Triangle Rasterization

Computer Graphics
COMP 770 (236)
Spring 2007

Instructor: Brandon Lloyd

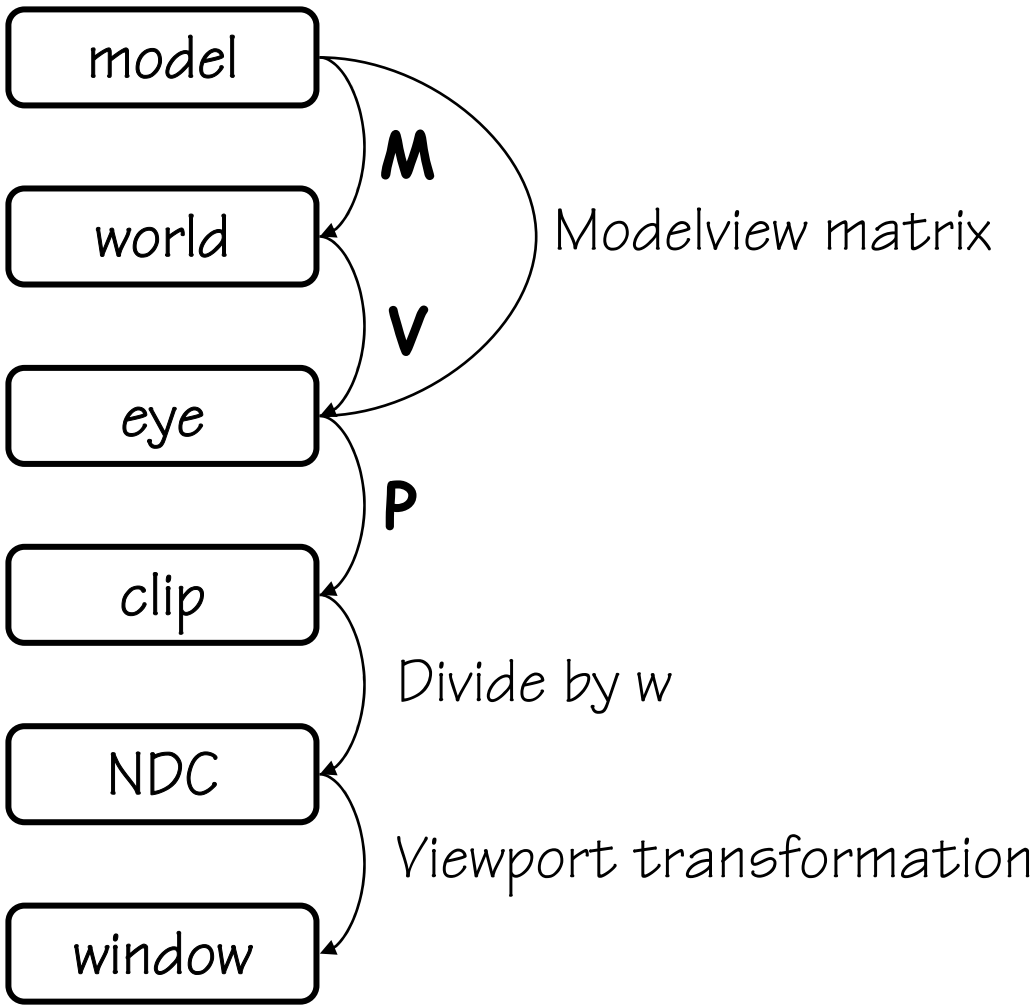
From last time…

- Lines and planes
- Culling
  - View frustum culling
  - Back-face culling
  - Occlusion culling
  - Hierarchical culling
- Clipping

Topics for today

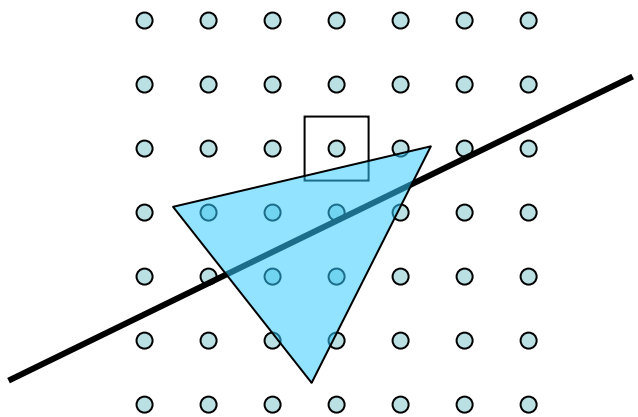
- Quick review of coordinate systems
- Motivation
  - What is rasterization?
  - Why triangles?
- Rasterization
  - Scan-line
  - Edge equations
- Interpolation
- Beyond triangles

Coordinate systems



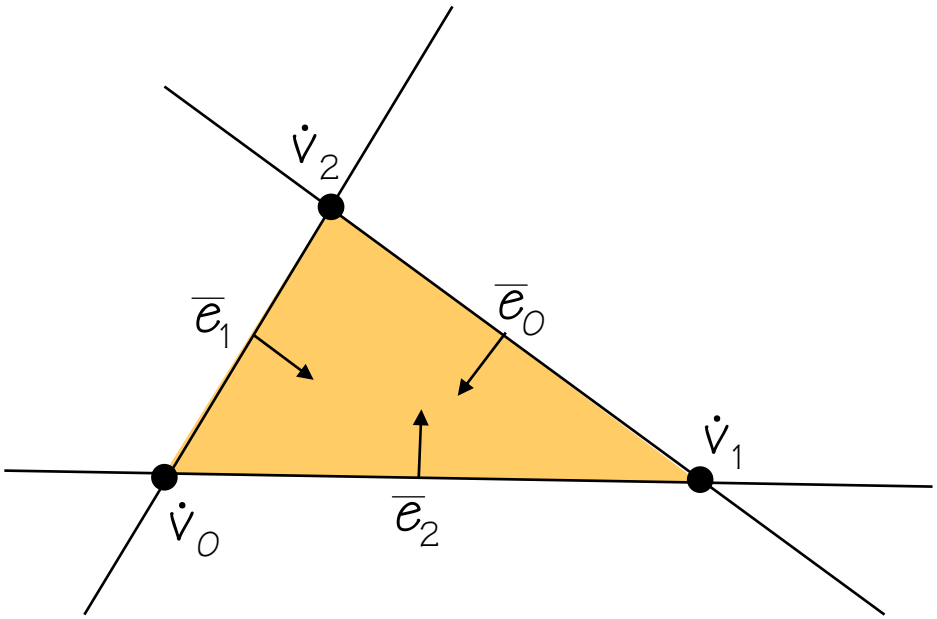
Primitive rasterization

- Rasterization converts vertex representation to pixel representation
  - **Coverage determination** – Computes which pixels (samples) belong to an "ideal" analytical primitive
  - **Parameter interpolation** – Computes parameters at covered pixels from parameters associated with primitive vertices
- Coverage is a 2-D sampling problem
- Possible coverage criteria:
  - distance of the primitive to sample point (often used with lines)
  - Percent coverage of a pixel (used to be popular)
  - Sample is inside the primitive (assuming it is closed)



Why triangles?

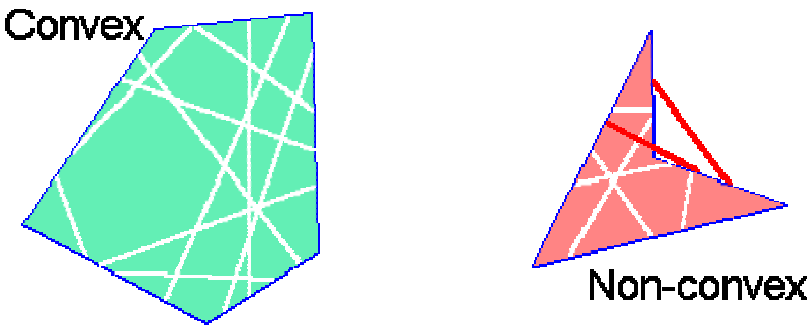
- Triangles are simple
  - minimal representation for a surface element (3 points or 3 edge equations)
  - triangles are linear (makes computations easier)



$$T = (\dot{v}_0, \dot{v}_1, \dot{v}_2)$$
$$T = (\bar{e}_0, \bar{e}_1, \bar{e}_2)$$

Why triangles?

- Triangles are convex
- What does it mean to be a convex?



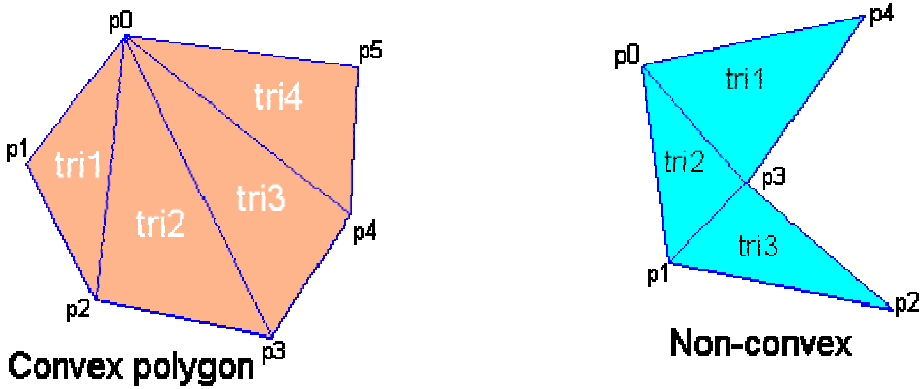
An object is convex if and only if any line segment connecting two points on its boundary is contained entirely within the object or one of its boundaries.

- Why is convexity important?

  Regardless of a triangle's orientation on the screen a given scan line will contain only a single segment or *span* of that triangle.

Why triangles?

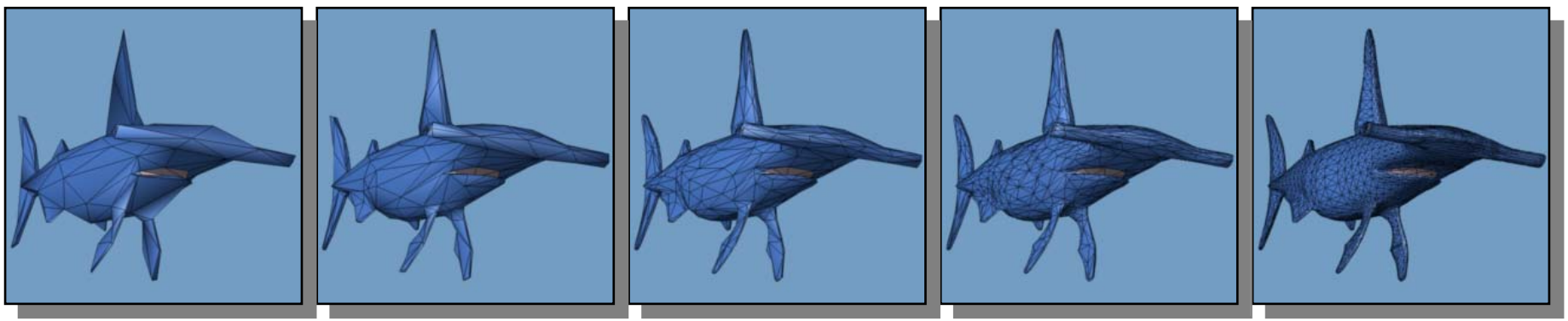
- Arbitrary polygons can be decomposed into triangles



- Decomposing a convex n-sided polygon is trivial
  - Suppose the polygon has ordered vertices $\{v_0, v_1, \ldots v_n\}$
  - It can be decomposed into triangles $\{(v_0,v_1,v_2), \{v_0,v_2,v_3), (v_0,v_i,v_{i+1}), \ldots (v_0,v_{n-1},v_n)\}$.
- Decomposing a non-convex polygon is non-trivial
  - sometimes have to introduce new vertices

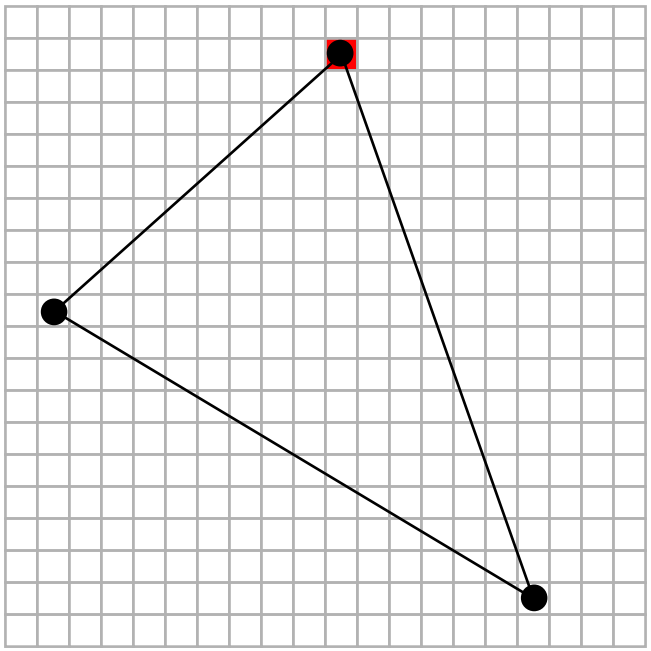
Why triangles?

- Triangles can approximate any 2-dimensional shape (or 3D surface)
- Polygons are a locally linear (planar) approximation.
- Improve the quality of fit by increasing the number edges or faces.



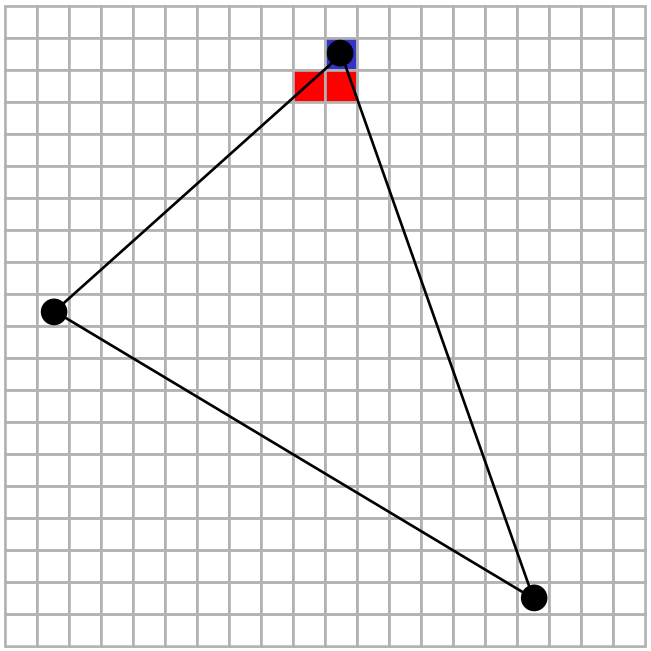
Scanline triangle rasterizer

- Walk along edges one scanline at a time
- Rasterize spans between edges



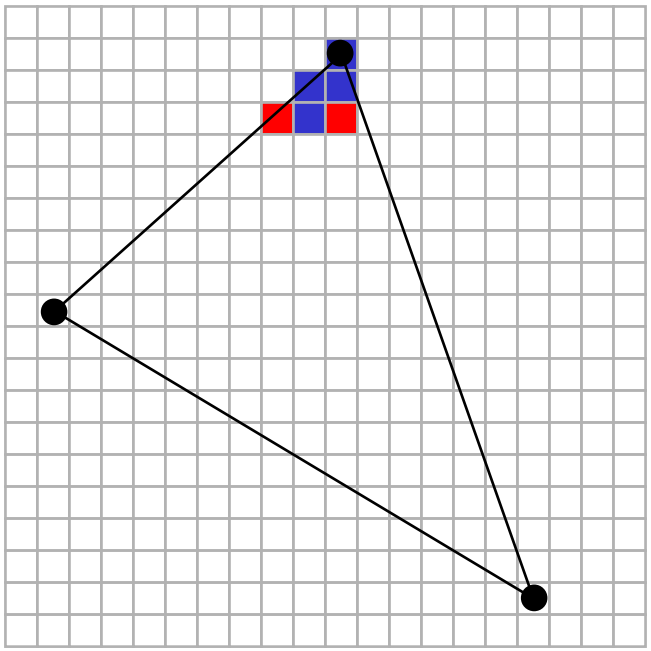
Scanline triangle rasterizer

- Walk along edges one scanline at a time
- Rasterize spans between edges



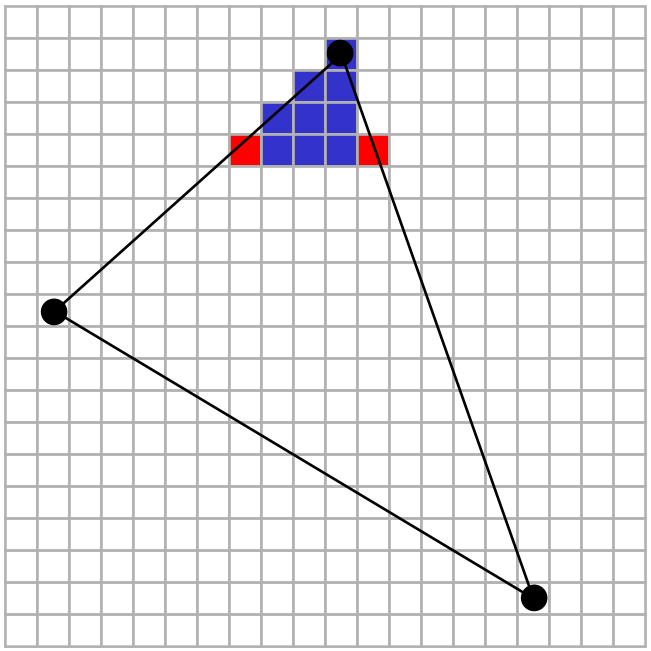
Scanline triangle rasterizer

- Walk along edges one scanline at a time
- Rasterize spans between edges



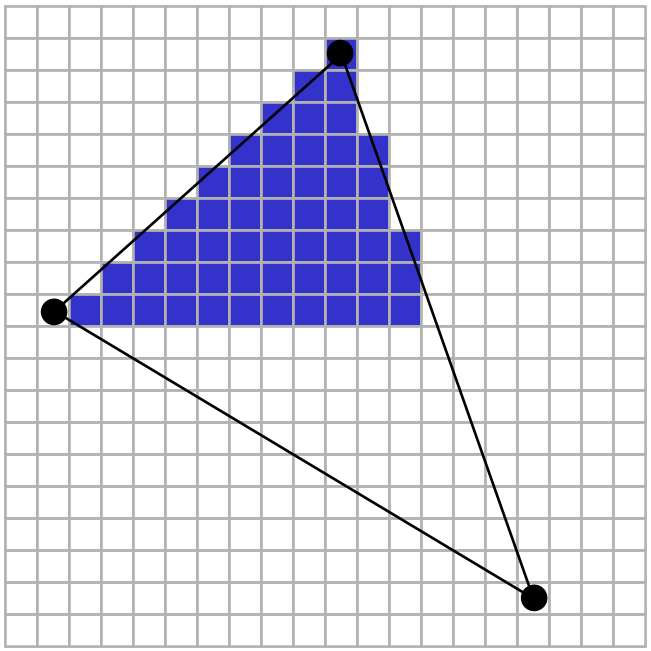
Scanline triangle rasterizer

- Walk along edges one scanline at a time
- Rasterize spans between edges



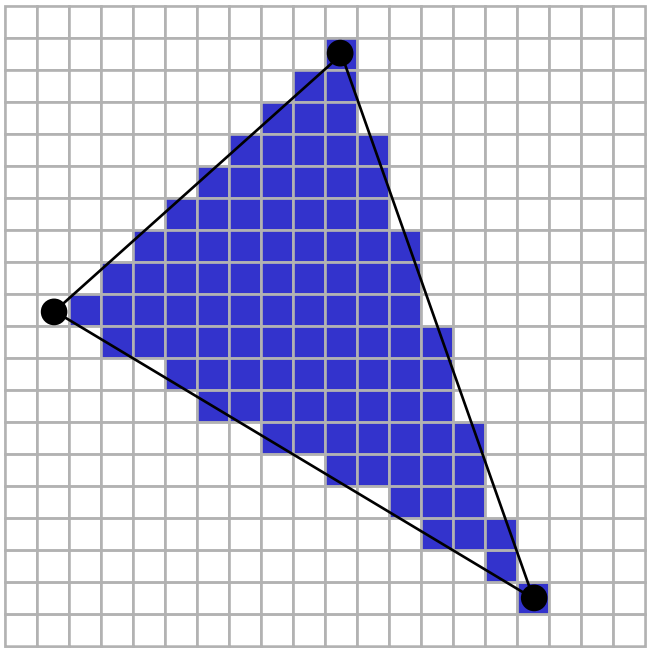
Scanline triangle rasterizer

- Walk along edges one scanline at a time
- Rasterize spans between edges

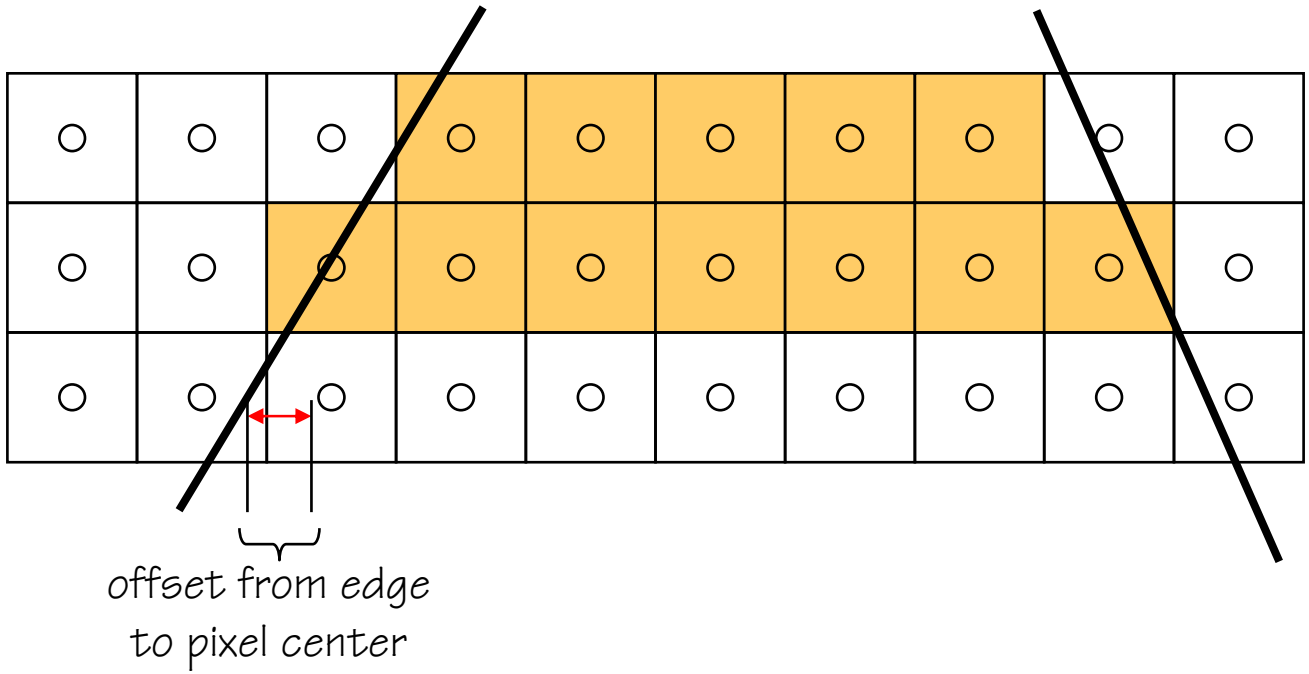


Scanline triangle rasterizer

- Walk along edges one scanline at a time
- Rasterize spans between edges



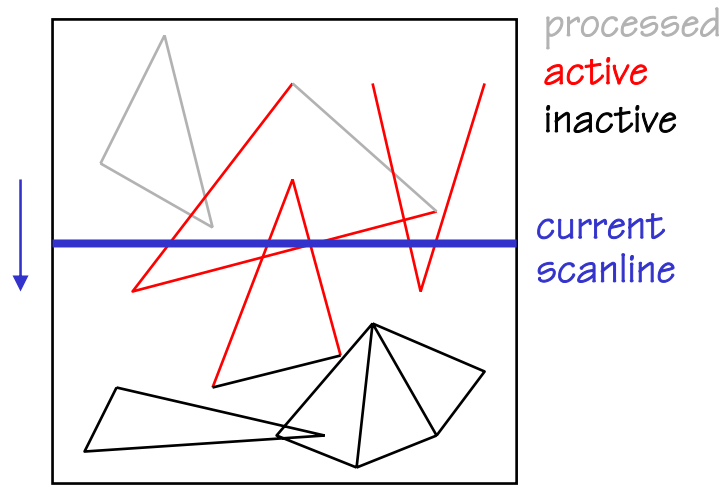
Fractional offsets

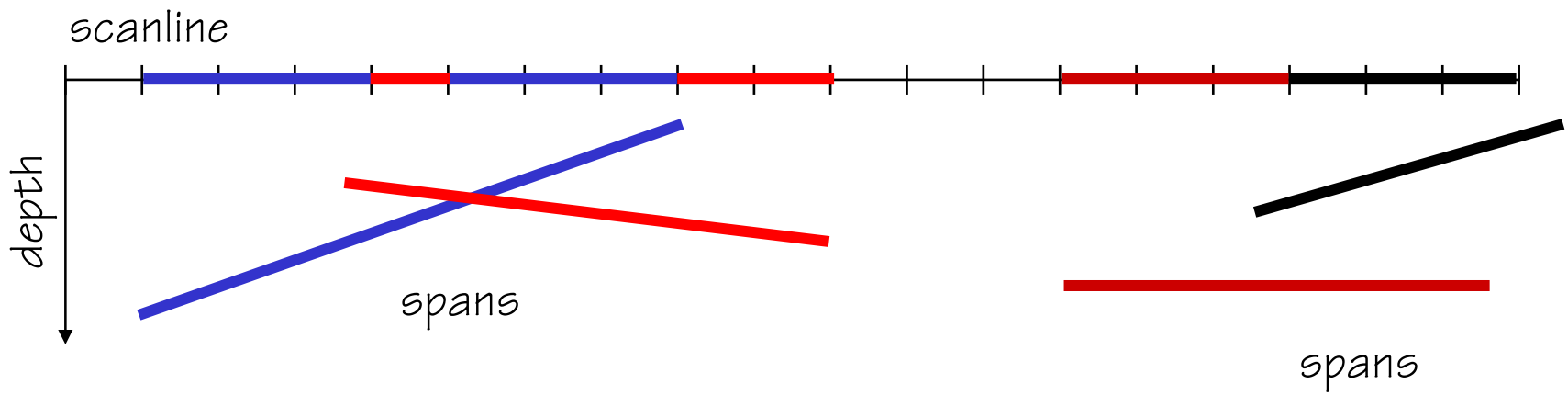


- Straightforward to interpolate values (e.g. colors) along the edges, but must be careful when offsetting from the edge to the pixel's center.

Scanline rasterizing entire scenes

- Sort all edges by start scanline into the Inactive Edge Table (IET)
- Move edges intersected by current scanline from IET to Active Edge Table (AET)
- Compute spans between active edges
- Sort spans by starting x
- Rasterize visible span segments
- Remove edges from AET when they no longer intersect the current scanline



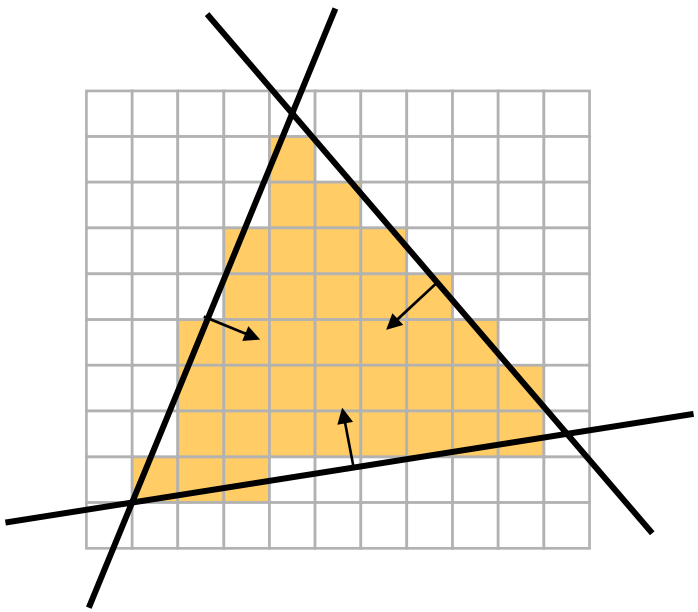


Scanline rasterization

- Advantages:
  - Can be made quite fast
  - Low memory usage for smallish scenes
  - Don't need full 2D z-buffer (can use 1D z-buffer on the scanline)
- Disadvantages:
  - Doesn't scale well to large scenes
  - Have to worry about fractional offsets
  - Lots of special cases

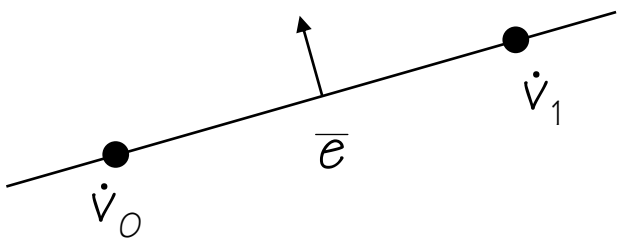
Rasterizing with edge equations

- Compute edge equations from vertices
- Compute interpolation equations from vertex parameters
- Traverse pixels evaluating the edge equations
- Draw pixels for which all edge equations are positive
- Interpolate parameters at pixels



Edge equation coefficients

- The cross product between 2 homogeneous points generates the line between them



$$\bar{e} = \dot{v}_0 \times \dot{v}_1$$
$$= [x_0 \quad y_0 \quad 1]^t \times [x_1 \quad y_1 \quad 1]^t$$
$$= [(y_0 - y_1) \quad (x_1 - x_0) \quad (x_0 y_1 - x_1 y_0)]$$
$$A \qquad B \qquad C$$

$$E(x,y) = Ax + By + C$$

- A pixel at (x,y) is "inside" an edge if $E(x,y) > 0$

Numerical precision

- Subtraction of two nearly equal floating point numbers results in catastophic cancellation which leaves only a few significant bits

$$1.234 \times 10^3 - 1.233 \times 10^3 = 1.000 \times 10^0$$

- When $x_0 y_1 \approx x_1 y_0$ computing $C = x_0 y_1 - x_1 y_0$ can result in loss of precision
- Reformulate C coefficent:

$$C = -\frac{A(x_0 + x_1) + B(y_0 + y_1)}{2}$$

Triangle area

$$\text{Area} = \tfrac{1}{2} \det \begin{bmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ 1 & 1 & 1 \end{bmatrix}$$

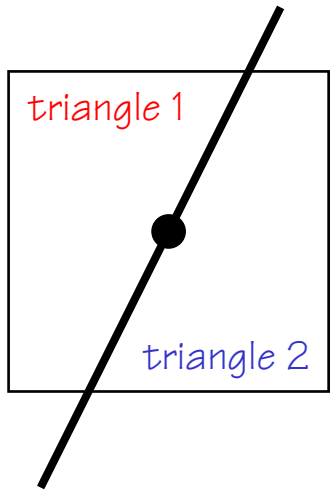
$$= \tfrac{1}{2}((x_1 y_2 - x_2 y_1) - (x_0 y_2 - x_2 y_0) + (x_0 y_1 - x_1 y_0))$$

$$= \tfrac{1}{2}(C_0 + C_1 + C_2)$$

- Area = 0 means that the triangle is not visible
- Area < 0 means the triangle is back facing:
  - Reject triangle if performing back-face culling
  - Otherwise, flip edge equations by multiplying by -1

Shared edges

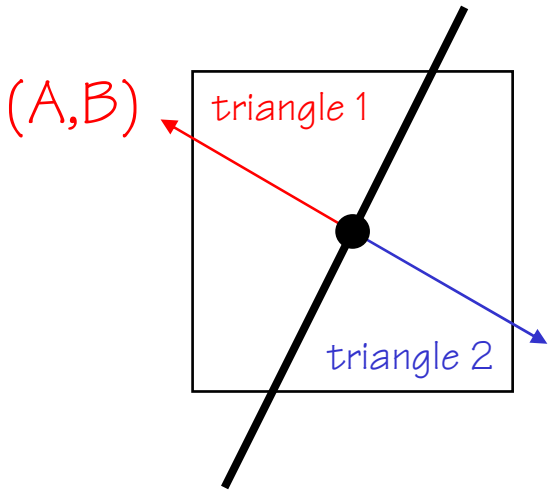
- Suppose two triangles share an edge. Which covers the pixel when the edge passes through the sample (E(x,y)=0)?
- Both
  - pixel color becomes dependent on order of triangle rendering
  - creates problems when rendering transparent objects - "double hitting"
- Neither
  - Missing pixels create holes in otherwise solid surface
- We need a consistent tie-breaker!



Shared edges

- A common tie-breaker:

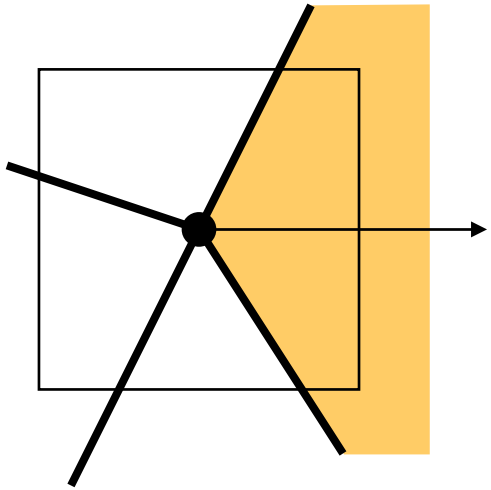
$$\text{bool } t = \begin{cases} A > 0 & \text{if } A \neq 0 \\ B > 0 & \text{otherwise} \end{cases}$$



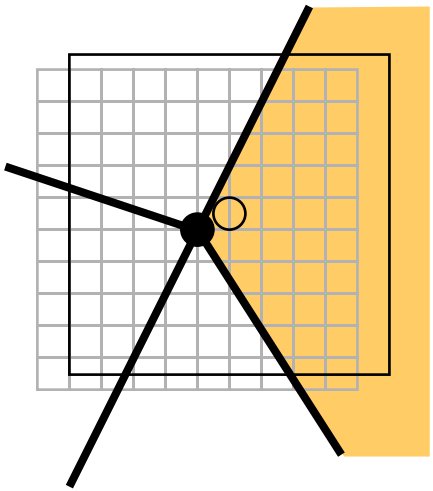
- Coverage determination becomes

```
if( E(x,y) >0 || (E(x,y)==0 && t))
    pixel is covered
```

Shared vertices



- Use “inclusion direction” as a tie breaker.
- Any direction can be used



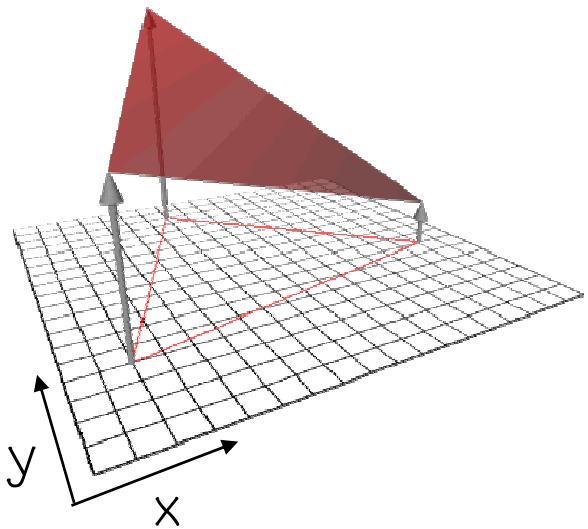
- Snap vertices to subpixel grid and displace so that no vertex can be at the pixel center

Other benefits of snapping to subpixel grid

- Simplicity
  - can use fixed-point arithmetic can be used (integer operations)
- Robustness
  - With sufficient bits, edge equations and areas can be computed exactly
- Quality
  - Smoother animation than if we snapped to the pixel grid

Interpolating parameters

- Specify a parameter, say redness (r) at each vertex of the triangle.
- Linear interpolation creates a planar function



$$r(x,y) = Ax + By + C$$

Solving for interpolation equation

- Given the redness of the three vertices, we can set up the following linear system:

$$\begin{bmatrix} r_0 & r_1 & r_2 \end{bmatrix} = \begin{bmatrix} A_r & B_r & C_r \end{bmatrix} \begin{bmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ 1 & 1 & 1 \end{bmatrix}$$

with the solution:

$$\begin{bmatrix} A_r & B_r & C_r \end{bmatrix} = \begin{bmatrix} r_0 & r_1 & r_2 \end{bmatrix} \frac{\begin{bmatrix} (y_1 - y_2) & (x_2 - x_1) & (x_1 y_2 - x_2 y_1) \\ (y_0 - y_2) & (x_2 - x_0) & (x_0 y_2 - x_2 y_0) \\ (y_0 - y_1) & (x_1 - x_0) & (x_0 y_1 - x_1 y_0) \end{bmatrix}}{\det \begin{bmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ 1 & 1 & 1 \end{bmatrix}}$$

Interpolation equation

- The parameter plane equation is just a linear combination of the edge equations

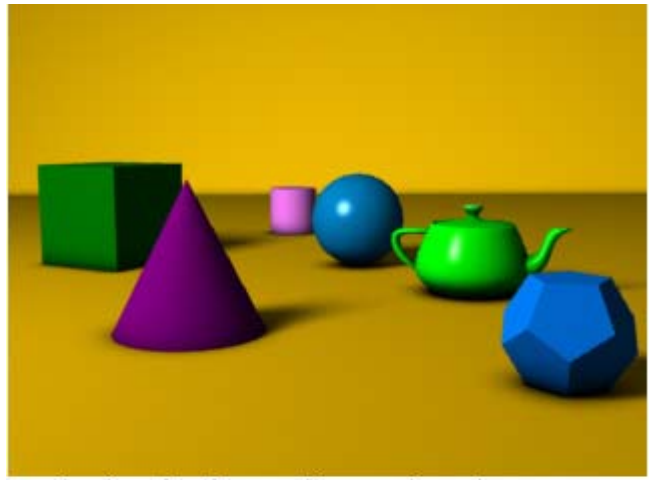
$$\begin{bmatrix} A_r & B_r & C_r \end{bmatrix} = \frac{1}{2 \cdot area} \begin{bmatrix} r_0 & r_1 & r_2 \end{bmatrix} \begin{bmatrix} \overline{e}_0 \\ \overline{e}_1 \\ \overline{e}_2 \end{bmatrix}$$

Extra work to interpolate a parameter:

- Transform parameter vector
- Compute one interpolation equation per pixel per parameter

Z-Buffering

- When rendering multiple triangles we need to determine which triangles are visible
- Use z-buffer to resolve visibility
  - stores the depth `at each pixel
- Initialize z-buffer to 1
  - Post-perspective z values lie between 0 and 1
- Linearly interpolate depth (z_{tri}) across triangles
  - Why can we do this?
- If $z_{tri}(x,y) <$ zBuffer[x][y]
  write to pixel at (x,y)
  zBuffer[x][y] = $z_{tri}(x,y)$



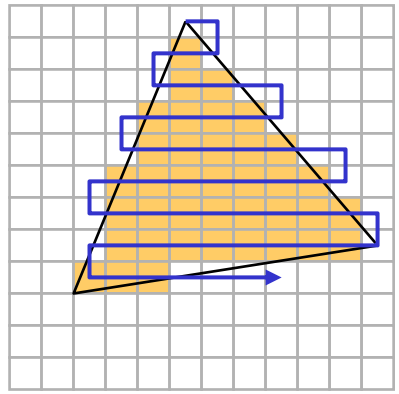
A simple three dimensional scene

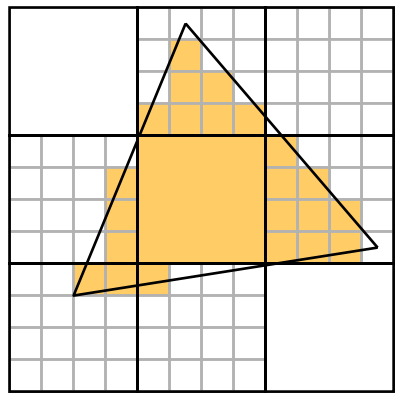
image from wikipedia.com

Z-buffer representation

Traversing pixels

- Free to traverse pixels how we please
  - Edge and interpolation equations can be computed at any point
- Try to minimize work
  - Restrict traversal to primitive bounding box
  - Zig-zag traversal avoids empty pixels
  - Hierarchical traversal
    - Knock out tiles of pixels (say 4x4) at a time
    - Test corners of tiles against equations
    - Test individual pixels of tiles not entirely inside or outside





Incremental update of linear equations

- Some computation can be saved by updating the edge and interpolation equations incrementally:

$$
\begin{aligned}
E(x,y) &= Ax + By + C \\
E(x+\Delta,y) &= A(x+\Delta) + By + C \\
&= E(x,y) + A\cdot\Delta \\
E(x,y+\Delta) &= Ax + B(y+\Delta) + C \\
&= E(x,y) + B\cdot\Delta
\end{aligned}
$$

- Equations can be updated with a single addition!

Triangle setup

- Compute edge equations
  - 3 cross products
- Compute triangle area
  - A few additions
- Cull zero area and back-facing triangles and/or flip edge equations
- Compute interpolation equations
  - Matrix/vector product per parameter

A Post-Triangle World?

- Are triangles really the best rendering primitive?

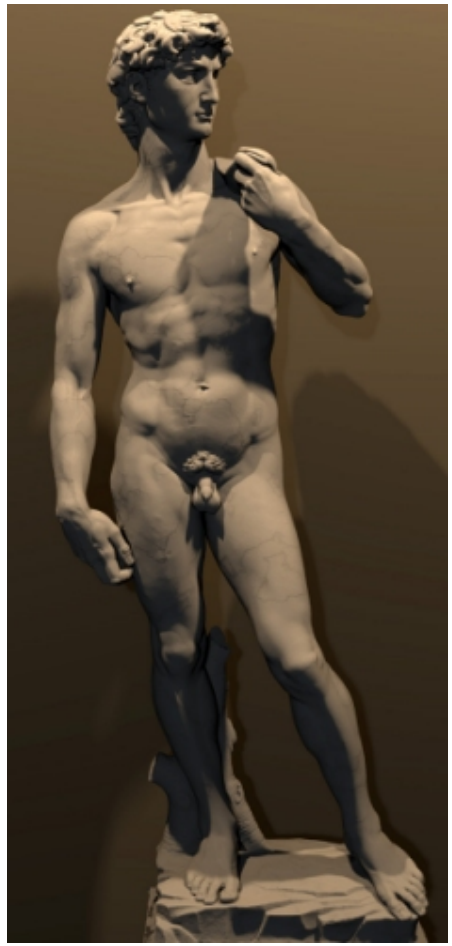
100,000,000 primitives

2,000,000 pixels

5 primitives/pixel

(Assuming that primitives are uniformly distributed over screen and only 10% are visible.)

- Cost to render a single triangle
  - specify 3 vertices
  - compute 3 edge equations
  - evaluate equations one



Models of this magnitude are being built today. The leading and most ambitious work in this area is Stanford's "Digital Michelangelo Project".

Point-cloud rendering

- Points have been proposed as a rendering primitive to address this problem.
- Key Attibutes:
  - Hierarchy
  - Incremental refinement
  - Compact representation (differental encoding)


130,712 Splats, 132 mS


259,975 Splats, 215 mS


1,017,149 Splats, 722 mS


14,835,967 Splats, 8308 mS

Next time

- Texture mapping
- Barycentric coodinates
- Perspective-correct interpolation
- Texture filtering