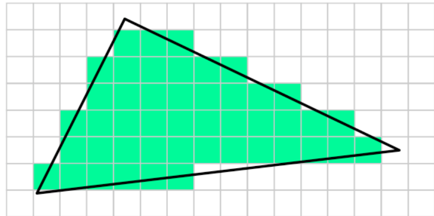
Computer Graphics

Lecture 6:

Rasterization, Visibility & Anti-aliasing

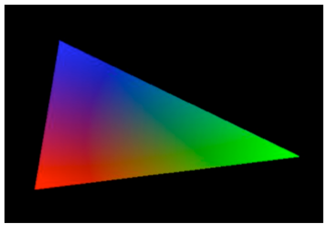
Rasterization

- Determine which pixels are drawn into the framebuffer
- Interpolate parameters (colors, texture coordinates, etc.)



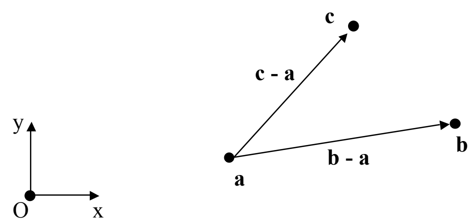
Rasterization

- What does interpolation mean?
- Examples: Colors, normals, shading, texture coordinates



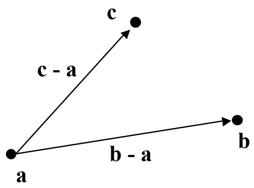
A triangle in terms of vectors

- 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

- The three non-collinear points determine a plane

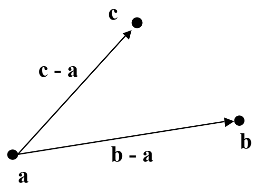


- Example: The vertices a, b and c determine a plane
- The vectors b-a and c-a form a basis for this plane

Basis vectors

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

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



Barycentric coordinates

- We can reorder the terms of the equation:

$$\begin{aligned}\mathbf{p} &= \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}) \\ &= (1 - \beta - \gamma)\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c} \\ &= \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}\end{aligned}$$

- In other words:

$$\mathbf{p}(\alpha,\beta,\gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$

- with

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

- α, β, γ and called barycentric coordinates

Barycentric coordinates

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

$$\mathbf{p}(\alpha,\beta,\gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c} \qquad \alpha + \beta + \gamma = 1$$

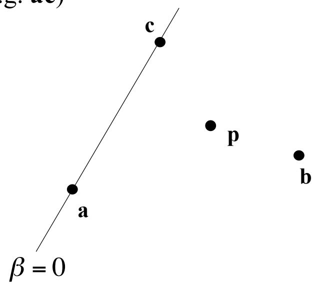
- For any point **p** inside the triangle (**a, b, c**):

$$0 < \alpha < 1$$
$$0 < \beta < 1$$
$$0 < \gamma < 1$$

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

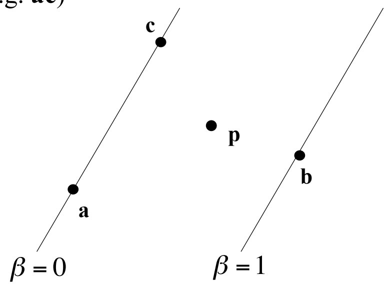
Barycentric coordinates and signed distances

- Let **p** = α**a**+β**b**+γ**c**. Each coordinate (e.g. β) is the signed distance from **p** to the line through a triangle edge (e.g. **ac**)



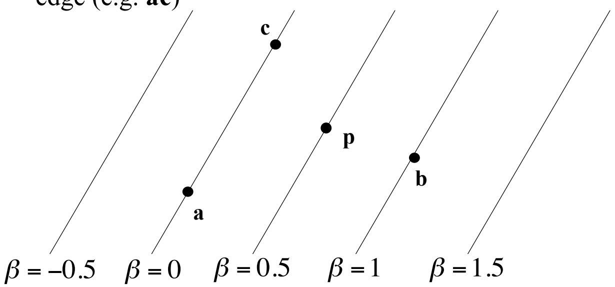
Barycentric coordinates and signed distances

- Let **p** = α**a**+β**b**+γ**c**. Each coordinate (e.g. β) is the signed distance from **p** to the line through a triangle edge (e.g. **ac**)



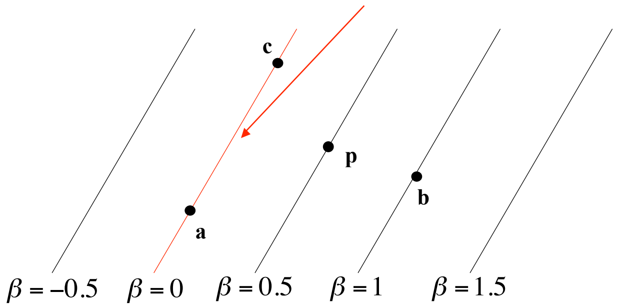
Barycentric coordinates and signed distances

- Let **p** = α**a**+β**b**+γ**c**. Each coordinate (e.g. β) is the signed distance from **p** to the line through a triangle edge (e.g. **ac**)



Barycentric coordinates and signed distances

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



Recall: Implicit equation for lines

- Implicit equation in 2D:

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

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

$$Ax + By + C = 0$$

- Implict line through two points (x_a, y_a) and (x_a, y_a)

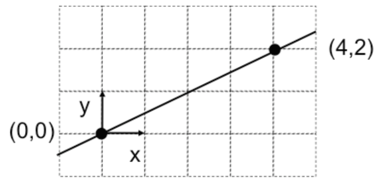
$$(y_a - y_b)x + (x_b - x_a)y + x_a y_b - x_b y_a = 0$$

Implicit equation for lines: Example

A =

B =

C =

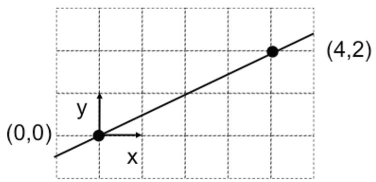


Implicit equation for lines: Example

Solution 1: -2x + 4y = 0

Solution 2: 2x - 4y = 0

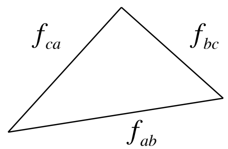
$$kf(x,y) = 0 \quad \text{for any } k$$



Edge equations

- Given a triangle with vertices (x_a,y_a), (x_b,y_b), and (x_c,y_2).
- The line equations of the edges of the triangle are:

$$f_{ab}(x,y) = (y_a - y_b)x + (x_b - x_a)y + x_a y_b - x_b y_a$$
$$f_{bc}(x,y) = (y_b - y_c)x + (x_c - x_b)y + x_b y_c - x_c y_a$$
$$f_{ca}(x,y) = (y_c - y_a)x + (x_a - x_c)y + x_c y_a - x_a y_c$$



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: $kf_{ac}(x,y)=\beta$
- We know that β = 1 at point **b**:

$$kf_{ac}(x,y)=1 \Leftrightarrow k=\frac{1}{f_{ac}(x_b,y_b)}$$

- The barycentric coordinate β for point **p** is:

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

Barycentric Coordinates

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

$$\alpha=\frac{f_{bc}(x,y)}{f_{bc}(x_c,y_c)} \qquad \beta=\frac{f_{ac}(x,y)}{f_{ac}(x_b,y_b)} \qquad \gamma=1-\alpha-\beta$$

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

Triangle Rasterization

- Many different ways to generate fragments for a triangle
- Checking (α, β, γ) is one method, e.g.

  (0< α <1 && 0< β <1 && 0 < γ <1)
- In practice, the graphics hardware use optimized methods:
  - fixed point precision (not floating-point)
  - incremental (use results from previous pixel)

Triangle Rasterization

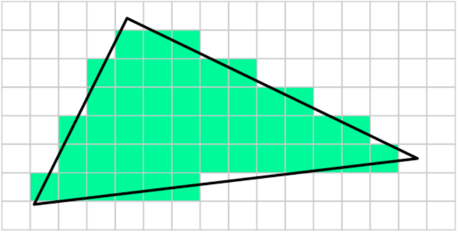
- We can use barycentric coordinates to rasterize and color triangles

```
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

Visibility: One triangle

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

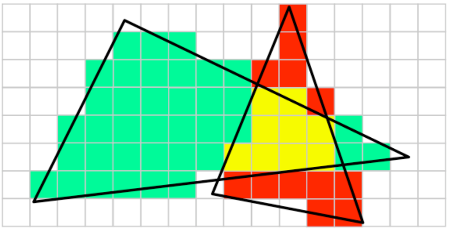


Hidden Surface Removal

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

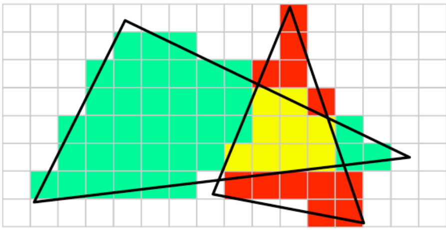
Visibility: Two triangles

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



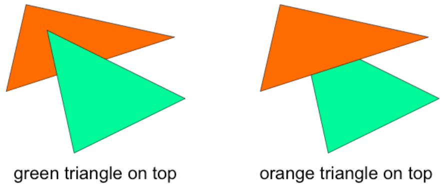
Visibility: Pixels vs Fragments

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



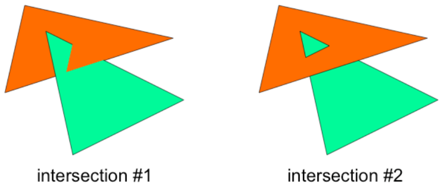
Visibility: Which triangle should be drawn first?

- Two possible cases:



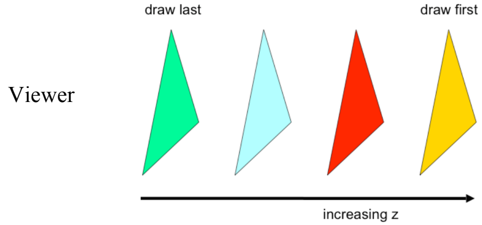
Visibility: Which triangle should be drawn first?

- Many other cases possible!



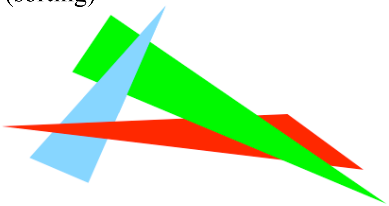
Visibility: Painter's Algorithm

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



Visibility: Painter's Algorithm - Problems

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

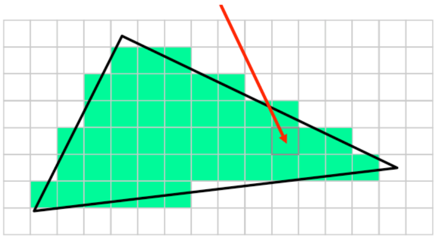


The Depth Buffer (Z-Buffer)

- 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

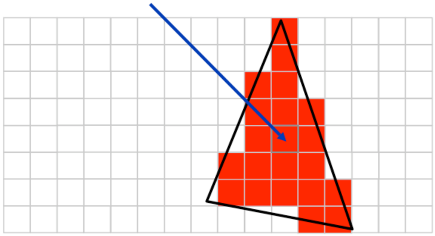
The Depth Buffer (Z-Buffer)

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



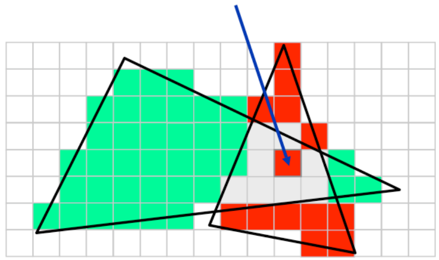
The Depth Buffer (Z-Buffer)

- 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 Depth Buffer (Z-Buffer)

- Many 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

The Z-Buffer Algorithm

```
• Let CB be color (frame) 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
```

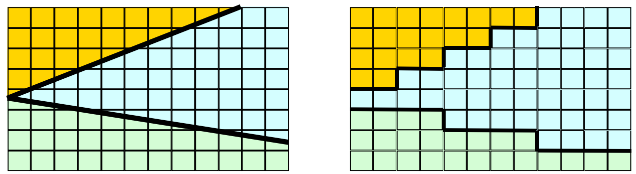

Z-buffer Algorithm Properties

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

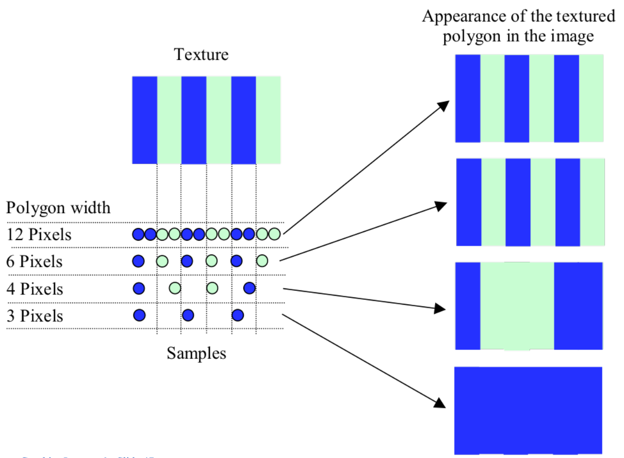
Alias Effects

- One major problem with rasterization is called alias effects, e.g straight lines or triangle boundaries look jagged
- These are caused by undersampling, and can cause unreal visual artefacts.
- It also occurs in texture mapping

Alias Effects at straight boundaries in raster images.



Desired Boundaries　　Pixels Set

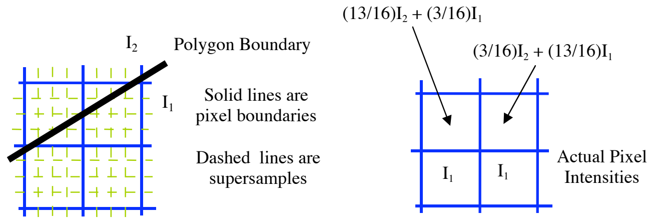


Anti-Aliasing

- The solution to aliasing problems is to apply a degree of blurring to the boundary such that the effect is reduced.
- The most successful technique is called Supersampling

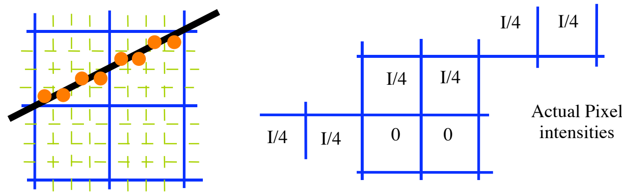
Supersampling

- The basic idea is to compute the picture at a higher resolution to that of the display area.
- Supersamples are averaged to find the pixel value.
- This has the effect of blurring boundaries, but leaving coherent areas of colour unchanged



Limitations of Supersampling

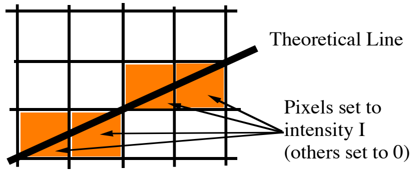
- Supersampling works well for scenes made up of filled polygons.
- However, it does require a lot of extra computation.
- It does not work for line drawings.



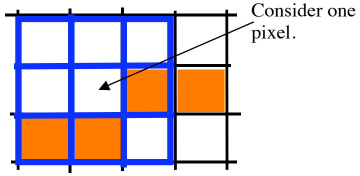
Convolution filtering

- The more common (and much faster) way of dealing with alias effects is to use a ‘filter’ to blur the image.
- This essentially takes an average over a small region around each pixel

For example consider the image of a line



Treat each pixel of the image



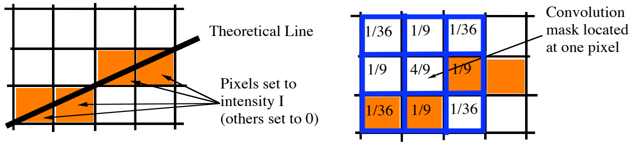
We replace the pixel by a local average, one possibility would be 3*I/9

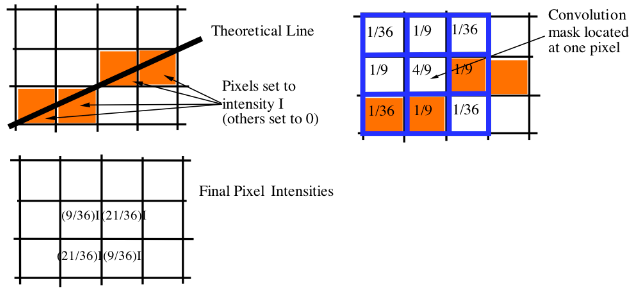
Weighted averages

- Taking a straight local average has undesirable effects.
- Thus we normally use a weighted average.

1/36 *

1	4	1
4	16	4
1	4	1





Pros and Cons of Convolution filtering

- Advantages:
  - It is very fast and can be done in hardware
  - Generally applicable
- Disadvantages:
  - It does degrade the image while enhancing its visual appearance.

Anti-Aliasing textures

- Similar
- When we identify a point in the texture map we return an average of texture map around the point.
- Scaling needs to be applied so that the less the samples taken the bigger the local area where averaging is done.