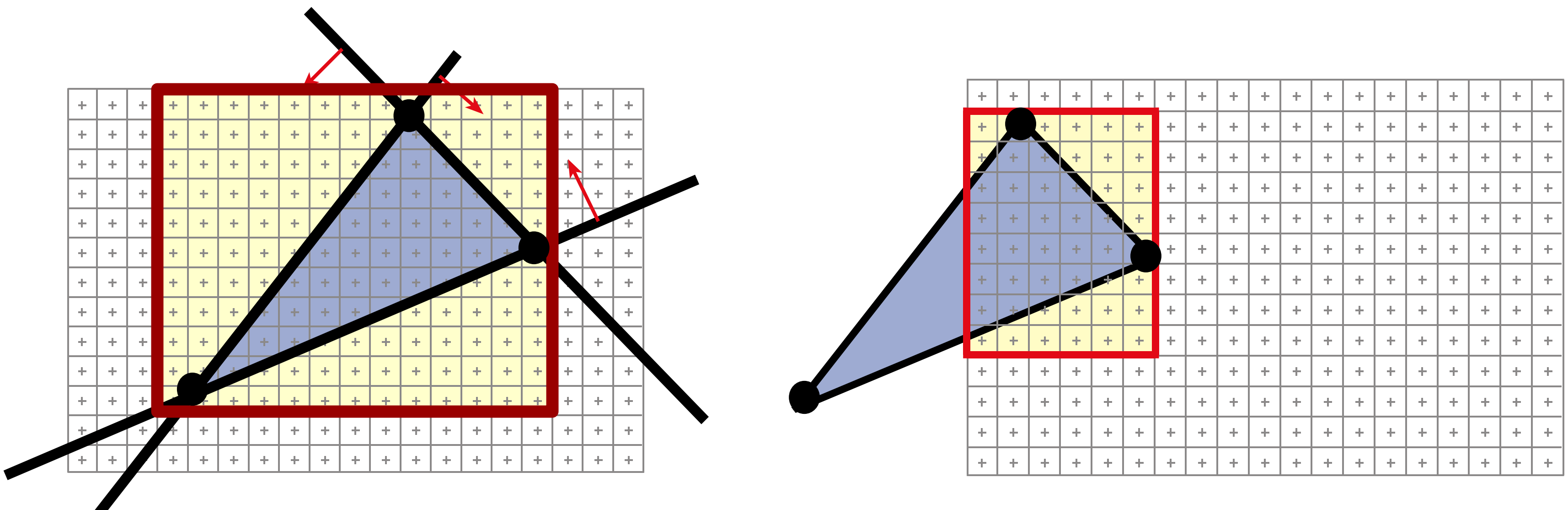
- Triangle setup
  - Compute coefficients for edge functions $a_\alpha, \ldots$ using 3x3 matrix inversion
- At each pixel of the image
  - Evaluate $\alpha/w, \beta/w, \gamma/w$ using pixel coordinates (*x/w*,*y/w*)
  - Perform inside test $0 < \alpha/w, \beta/w, \gamma/w$

Open issues

- Matrix to find edge functions may be singular
  - Triangle has zero area before projection
  - Projected triangle has zero area
  - No need to draw triangle in this case
- Determinant may be negative
  - Backfacing triangle
  - Allows backface culling
- Do we really need to test each pixel on the screen?

Binning

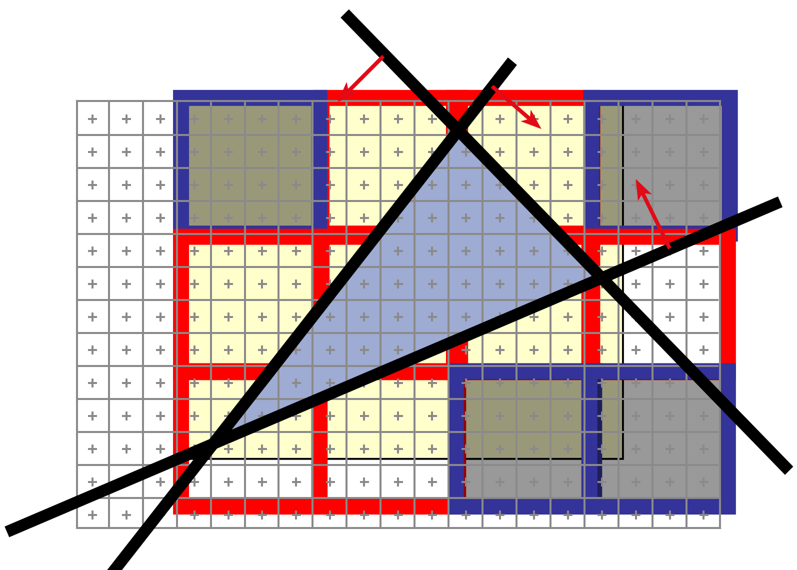
- Try to determine tightly enclosing area for triangle
  - Patent (NVidia) http://www.patentstorm.us/patents/6765575.html
- Simpler but potentially inefficient solution: 3 cases
  1. If all vertices have $w>0$, project them, find axis aligned bounding box, limit extent to image boundaries
  2. If all vertices have $w<0$, triangle is behing eye, don‘t draw
  3. Otherwise, don‘t project vertices, test all image pixels (inefficient, but happens rarely)



Axis aligned bounding boxes based on projected vertices

Improvement

- If block of n x n pixels is outside triangle, discard whole block, no need to test individual pixels
- Conservative test
  - Never discard a block that intersects the triangle
  - May still test pixels of some blocks that are outisde triangle, but most of them are discarded
- How?



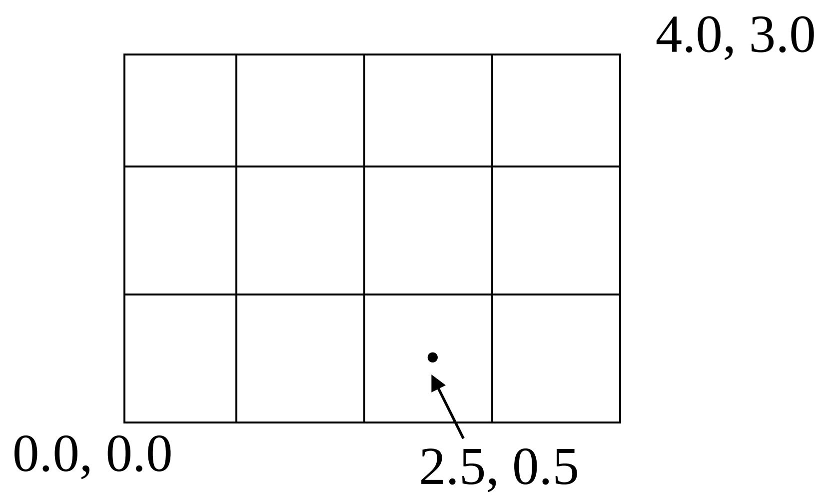
4 x 4 Blocks

Further improvement

- Can have hierarchy of blocks, usually two levels
- Find right size of blocks for best performance (experimentally)
  - Fixed number of pixels per block, e.g., 4x4 pixels

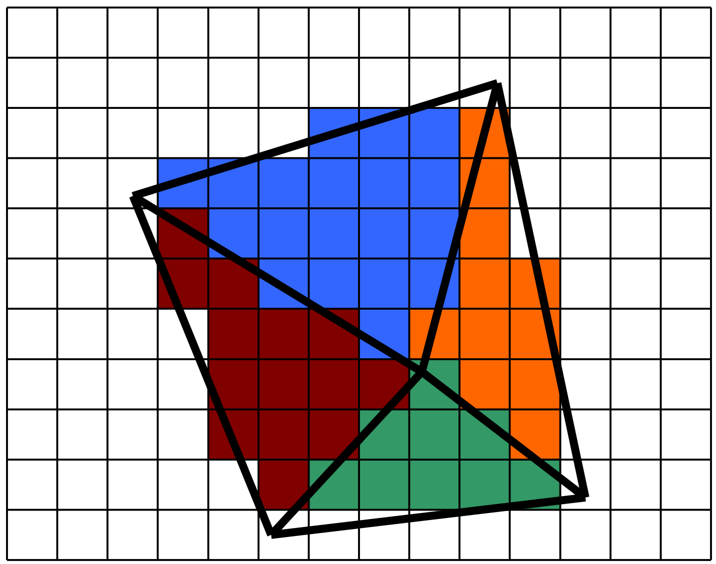
Where is the center of a pixel?

- Depends on conventions
- With our viewport transformation from last lecture
  - 4 x 3 pixels ⇔ viewport coordinates are in [0…4]x[0…3]
  - Center of lower left pixel is 0.5, 0.5
  - Center of upper right pixel is 3.5, 2.5



Shared edges

- Each pixel needs to be rasterized exactly once
- Result image is independent of drawing order
- Rule: If pixel center exactly touches an edge or vertex
  - Fill pixel only if triangle extends to the right



Implementation optimizations

- Performance of rasterizer is crucial, since it‘s „inner loop“ of renderer
- CPU: performance optimizations
  - Integer arithmetic
  - Incremental calculations
  - Multi-threading
  - Vector operations (SSE instructions)
  - Use C/C++ or assembler
- GPU: hardwired!

Today

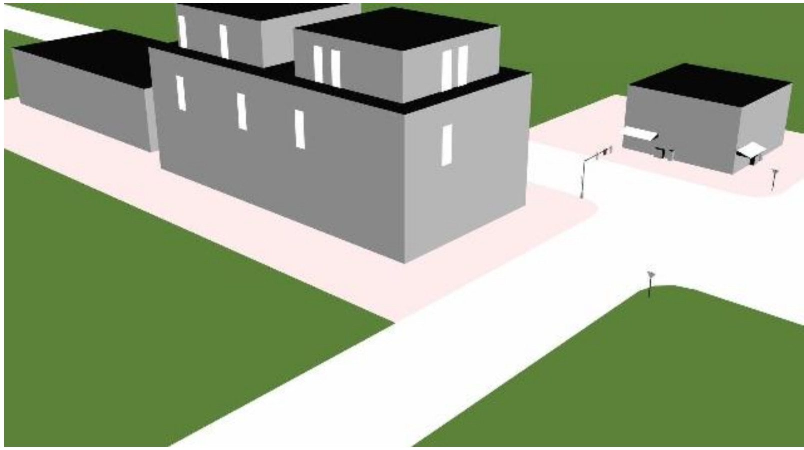
Drawing triangles

- Homogeneous rasterization
- Texture mapping
- Perspective correct interpolation
- Visibility

Large triangles

Pros

- Often ok for simple geometry
- Fast to render



Cons

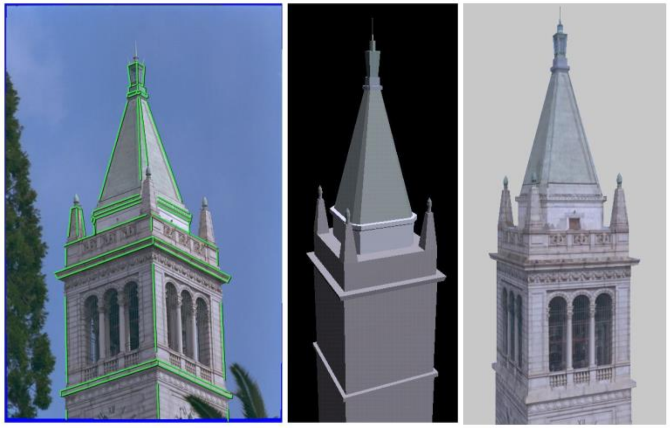
- Per vertex colors look bad
- Need more interesting surfaces
  - Detailed color variation, small scale bumps, roughness
- Ideas?

Texture mapping

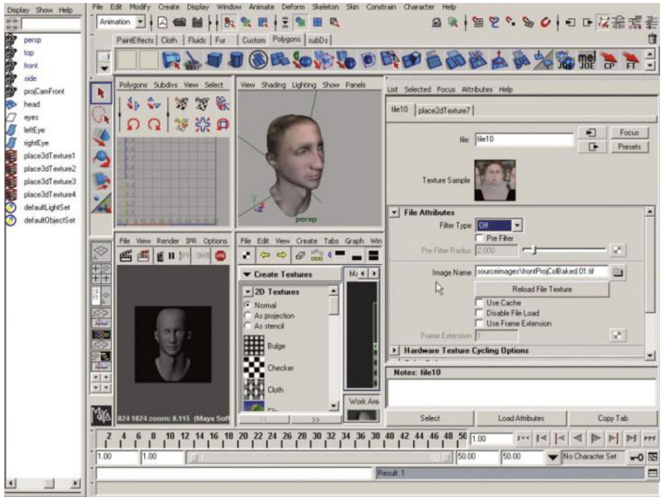
- Glue textures (images) onto surfaces
- Same triangles, much more interesting appearance
- Think of colors as reflectance coefficients
  - How much light is reflected for each color
  - More later in course

Creating textures

- Photographs
- Paint directly on surfaces using modeling program
- Stored as image files



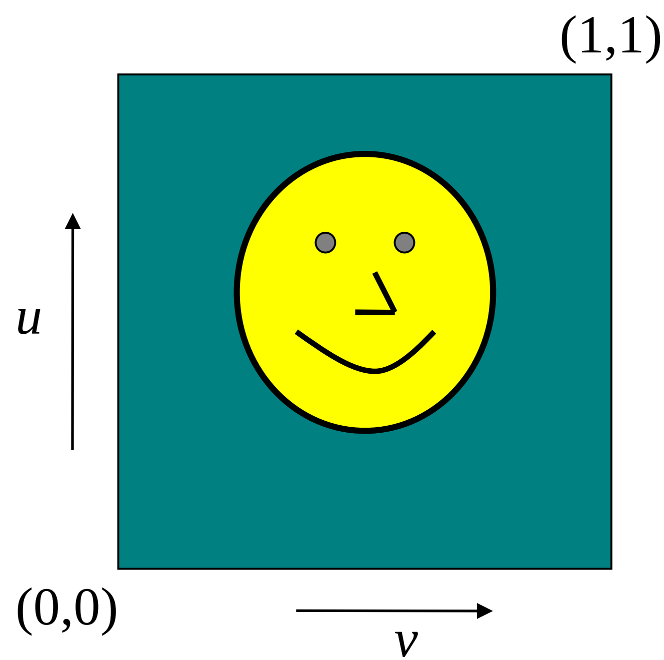
Images by Paul Debevec



Texture painting in Maya

Texture mapping

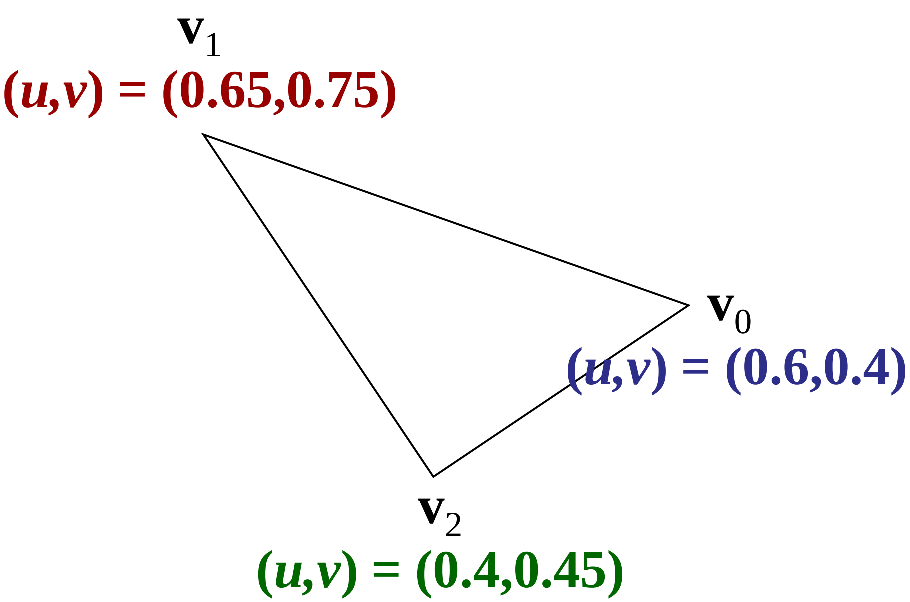
- Goal: assign locations in texture image to locations on 3D geometry
- Introduce texture space
  - Texture pixels (texels) have texture coordinates (u,v)
- Common convention
  - Bottom left corner of texture is $(u,v)=(0,0)$
  - Top right corner is $(u,v)=(1,1)$
  - Requires scaling of (u,v) to access actual texture pixels stored in 2D array



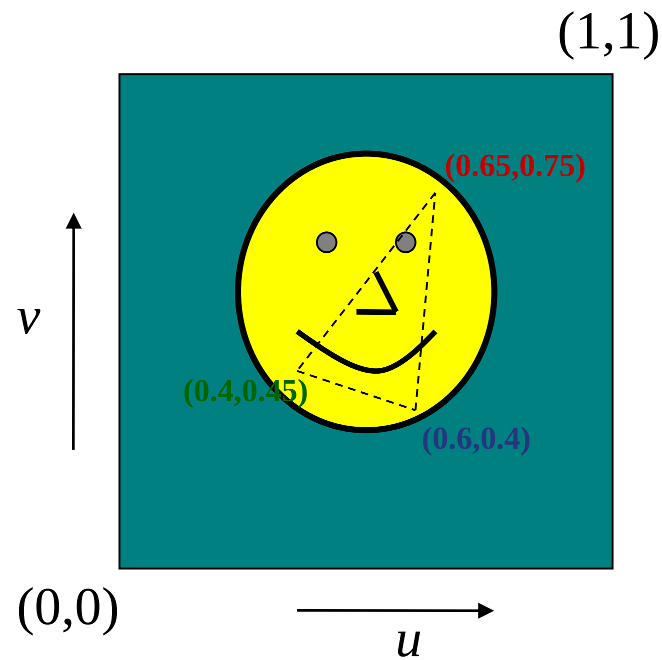
Texture space

Texture mapping

- Store texture coordinates at each triangle vertex



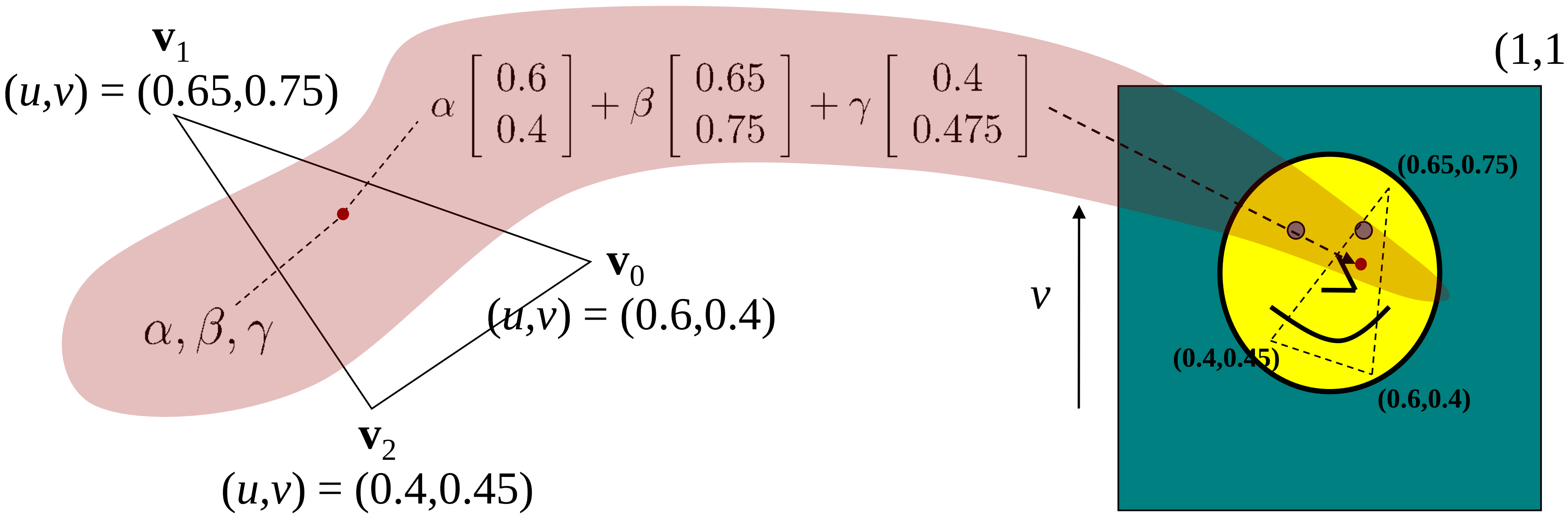
Triangle (in any space before projection)



Texture space

Texture mapping

- Each point on triangle has barycentric coordinates with $0<\alpha,\beta,\gamma,\ \alpha+\beta+\gamma=1$
- Use barycentric coordinates to interpolate texture coordinates

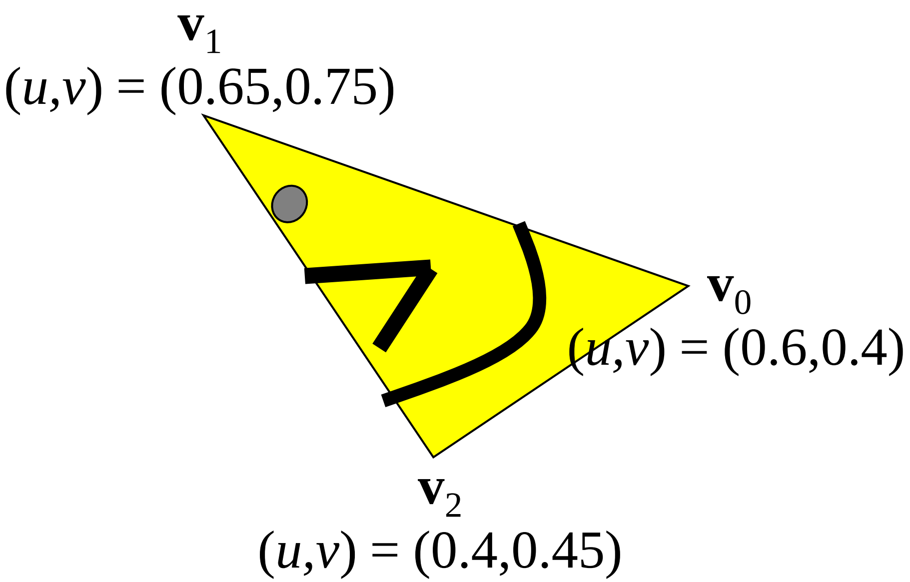


Triangle (in any space before projection)

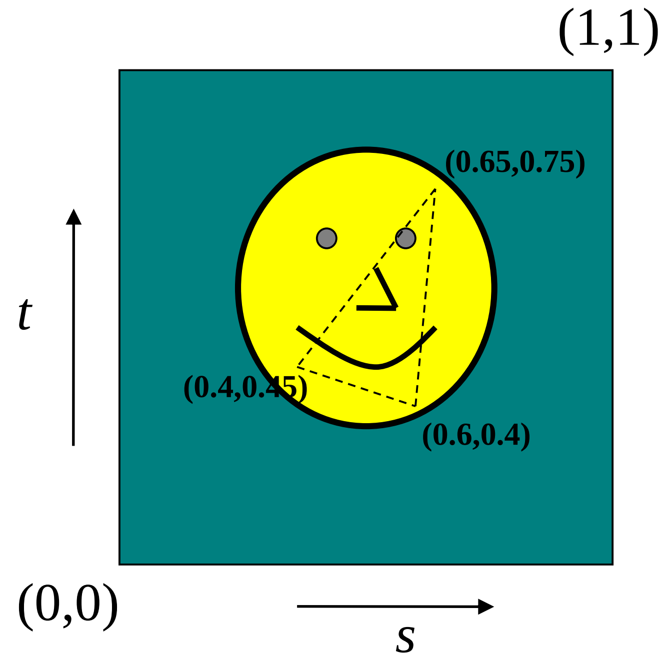
Texture space

Texture mapping

- Each point on triangle has corresponding point in texture
- Texture is “glued” on triangle



Triangle (in any space before projection)



Texture space

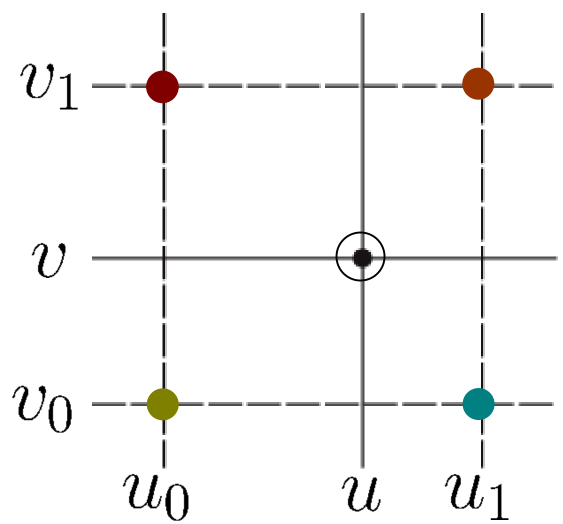
Rendering

- Given
  - Texture coordinates at each vertex
  - Texture image
- At each pixel, interpolate texture coordinates
- Look up corresponding texel
- Paint current pixel with texel color
- All computations on GPU

Texture look-up

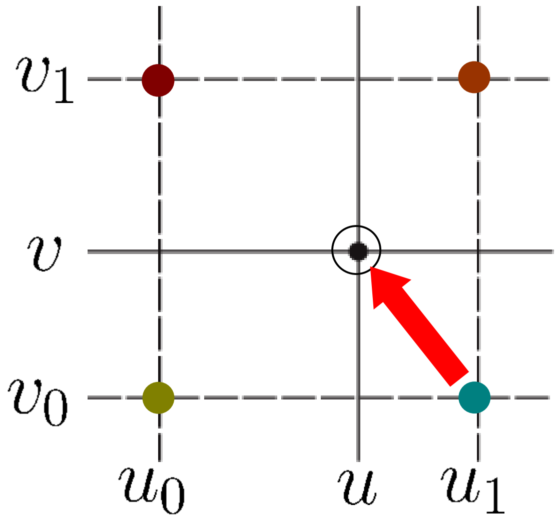
- Given interpolated texture coordinates (u, v) at current pixel
- Closest four texels in texture space are at

  $(u_0, v_0), (u_1, v_0), (u_1, v_0), (u_1, v_1)$
- How to compute color of pixel?



Nearest-neighbor interpolation

- Use color of closest texel



- Simple, but low quality

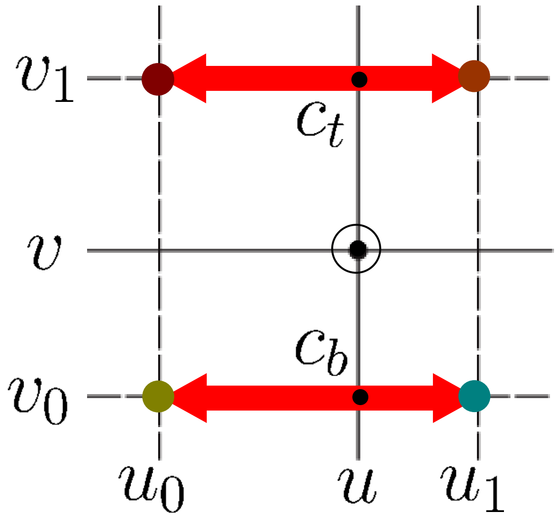
Bilinear interpolation

http://en.wikipedia.org/wiki/Bilinear_interpolation

1. Linear interpolation horizontally

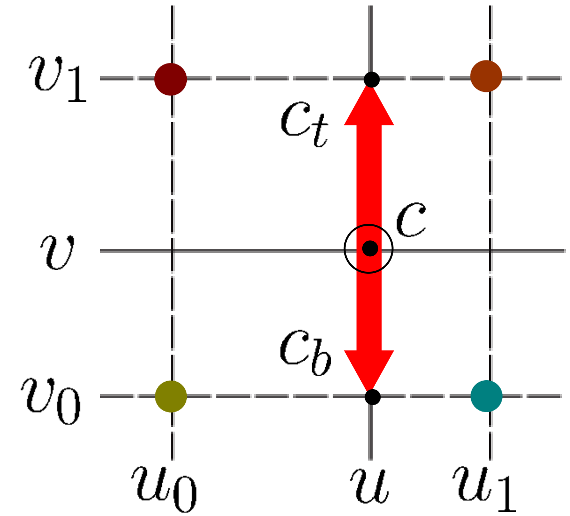
$$w_u = \frac{u - u_0}{u_1 - u_0}$$

$$\begin{aligned} c_b &= tex(u_0, v_0)(1 - w_u) + tex(u_1, v_0)w_u \\ c_t &= tex(u_0, v_1)(1 - w_u) + tex(u_1, v_1)w_u \end{aligned}$$



Bilinear interpolation

http://en.wikipedia.org/wiki/Bilinear_interpolation



1. Linear interpolation horizontally

$$w_u = \frac{u - u_0}{u_1 - u_0}$$

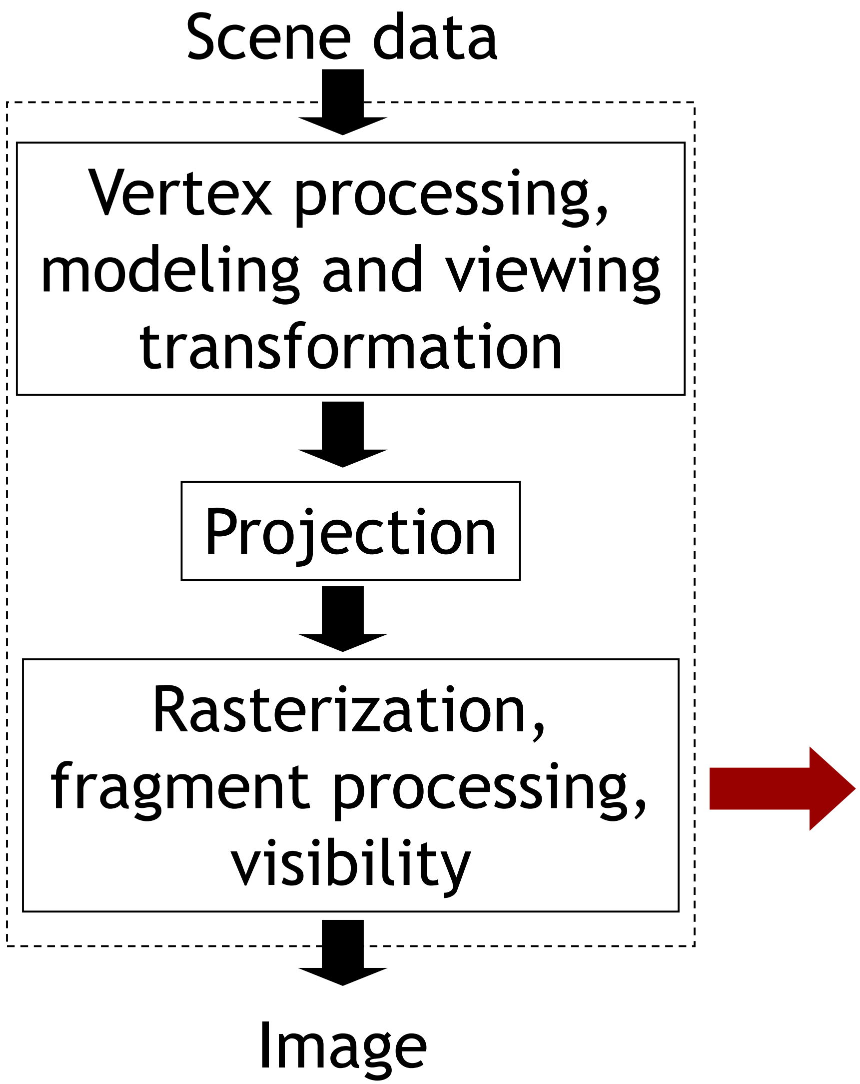
$$\begin{aligned} c_b &= tex(u_0, v_0)(1 - w_u) + tex(u_1, v_0)w_u \\ c_t &= tex(u_0, v_1)(1 - w_u) + tex(u_1, v_1)w_u \end{aligned}$$

2. Linear interpolation vertically

$$w_v = \frac{v - v_0}{v_1 - v_0}$$

$$c = c_b(1 - w_v) + c_t w_v$$

Texture mapping



Fragment processing includes texture mapping
(and shading, later in course)

Today

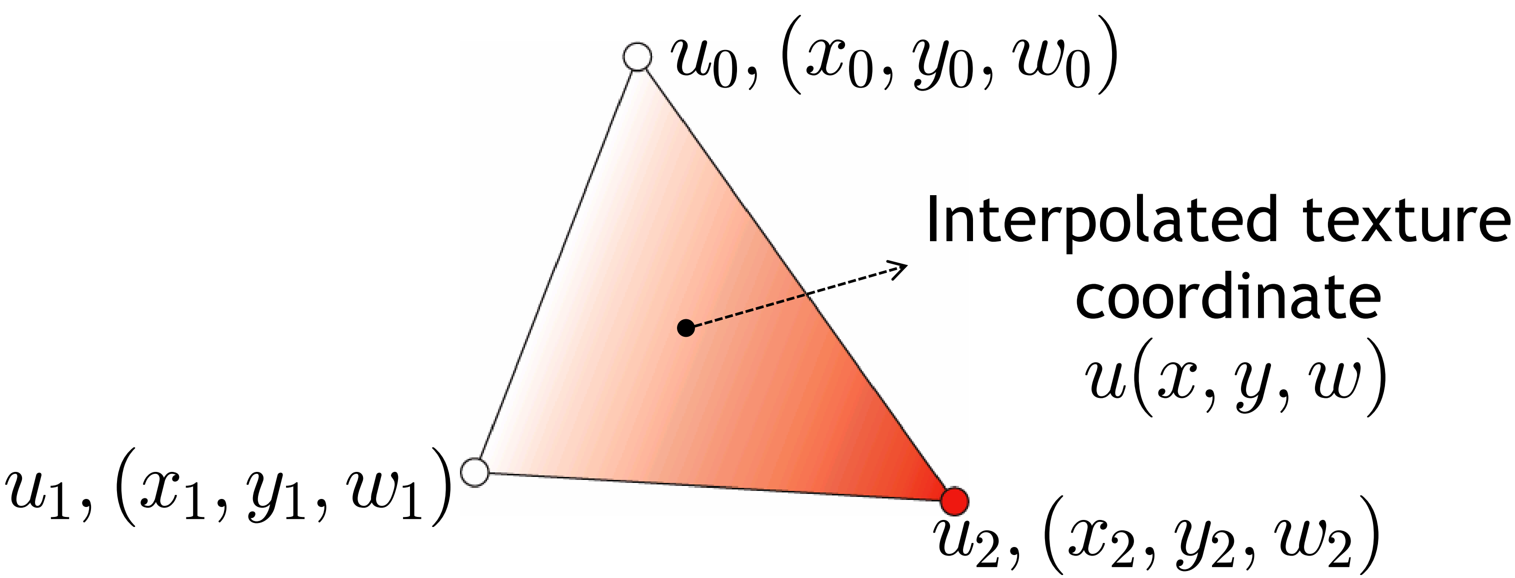
Drawing triangles

- Homogeneous rasterization
- Texture mapping
- Perspective correct interpolation
- Visibility

Attribute interpolation

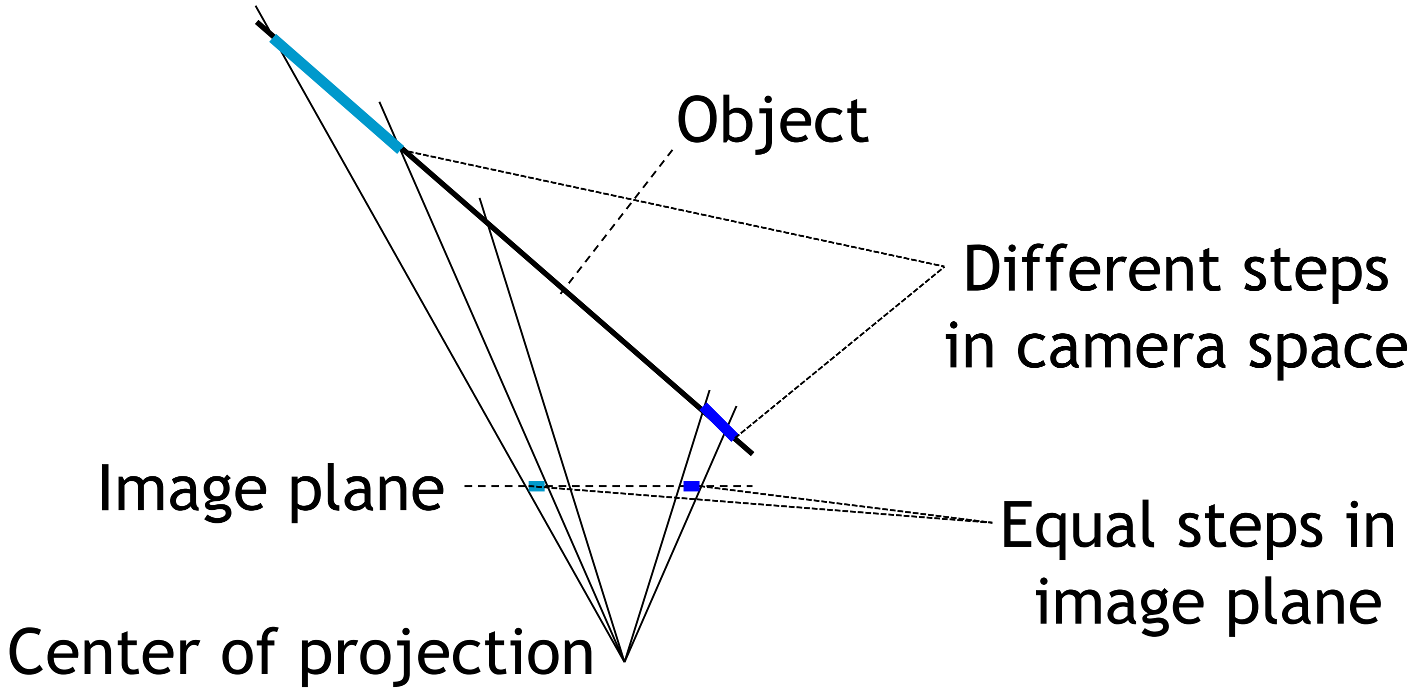
- Rasterizer needs to
  - Determine inside/outside test for each pixel
  - Fill in triangle by interpolating vertex attributes
  - For example (u,v) texture coordinates, color, etc.

Triangle before projection



Observation

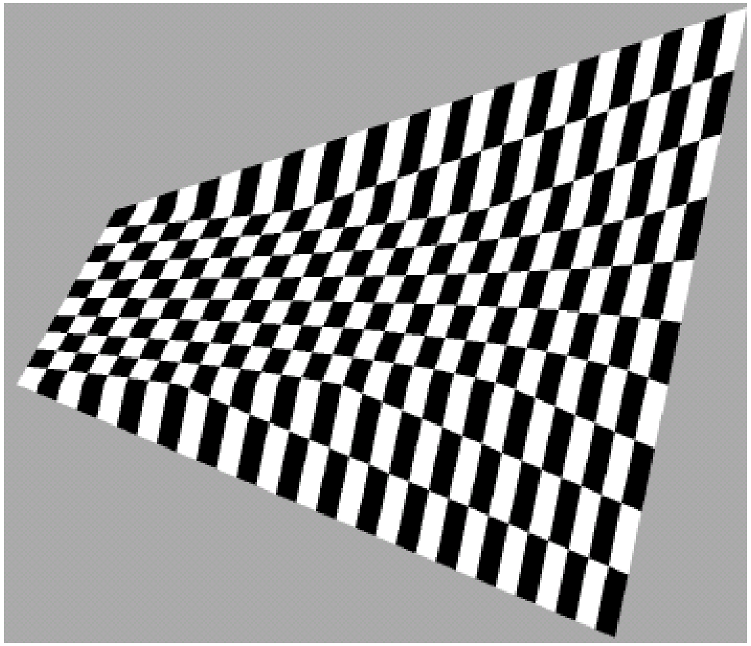
- Linear interpolation in image coordinates does not correspond to linear interpolation in camera space
- “Equal step size on image plane does not correspond to equal step size on object”



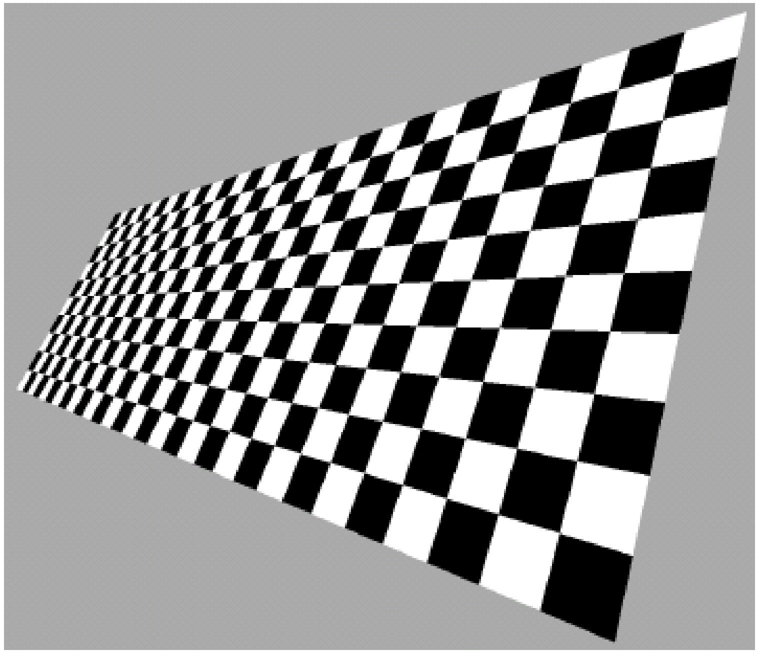
- Perspective correct interpolation: “translate step size in image plane correctly to step size on object”

Perspective correct interpolation

Linear interpolation of texture coordinates on image plane

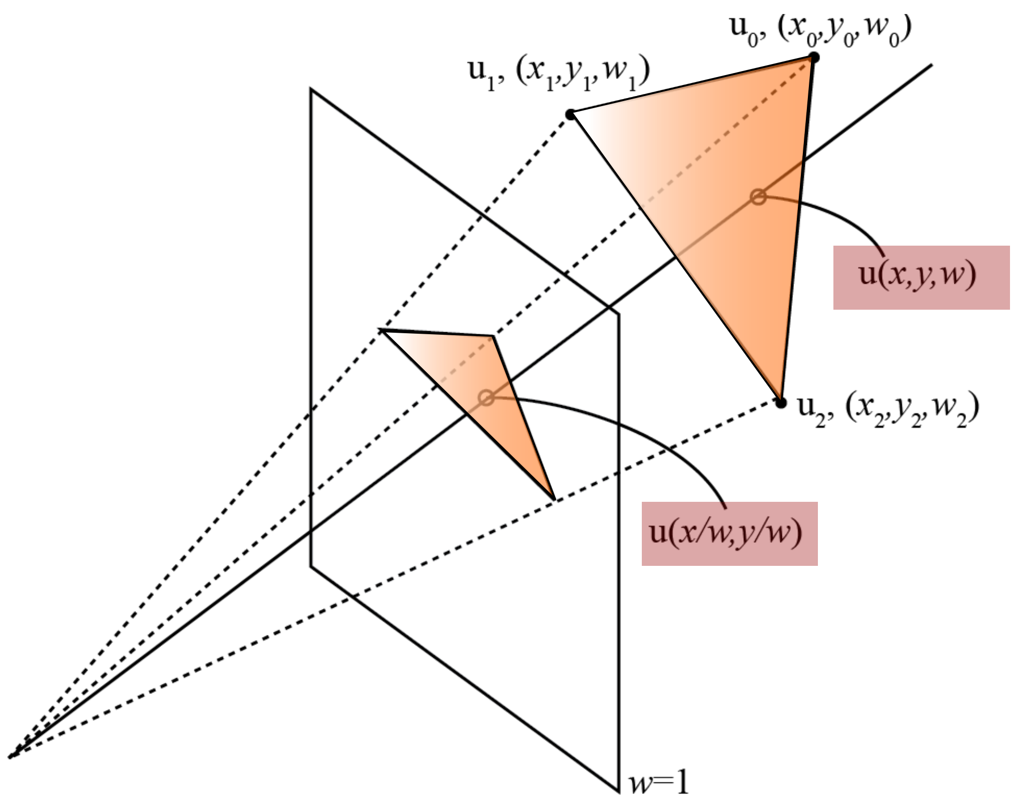


Perspective correct interpolation



Strategy

1. Find linear function $u(x,y,w)$ in 2D homogeneous space that interpolates vertex attribute u
2. Project to pixel coordinates, find function of pixel coordinates $u(x/w,y/w)$



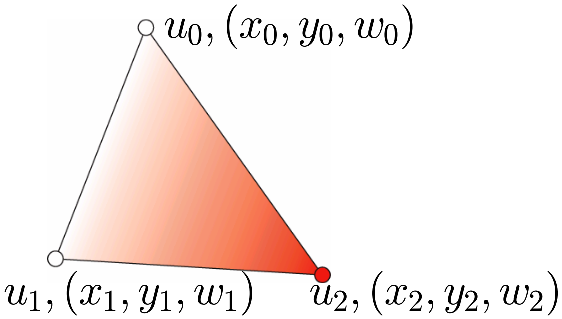
Step 1: 2D homogeneous interp.

- Linear function for vertex attribute u

$$u(x,y,w) = a_u x + b_u y + c_u w$$

- Interpolation constraints (as for edge fncts.)

$$\begin{bmatrix} x_0 & y_0 & w_0 \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{bmatrix} \begin{bmatrix} a_u \\ b_u \\ c_u \end{bmatrix} = \begin{bmatrix} u_0 \\ u_1 \\ u_2 \end{bmatrix}$$



Unknown coefficients

Given u texture coordinate at vertices

$$a_u x_2 + b_u y_2 + c_u w_2 = u_2$$

Step 1: 2D homogeneous interp.

- Linear function for vertex attribute u

$$u(x,y,w) = a_u x + b_u y + c_u w$$

$$\begin{bmatrix} x_0 & y_0 & w_0 \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{bmatrix} \begin{bmatrix} a_u \\ b_u \\ c_u \end{bmatrix} = \begin{bmatrix} u_0 \\ u_1 \\ u_2 \end{bmatrix}$$

Given vertex coordinates — Unknown — Given texture coordinates

- Same matrix inversion to find coefficients

$$\begin{bmatrix} a_u \\ b_u \\ c_u \end{bmatrix} = \begin{bmatrix} x_0 & y_0 & w_0 \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{bmatrix}^{-1} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \end{bmatrix}$$

Step 2: projection to pixel coord.

- Homogeneous division yields function of pixel coordinates

$$u/w = a_u(x/w) + b_u(y/w) + c_u$$

- But: we need u, not u/w as function of pixels x/w, y/w

- Trick: get coefficients of constant function

$$1 \equiv a_1 x + b_1 y + c_1 w \qquad \begin{bmatrix} x_0 & y_0 & w_0 \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{bmatrix} \begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

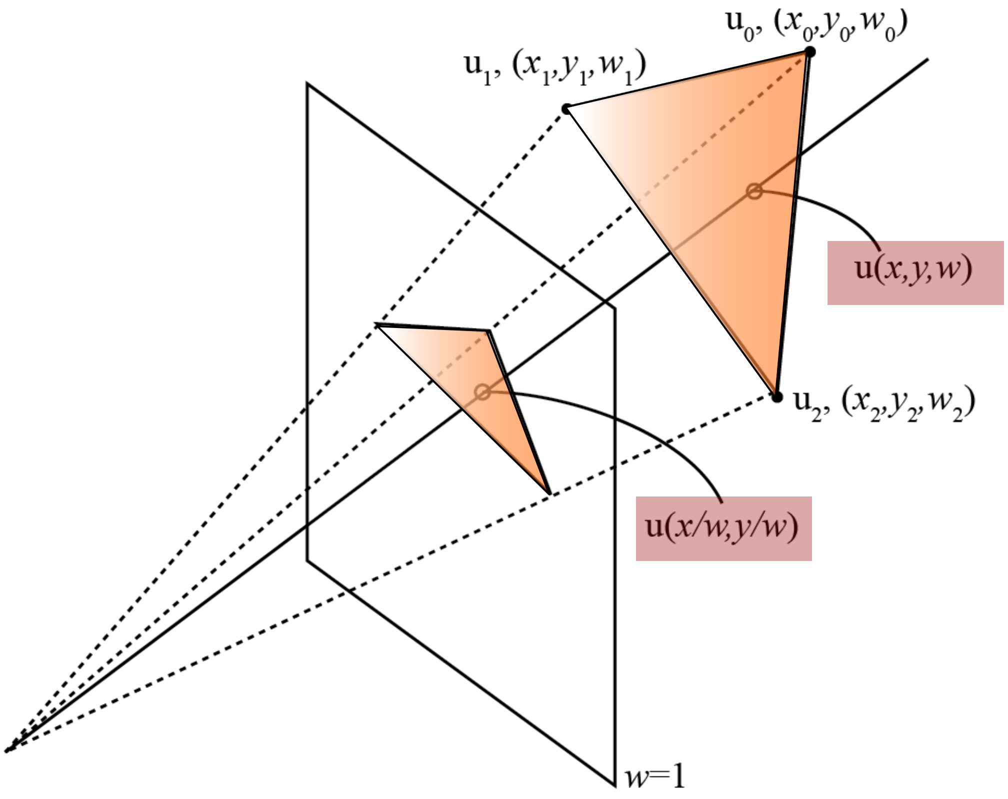
- Homogeneous division

$$1/w = a_1(x/w) + b_1(y/w) + c_1$$

Step 2: projection to pixel coord.

- Finally

$$u(x/w, y/w) = \frac{(u/w)}{(1/w)}$$



Summary

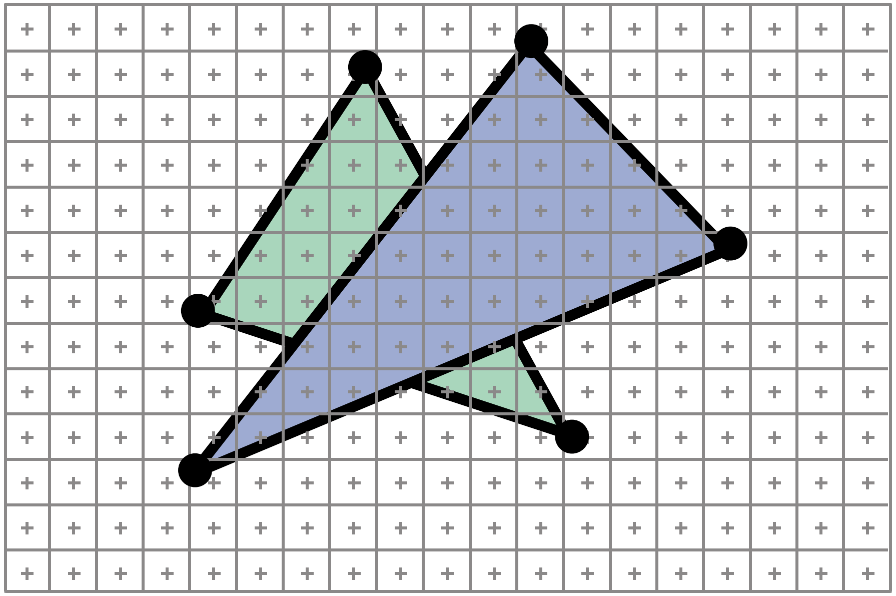
- Triangle setup
  - Invert 3x3 matrix
  - Compute coefficients for edge functions $a_\alpha, \ldots$, attribute functions $a_u, \ldots$, constant fnct. $a_1, \ldots$
  - Requires 3x3 matrix-vector multiplication each
- At each pixel (*x/w*,*y/w*)
  - Linearly interpolate *1/w*
  - For each attribute function
    - Linearly interpolate *function/w*
    - Divide (*function/w*)/(*1/w*)

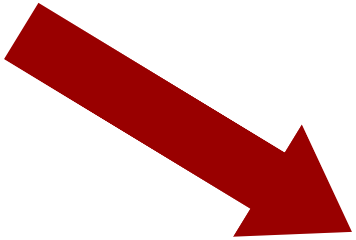
Today

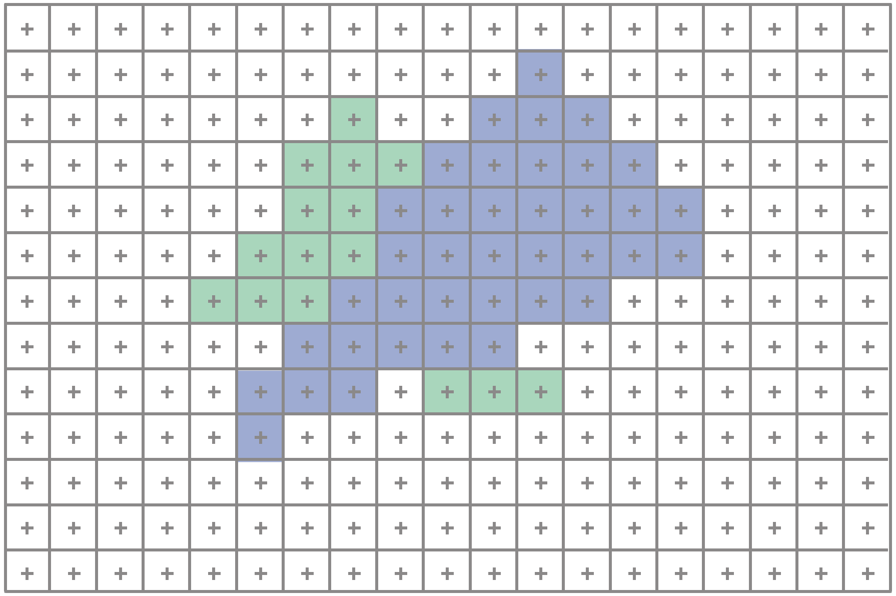
Drawing triangles

- Homogeneous rasterization
- Texture mapping
- Perspective correct interpolation
- Visibility

Visibility





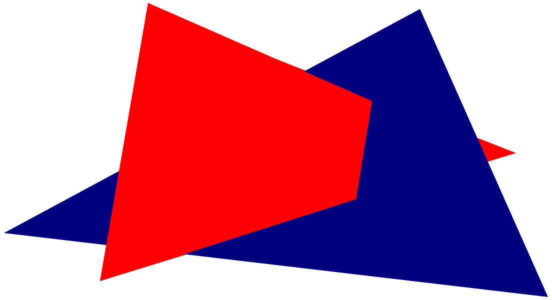


- At each pixel, need to determine which triangle is visible

Painter's algorithm

http://en.wikipedia.org/wiki/Painter's_algorithm

- Paint from back to front
- Every new pixel always paints over previous pixel
- Need to sort geometry according to depth
- May need to split triangles if they intersect



- Old style, before memory became cheap

Z-buffering

http://en.wikipedia.org/wiki/Z-buffering

- Store "depth" at each pixel
  - Store $1/w$ because we compute it for rasterization already
- Depth test
  - During rasterization, compare stored value to new value
  - Update pixel only if new $1/w$ value is larger

```
setpixel(int x, int y, color c, float w)
if((1/w)>zbuffer(x,y)) then
  zbuffer(x,y) = (1/w)
  color(x,y) = c
```

- In graphics hardware, z-buffer is dedicated memory reserved for GPU (graphics memory)
- Depth test is performed by GPU

Next time

- Color