Scan Conversion & Shading

Thomas Funkhouser
Princeton University
COS 426, Fall 1999

3D Rendering Pipeline (for direct illumination)
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
  
  ```
  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
  
  ```cpp
  void SetPixel(int x, int y, Color rgba)
  ```

- Example: Filling the inside of a triangle

![Triangle](image)

Triangle Scan Conversion

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

![Polygons](image)
Triangle Scan Conversion

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

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);
    }
}
```
### 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

```cpp
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?

```c
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
Triangle Sweep-Line Algorithm

```c
void ScanTriangle(Triangle T, Color rgba) {
    for each edge pair {
        initialize x_L, x_R;
        compute dx_L/dy_L and dx_R/dy_R;
        for each scanline at y
            for (int x = x_L; x <= x_R; x++)
                SetPixel(x, y, rgba);
        x_L += dx_L/dy_L;
        x_R += dx_R/dy_R;
    }
}
```

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?
**Polygon Scan Conversion**

- Need better test for points inside polygon
  - Triangle method works only for convex polygons

**Inside Polygon Rule**

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

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

  ![Concave](image1)  ![Self-Intersecting](image2)  ![With Holes](image3)

Polygon Sweep-Line Algorithm

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

  ![Triangle](image4)  ![Polygon](image5)
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

Hardware Scan Conversion

- Convert everything into triangles
  - Scan convert the triangles
Hardware Antialiasing

- 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?

\[ I = I_E + K_A I_{AL} + \sum_i (K_D (N \cdot L_i) I_i + K_S (V \cdot R_i)^n I_i) \]
Polygon Shading

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

\[ I = I_E + K_A I_{AL} + \sum_i (K_D (N \cdot L_i) I_i) + K_S (V \cdot R_i)^n I_i \]

Polygon Shading Algorithms

- Flat Shading
- Gouraud Shading
- Phong Shading
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

\[ I = I_E + K_A I_{AL} + \sum_i (K_D (N \cdot L_i) I_i + K_S (V \cdot R_i)^n I_i) \]
Flat Shading

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

Objects look like they are composed of polygons
- OK for polyhedral objects
- Not so good for ones with 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?

\[ I = I_E + K_A I_{AL} + \sum_i (K_D (N \cdot L_i) I_i + K_S (V \cdot R_i)^n I_i) \]
Gouraud Shading

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

```
\[ A = \alpha l_1 + (1-\alpha)l_3 \]
```

```
\[ B = \beta l_2 + (1-\beta)l_3 \]
```

```
\[ I = \varphi A + (1-\varphi)B \]
```
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

Flat Shading  Gouraud Shading
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?

\[ I = I_E + K_A I_{AL} + \sum_i (K_D (N \cdot L_i)I_i + K_S (V \cdot R_i)^n I_i) \]
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

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

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

<table>
<thead>
<tr>
<th>Less expensive</th>
<th>More accurate</th>
</tr>
</thead>
</table>

- **Spatial coherence**

- **Sweep-line algorithm for polygons**