Course News

Assignment 2
- Due March 2

Homework 3
- Discussed in labs next week

Reading (this week)
- Chapter 3

Reading (next week)
- Chapter 8

Scan Conversion
Wolfgang Heidrich

The Rendering Pipeline

Geometry Database
- Model/View Transformation
- Lighting
- Perspective Transformation
- Clipping

Geometry Processing
- Scan Conversion
- Texturing
- Depth Test
- Blending
- Framebuffer

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

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**

```c
for (y=myT; y<myT; y++) {
    for (x=myL; x<myR; x++)
        setPixel(x, y);
    myL += dxL;
    myR += dxR;
}
```

**Edge Walking Triangles**

- Split triangles into two regions with continuous left and right edges
- ScanTrapezoid($x_p, y_p, y_m, 1, \frac{1}{m_1}, 1, 1$)
- ScanTrapezoid($x_p, y_p, y_m, 1, \frac{1}{m_2}, 1, 1$)

**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

**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
Discussion of Polygon Scan Conversion Algorithms

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

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

Triangle Rasterization Issues

Shared Edge Ordering

Triangle Rasterization Issues

Sliver

Triangle Rasterization Issues

Moving Slivers

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

Geometry Database
ModelView Transform.
Lighting
Perspective Transform.
Clipping

Scan Conversion
Texturing
Depth Test
Blending
Frame-buffer

Shading

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

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

- Simplest 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 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.

- Not the same as Phong lighting:
  - Common confusion.
  - Phong lighting: empirical model to calculate illumination at a point on a surface.
**Phong Shading**

*Linearly interpolate the vertex normals*
- Compute lighting equations at each pixel
- Can use specular component

\[
I_{\text{total}} = k_a I_{\text{ambient}} + \sum_{i=1}^{n} I_i \left( k_d (\mathbf{n} \cdot \mathbf{l}_i) + k_s (\mathbf{v} \cdot \mathbf{r}_i)^n \right)
\]

remember: normals used in diffuse and specular terms

discontinuity in normal's rate of change harder to detect

**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*

Gouraud vs. Phong

**How to Interpolate?**

*Need to propagate vertex attributes to pixels*
- Interpolate between vertices:
  - \( z \) (depth)
  - \( r, g, b \) color components
  - \( N_1, N_2, N_3 \) 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 \( x \)

**Linear Interpolation**

*Most common approach, and what OpenGL does*
- Perform Phong lighting at the vertices
- Linearly interpolate the resulting colors over faces
  - Along edges
  - Along scanlines

*Same as Barycentric Coordinates!*

edge: mix of \( c_1, c_2 \)

interior: mix of \( c_1, c_2, c_3 \)
2. Barycentric Coordinates

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

Barycentric Coordinates
- Convex combination of 3 points
  \[ x = \alpha \cdot x_1 + \beta \cdot x_2 + \gamma \cdot x_3 \]
  with \( \alpha + \beta + \gamma = 1 \), \( 0 \leq \alpha, \beta, \gamma \leq 1 \)
- \( \alpha, \beta, \gamma \) are called *barycentric coordinates*

One way to compute them:
\[ x = \alpha x_1 + \beta x_2 + \gamma x_3 \] with
\[ \alpha = \frac{A_1}{A} \]
\[ \beta = \frac{A_2}{A} \]
\[ \gamma = \frac{A_3}{A} \]

3. Plane Equation

*Observation: Quantities vary linearly across image plane*
- E.g.: \( r = Ax + By + C \)
  - \( r \): red channel of the color
  - Same for \( g, b, N_x, N_y, N_z, z \)...
- From info at vertices we know:
  \[ r_1 = Ax_1 + By_1 + C \]
  \[ r_2 = Ax_2 + By_2 + C \]
  \[ r_3 = Ax_3 + By_3 + C \]
  - Solve for \( A, B, C \)
  - One-time set-up cost per triangle and interpolated quantity

Coming Up:

*Next week*
- Clipping, hidden surface removal