Scan Conversion

Wolfgang Heidrich

Course News

Assignment 2
- Due Monday, Feb 28

Homework 3
- Discussed in labs this week

Homework 4
- Hidden surface removal, out today

Reading
- Chapters 8, 9
- Hidden surface removal, shading

The Rendering Pipeline

Scan Conversion - Rasterization

Convert continuous rendering primitives into discrete fragments/pixels

- Lines
  - Midpoint/Bresenham
- Triangles
  - Flood fill
  - Scanline
  - Implicit formulation
  - Interpolation

Scan Conversion of Polygons

One possible scan conversion

A General Algorithm

- Intersect each scanline with all edges
- Sort intersections in x
- Calculate parity to determine in/out
- Fill the 'in' pixels
Edge Walking

for (y=yB; y<=yT; y++) {
    for (x=xL; x<=xR; x++)
        setPixel(x, y);
    xL += DxL;
    xR += DxR;
}

\[
\begin{align*}
    &x_L, x_R, \Delta x_L, \Delta x_R, y_L, y_R, \Delta y_L, \Delta y_R \in \mathbb{R} \ni x_L < x_R, \Delta x_L > 0, y_L < y_R, \Delta y_L > 0, X_L, Y_L, X_R, Y_R \in \mathbb{R} \\
    \end{align*}
\]

Modern Rasterization: Edge Equations

Define a triangle as follows:

Using Edge Equations

Usage:
- Go over each pixel in bounding rectangle
- Check if pixel is inside/outside of triangle
  - Using sign of edge equations

Computing Edge Equations

Implicit equation of a triangle edge:

\[
L(x, y) = \frac{(y - y_1)}{(x - x_1)}(x - x_2) - (y - y_1) = 0
\]

(see Bresenham algorithm)
- \(L(x, y)\) positive on one side of edge, negative on the other

Question:
- What happens for vertical lines?

Edge Equations

Multiply with denominator

\[
L(x, y) = (y - y_1)(x - x_2) - (y - y_2)(x - x_1) = 0
\]

- Avoids singularity
- Works with vertical lines

What about the sign?
- Which side is in, which is out?
**Edge Equations**

**Determining the sign**
- Which side is “in” and which is “out” depends on order of start/end vertices...
- Convention: specify vertices in counter-clockwise order

\[ L(x, y) = -(y_e - y_s)(x - x_s) + (y - y_s)(x_e - x_s) = 0 \]

**Counter-Clockwise Triangles**
- The equation \( L(x, y) \) as specified above is negative inside, positive outside
  - Flip sign:
  \[ L(x, y) = -(y_e - y_s)(x - x_s) + (y - y_s)(x_e - x_s) = 0 \]

**Clockwise triangles**
- Use original formula
  \[ L(x, y) = (y_e - y_s)(x - x_s) - (y - y_s)(x_e - x_s) = 0 \]

**Discussion of Polygon Scan Conversion Algorithms**

**On old hardware:**
- Use first scan-conversion algorithm
  - Scan-convert edges, then fill in scanlines
  - Compute interpolated values by interpolating along edges, then scanlines
  - Requires clipping of polygons against viewing volume
  - Faster if you have a few, large polygons
  - Possibly faster in software

**Modern GPUs:**
- Use edge equations
- And plane equations for attribute interpolation
- No clipping of primitives required
- Faster with many small triangles

**Additional advantage:**
- Can control the order in which pixels are processed
- Allows for more memory-coherent traversal orders
  - E.g. tiles or space-filling curve rather than scanlines

**Triangle Rasterization Issues (Independent of Algorithm)**

**Exactly which pixels should be lit?**
- A. Those pixels inside the triangle edge (of course)

**But what about pixels exactly on the edge?**
- Draw them: order of triangles matters (it shouldn’t)
- Don’t draw them: gaps possible between triangles

**We need a consistent (if arbitrary) rule**
- Example: draw pixels on left or top edge, but not on right or bottom edge

**Shared Edge Ordering**
Triangle Rasterization Issues

**Silver**

Moving Silvers

Triangle Rasterization Issues

**These are ALIASING Problems**
- Problems associated with representing continuous functions (triangles) with finite resolution (pixels)
- More on this problem when we talk about sampling...

Shading

Wolfgang Heidrich

The Rendering Pipeline

Input to Scan Conversion:
- Vertices of triangles (lines, quadrilaterals…)
- Color (per vertex)
- Specified with glColor
- Or: computed with lighting
- World-space normal (per vertex)
- Left over from lighting stage

Shading Task:
- Determine color of every pixel in the triangle
Shading

**How can we assign pixel colors using this information?**

- Easiest: flat shading
  - Whole triangle gets one color (color of 1st vertex)
- Better: Gouraud shading
  - Linearly interpolate color across triangle
- Even better:
  - Linearly interpolate the normal vector
  - Compute lighting for every pixel
  - Note: not supported by rendering pipeline as discussed so far

Flat Shading

- Simolest approach calculates illumination at a single point for each polygon

  Obviously inaccurate for smooth surfaces

Flat Shading Approximations

- *If an object really is faceted, is this accurate?*

  No!

  - For point sources, the direction to light varies across the facet
  - For specular reflectance, direction to eye varies across the facet

Improving Flat Shading

- *What if we evaluate Phong lighting model at each pixel of the polygon?*
  - Better, but result still clearly faceted

  For smoother-looking surfaces, we introduce vertex normals at each vertex

  - Usually different from facet normal
  - Used only for shading
  - Think of as a better approximation of the real surface that the polygons approximate

Vertex Normals

- *Vertex normals may be*
  - Provided with the model
  - Computed from first principles
  - Approximated by averaging the normals of the facets that share the vertex
**Gouraud Shading Artifacts**

- Often appears dull, chalky
- Lacks accurate specular component
  - If included, will be averaged over entire polygon

**Phong Shading**

- Linearly interpolating surface normal across the facet, applying Phong lighting model at every pixel
  - Same input as Gouraud shading
  - Pros: much smoother results
  - Cons: considerably more expensive

**Phong Shading Difficulties**

- Computationally expensive
  - Per-pixel vector normalization and lighting computation!
  - Floating point operations required
- Lighting after perspective projection
  - Messes up the angles between vectors
  - Have to keep eye-space vectors around
- No direct support in standard rendering pipeline
  - But can be simulated with texture mapping, procedural shading hardware (see later)

**Shading Artifacts: Silhouettes**

- Polygonal silhouettes remain

**Mach bands**

- Eye enhances discontinuity in first derivative
- Very disturbing, especially for highlights
How to Interpolate?

Need to propagate vertex attributes to pixels
- Interpolate between vertices:
  - z (depth)
  - r,g,b color components
  - \( N_x, N_y, N_z \) surface normals
  - \( u,v \) texture coordinates (talk about these later)
- Three equivalent ways of viewing this (for triangles)
1. Linear interpolation
2. Barycentric coordinates
3. Plane Equation

1. Linear Interpolation

Interpolate quantity along \( L \) and \( R \) edges
-as a function of \( y \)
- Then interpolate quantity as a function of \( x \)

\[
P(x, y) = \alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3
\]

\[
\alpha + \beta + \gamma = 1
\]

0 \leq \alpha, \beta, \gamma \leq 1

\[
\alpha = \frac{A_1}{A}
\]

\[
\beta = \frac{A_2}{A}
\]

\[
\gamma = \frac{A_3}{A}
\]

2. Barycentric Coordinates

Have seen this before
- Barycentric Coordinates: weighted combination of vertices, with weights summing to 1
- \( P = \alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3 \)
- \( \alpha + \beta + \gamma = 1 \)
- \( 0 \leq \alpha, \beta, \gamma \leq 1 \)

\[
P(x, y) = \alpha \cdot x_1 + \beta \cdot x_2 + \gamma \cdot x_3
\]

One way to compute them:

\[
x = \alpha x_1 + \beta x_2 + \gamma x_3 \text{ with}
\]

\[
\alpha = \frac{A_1}{A}
\]

\[
\beta = \frac{A_2}{A}
\]

\[
\gamma = \frac{A_3}{A}
\]
Barycentric Coordinates

**How to compute areas?**

- Cross products!

  e.g.: 
  
  $$A_i = \frac{1}{2} \|x_i - x_j\times(x - x_j)\|$$

3. Plane Equation

**Observation:** Quantities vary linearly across image plane

- e.g. \( r = Ax + By + C \)
- \( r \) = red channel of the color
- Same for \( g, b, Nx, Ny, Nz, z \)...

  From info at vertices we know:
  
  \[
  \begin{align*}
  r_1 &= Ax_1 + By_1 + C \\
  r_2 &= Ax_2 + By_2 + C \\
  r_3 &= Ax_3 + By_3 + C 
  \end{align*}
  \]

- Solve for \( Ax, By, C \)
- One-time set-up cost per triangle and interpolated quantity

Coming Up:

**Wednesday/Friday**

- Clipping, hidden surface removal