Rasterization

May 1, 2006

Triangles Only

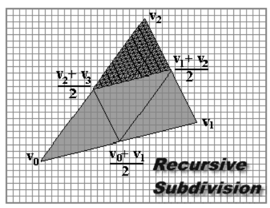
- We will discuss the rasterization of triangles only.
- Why?
  - Polygon can be decomposed into triangles.
  - A triangle is always convex.
  - Results in algorithms that are more hardware friendly.

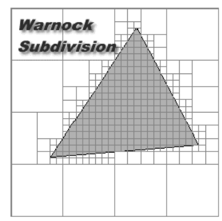
Scan-converting Triangles

The two most common strategies for scan-converting triangles are *edge walking* and *edge equations*

There are, however, other techniques including:

- Recursive subdivision of primitive (micro-polygons)
- Recursive subdivision of screen (Warnock's algorithm)





Being Hardware Friendly

- [Angel 4e] Section 7.11.3 or
- [Angel 3e] Section 8.11.6:
  - Intersect scan lines with polygon edges.
  - Sort the intersections, first by scan lines, then by order of x on each scan line.
  - It works for polygons in general, not just in triangles.
  - O(n log n) complexity ➔ feasible in software implementation only (i.e., not hardware friendly)

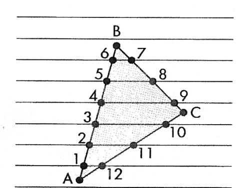


Figure 8.55 Polygon generated by vertex list.

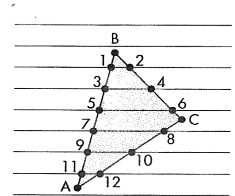
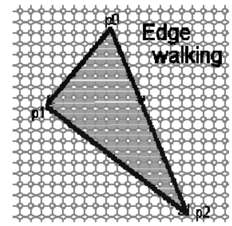


Figure 8.56 Desired order of vertices.

Edge-Walking Triangle Rasterizer

Notes on edge walking:

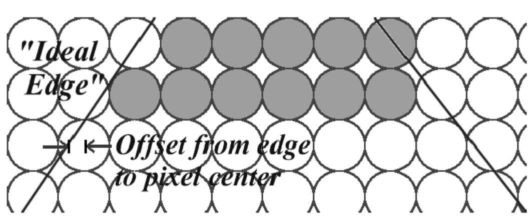
- Sort the vertices in both x and y
- Determine if the middle vertex, or *breakpoint* lies on the left or right side of the polygon. If the triangle has an edge parallel to the scan line direction then there is no breakpoint.
- Determines the left and right extents for each scan line (called *spans*).
- Walk down the left and right edges filling the pixels in-between until either a breakpoint or the bottom vertex is reached.



Advantages and Disadvantages:

- Generally very fast
- Loaded with special cases (left and right breakpoints, no breakpoints)
- Difficult to get right
- Requires computing fractional offsets when interpolating parameters across the triangle

Fractional Offsets



We can use *ceiling* to find the leftmost pixel in span and *floor* to find the rightmost pixel.

The trick comes when interpolating color values. It is straightforward to interpolate along the edges, but you must be careful when offsetting from the edge to the pixels center.

Color and Z

- Now we know which pixels must be drawn. The next step is to find their colors and Z's.
- Gouraud shading: linear interpolation of the vertex colors.
- Isn't it straightforward?
  - Interpolate along the edges. (Y direction)
  - Then interpolate along the span. (X direction)

Interpolation in World Space vs Screen Space

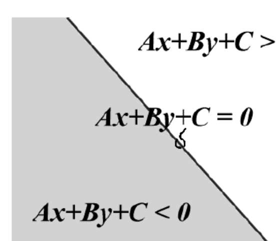
- p1=(x1, y1, z1, c1); p2=(x2, y2, z2, c2); p3=(x3, y3, z3, c3) in world space
- If (x3, y3) = (1-t)(x1, y1) + t(x2, y2) then z3=(1-t)z1+t z2; c3=(1-t)c1+t c2
- But, remember that we are interpolating on screen coordinates (x', y'):

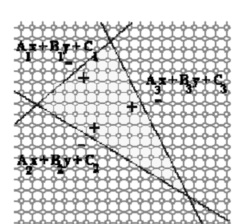
$$\begin{bmatrix} x'w \\ y'w \\ z'w \\ w \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & p & q \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Let $p'_1=(x'_1, y'_1)$; $p'_2=(x'_2, y'_2)$ and
  $p'_3=(x'_3, y'_3)= (1-s)(x'_1, y'_1) + s(x'_2, y'_2)$
- Does s=t? If not, should we compute z3 and c3 by s or t?
- Express s in t (or vice versa), we get something like: $$s = \frac{t \cdot w_2}{w_1 + t(w_2 - w_1)}$$
- So, if we interpolate z on screen space, we get the z of "some other point on the line"
- This is OK for Z's, but may be a problem for texture coordinates (topic of another lecture)

Rasterizing Triangles with Edge Equations

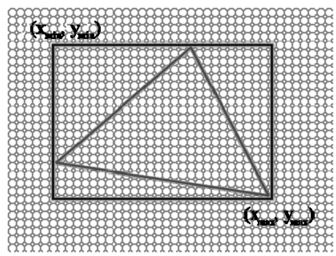
- An edge equation is simply a discriminating function like those used in curve and line-drawing algorithms.
- An edge equation segments a planar region into three parts, a boundary, and two half-spaces. The boundary is identified by points where the edge equation is equal to zero. The half-spaces are distinguished by differences in the edge equation's sign. We can choose which half-space is positive by multiplication by -1.
- We can scale all three edges so that their negative half-spaces are on the triangle's exterior.





Notes on using Edge Equations

- Compute edge equations from vertices
- Orient edge equations
- Compute a bounding box
- Scan through pixels in bounding box evaluating the edge equations
- When all three are positive then draw the pixel.



A Post-Triangle World?

Are triangles really the best rendering primitive?

100,000,000 primitive models displayed on 2,000,000 pixel displays.

Even even if we assume that only 10% of the primitives are visible, and they are uniformly distributed over the whole screen, that's still 5 primitives/pixel. Remember, that in order to draw a single triangle we must specify 3 vertices, determine three colors, and interpolate within 3 edges. On average, these triangle will impact only a fraction of a pixel.

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

Appendix

Derivation of *s* and *t*

- Two end points $P_1=(x_1, y_1, z_1)$ and $P_2=(x_2, y_2, z_2)$. Let $P_3=(1-t)P_1+(t)P_2$
- After projection, P_1, P_2, P_3 are projected to (x'_1, y'_1), (x'_2, y'_2), (x'_3, y'_3) in screen coordinates. Let $(x'_3, y'_3)=(1-s)(x'_1, y'_1) + s(x'_2, y'_2)$.

- (x'$_1$, y'$_1$), (x'$_2$, y'$_2$), (x'$_3$, y'$_3$) are obtained from P$_1$, P$_2$, P$_3$ by:

$$\begin{bmatrix} x'_1 w_1 \\ y'_1 w_1 \\ z'_1 w_1 \\ w_1 \end{bmatrix} = M \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix}, \qquad \begin{bmatrix} x'_2 w_2 \\ y'_2 w_2 \\ z'_2 w_2 \\ w_2 \end{bmatrix} = M \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x'_3 w_3 \\ y'_3 w_3 \\ z'_3 w_3 \\ w_3 \end{bmatrix} = M \begin{bmatrix} x_3 \\ y_3 \\ z_3 \\ 1 \end{bmatrix} = M((1-t)\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} + t \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix})$$

Since

$$M \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = \begin{bmatrix} x'_1 w_1 \\ y'_1 w_1 \\ z'_1 w_1 \\ w_1 \end{bmatrix}, \qquad M \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} = \begin{bmatrix} x'_2 w_2 \\ y'_2 w_2 \\ z'_2 w_2 \\ w_2 \end{bmatrix}$$

We have:

$$\begin{bmatrix} x'_3 w_3 \\ y'_3 w_3 \\ z'_3 w_3 \\ w_3 \end{bmatrix} = (1-t) M \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} + t \cdot M \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix}$$

$$= (1-t) \begin{bmatrix} x'_1 w_1 \\ y'_1 w_1 \\ z'_1 w_1 \\ w_1 \end{bmatrix} + t \begin{bmatrix} x'_2 w_2 \\ y'_2 w_2 \\ z'_2 w_2 \\ w_2 \end{bmatrix}$$

When P_3 is projected to the screen, we get (x'_3, y'_3) by dividing by w, so:

$$(x'_3, y'_3) = \left(\frac{(1-t)x'_1 w_1 + t \cdot x'_2 w_2}{(1-t)w_1 + t \cdot w_2}, \frac{(1-t)y'_1 w_1 + t \cdot y'_2 w_2}{(1-t)w_1 + t \cdot w_2}\right)$$

But remember that

$$(x'_3, y'_3) = (1-s)(x'_1, y'_1) + s(x'_2, y'_2)$$

Looking at x coordinate, we have

$$(1-s)x_1 + s \cdot x_2 = \frac{(1-t)x'_1 w_1 + t \cdot x'_2 w_2}{(1-t)w_1 + t \cdot w_2}$$

We may rewrite s in terms of t, w_1, w_2, x'_1, and x'_2.

In fact, $$s = \frac{t \cdot w_2}{(1-t)w_1 + t \cdot w_2} = \frac{t \cdot w_2}{w_1 + t(w_2 - w_1)}$$

or conversely

$$t = \frac{s \cdot w_1}{s \cdot w_1 + (1-s)w_2} = \frac{s \cdot w_1}{s(w_1 - w_2) + w_2}$$

Surprisingly, x'_1 and x'_2 disappear.