Overview

• Scan conversion
  ◦ Figure out which pixels to fill

• Shading
  ◦ Determine a color for each filled pixel

Scan Conversion

• Render an image of a geometric primitive by setting pixel colors

```c
void SetPixel(int x, int y, Color rgba)
```

• Example: Filling the inside of a triangle
Scan Conversion

• Render an image of a geometric primitive by setting pixel colors

  ```c
  void SetPixel(int x, int y, Color rgba)
  ```

• Example: Filling the inside of a triangle

![Scan Conversion Diagram](image)

Triangle Scan Conversion

• Properties of a good algorithm
  - Symmetric
  - Straight edges
  - Anti-aliased edges
  - No cracks between adjacent primitives
  - MUST BE FAST!

![Triangle Scan Conversion Diagram](image)

Simple Algorithm

• Color all pixels inside triangle

  ```c
  void ScanTriangle(Triangle T, Color rgba){
    for each pixel P at (x,y){
      if (Inside(T, P)){
        SetPixel(x, y, rgba);
      }
    }
  }
  ```

![Simple Algorithm Diagram](image)
Line defines two halfspaces
• Implicit equation for a line
  - On line: \( ax + by + c = 0 \)
  - On right: \( ax + by + c < 0 \)
  - On left: \( ax + by + c > 0 \)

Inside Triangle Test
• A point is inside a triangle if it is in the positive halfspace of all three boundary lines
  - Triangle vertices are ordered counter-clockwise
  - Point must be on the left side of every boundary line

Boolean Inside(Triangle T, Point P)
{  
    for each boundary line L of T {
        Scalar d = L.a*P.x + L.b*P.y + L.c;
        if (d < 0.0) return FALSE;
    }
    return TRUE;
}

Simple Algorithm
• What is bad about this algorithm?

void ScanTriangle(Triangle T, Color rgba){
    for each pixel P at (x,y){
        if (Inside(T, P))
            SetPixel(x, y, rgba);
    }
}
Triangle Sweep-Line Algorithm

- Take advantage of spatial coherence
  - Compute which pixels are inside using horizontal spans
  - Process horizontal spans in scan-line order
- Take advantage of edge linearity
  - Use edge slopes to update coordinates incrementally

```c
void ScanTriangle(Triangle T, Color rgba){
    for each edge pair {
        initialize xL, xR;
        compute dxL/dyL and dxR/dyR;
        for each scanline at y
            for (int x = xL; x <= xR; x++)
                SetPixel(x, y, rgba);
        xL += dxL/dyL;
        xR += dxR/dyR;
    }
}
```

Bresenham’s algorithm works the same way, but uses only integer operations!

Polygon Scan Conversion

- Fill pixels inside a polygon
  - Triangle
  - Quadrilateral
  - Convex
  - Star-shaped
  - Concave
  - Self-intersecting
  - Holes

What problems do we encounter with arbitrary polygons?
**Inside Polygon Rule**

- What is a good rule for which pixels are inside?

  - Concave
  - Self-Intersecting
  - With Holes

**Inside Polygon Rule**

- Odd-parity rule
  - Any ray from P to infinity crosses odd number of edges

---

**Polygon Sweep-Line Algorithm**

- Incremental algorithm to find spans, and determine insideness with odd parity rule
  - Takes advantage of scanline coherence

  ```c
  void ScanPolygon(Triangle T, Color rgba){
      sort edges by maxy
      make empty "active edge list"
      for each scanline (top-to-bottom) {
          insert/remove edges from "active edge list"
          update x coordinate of every active edge
          sort active edges by x coordinate
          for each pair of active edges (left-to-right)
          SetPixels(x_i, x_{i+1}, y, rgba);
      }
  }
  ```

---

**Polygon Sweep-Line Algorithm**

- Incremental algorithm to find spans, and determine insideness with odd parity rule
  - Takes advantage of scanline coherence

  ```c
  void ScanPolygon(Triangle T, Color rgba){
      sort edges by maxy
      make empty "active edge list"
      for each scanline (top-to-bottom) {
          insert/remove edges from "active edge list"
          update x coordinate of every active edge
          sort active edges by x coordinate
          for each pair of active edges (left-to-right)
          SetPixels(x_i, x_{i+1}, y, rgba);
      }
  }
  ```
Hardware Scan Conversion
- Convert everything into triangles
  - Scan convert the triangles

Hardware Anti-aliasing
- Supersample pixels
  - Multiple samples per pixel
  - Average subpixel intensities (box filter)
  - Trades intensity resolution for spatial resolution

Overview
- Scan conversion
  - Figure out which pixels to fill
- Shading
  - Determine a color for each filled pixel

Shading
- How do we choose a color for each filled pixel?
  - Each illumination calculation for a ray from the eyepoint through the view plane provides a radiance sample
    - How do we choose where to place samples?
    - How do we filter samples to reconstruct image?

Emphasis on methods that can be implemented in hardware
Ray Casting

- Simplest shading approach is to perform independent lighting calculation for every pixel
  - When is this unnecessary?

\[ I = I_e + K_a I_{at} + \sum (K_d (N \cdot L_i) I_i + K_t (V \cdot R_i)^t I_i) \]

Polygon Shading

- Can take advantage of spatial coherence
  - Illumination calculations for pixels covered by same primitive are related to each other

\[ I = I_e + K_a I_{at} + \sum (K_d (N \cdot L_i) I_i + K_t (V \cdot R_i)^t I_i) \]

Polygon Shading Algorithms

- Flat Shading
- Gouraud Shading
- Phong Shading
Flat Shading

- What if a faceted object is illuminated only by directional light sources and is either diffuse or viewed from infinitely far away

\[ I = I_0 + K_d I_d + \sum (K_p (N \cdot L) I_l + K_a (V \cdot R)^a I_l) \]

Flat Shading

- One illumination calculation per polygon
  - Assign all pixels inside each polygon the same color

Flat Shading

- Objects look like they are composed of polygons
  - OK for polyhedral objects
  - Not so good for smooth surfaces

Polygon Shading Algorithms

- Flat Shading
- Gouraud Shading
- Phong Shading
Gouraud Shading

- What if smooth surface is represented by polygonal mesh with a normal at each vertex?

\[ I = I_g + K_D I_L + \sum (K_R (N \cdot L_i) I_I + K_V (V \cdot R_i) I_I) \]

Gouraud Shading

- Method 1: One lighting calculation per vertex
  - Assign pixels inside polygon by interpolating colors computed at vertices

Gouraud Shading

- Bilinearly interpolate colors at vertices down and across scan lines

Gouraud Shading

- Smooth shading over adjacent polygons
  - Curved surfaces
  - Illumination highlights
  - Soft shadows

Mesh with shared normals at vertices
**Gouraud Shading**

- Produces smoothly shaded polygonal mesh
  - Piecewise linear approximation
  - Need fine mesh to capture subtle lighting effects

**Polygon Shading Algorithms**

- Flat Shading
- Gouraud Shading
- Phong Shading

**Phong Shading**

- What if polygonal mesh is too coarse to capture illumination effects in polygon interiors?

\[
I = I_e + K_d I_d + \sum (K_p (N \cdot L_i) I_i + K_l (V \cdot R_i)^n I_i)
\]

**Phong Shading**

- Method 2: One lighting calculation per pixel
  - Approximate surface normals for points inside polygons by bilinear interpolation of normals from vertices
Phong Shading

- Bilinearly interpolate surface normals at vertices down and across scan lines

\[ A = \alpha N_1 + (1 - \alpha) N_3 \]
\[ B = \beta N_2 + (1 - \beta) N_3 \]
\[ I = \phi A + (1 - \phi) B \]

Polygon Shading Algorithms

- Wireframe
- Flat
- Gouraud
- Phong

Shading Issues

- Problems with interpolated shading:
  - Polygonal silhouettes
  - Perspective distortion
  - Orientation dependence (due to bilinear interpolation)
  - Problems computing shared vertex normals
  - Problems at T-vertices

Summary

- 2D polygon scan conversion
  - Paint pixels inside primitive
  - Sweep-line algorithm for polygons

- Polygon Shading Algorithms
  - Flat
  - Gouraud
  - Phong
  - Ray casting

- Key ideas:
  - Sampling and reconstruction
  - Spatial coherence

Less expensive
More accurate