Scan Conversion & Shading
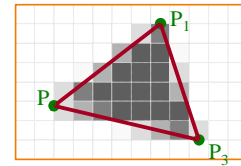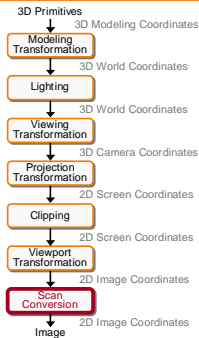
Thomas Funkhouser
Princeton University
COS 426, Spring 2004

3D Rendering Pipeline (for direct illumination)

3D Primitives
↓ 3D Modeling Coordinates
Modeling Transformation
↓ 3D World Coordinates
Lighting
↓ 3D World Coordinates
Viewing Transformation
↓ 3D Camera Coordinates
Projection Transformation
↓ 2D Screen Coordinates
Clipping
↓ 2D Screen Coordinates
Viewport Transformation
↓ 2D Image Coordinates
Scan Conversion
↓ 2D Image Coordinates
Image

Scan Conversion & Shading


Overview

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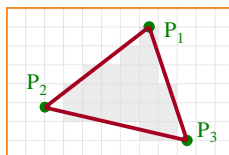 ○ Determine a color for each filled pixel

---

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
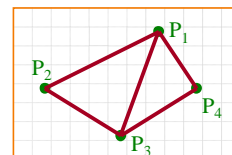
  ```
  void SetPixel(int x, int y, Color rgba)
  ```
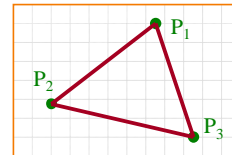
- Example: Filling the inside of a triangle

Triangle Scan Conversion

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

1

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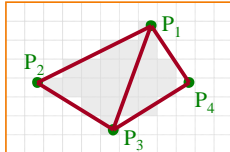
- Properties of a good algorithm
  - Symmetric
  - Straight edges
  - Antialiased edges
  - No cracks between adjacent primitives
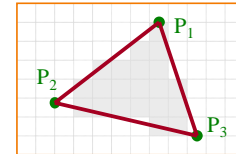 - MUST BE FAST!

Simple Algorithm

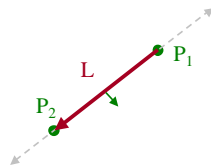- Color all pixels inside triangle

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


Line defines two halfspaces

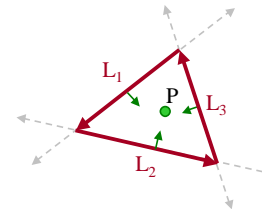- Implicit equation for a line
  - On line: $ax + by + c = 0$
  - On right: $ax + by + c < 0$
  - On left: $ax + by + c > 0$
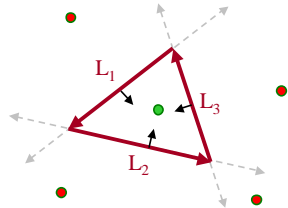

Inside Triangle Test

- 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

Inside Triangle Test

```
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;
}
```
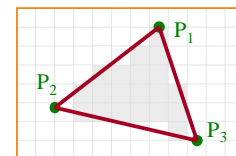

Simple Algorithm

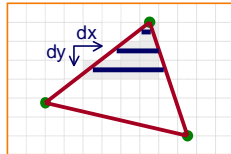- What is bad about this algorithm?

```
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
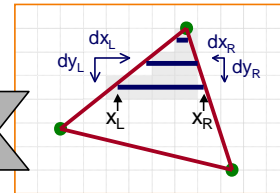 - Use edge slopes to update coordinates incrementally

Triangle Sweep-Line Algorithm

```
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;
  }
}
```
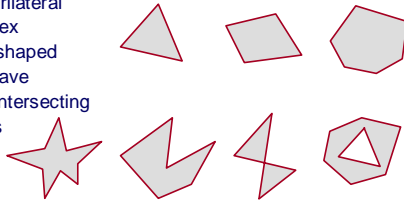
Bresenham's algorithm works the same way, but uses only integer operations!

Polygon Scan Conversion

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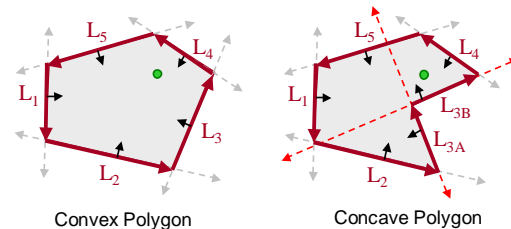 - Concave
  - Self-intersecting
  - Holes

What problems do we encounter with arbitrary polygons?

Polygon Scan Conversion

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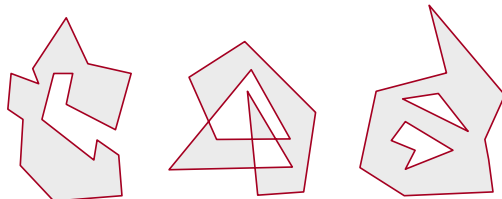 - Triangle method works only for convex polygons

Convex Polygon   Concave Polygon

Inside Polygon Rule

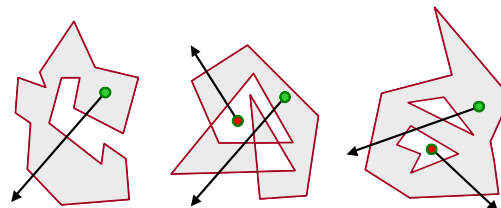- What is a good rule for which pixels are inside?

Concave   Self-Intersecting   With Holes

Inside Polygon Rule

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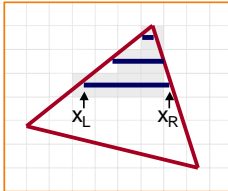

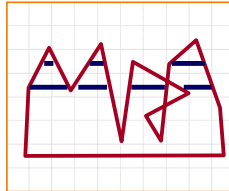Concave   Self-Intersecting   With Holes

Polygon Sweep-Line Algorithm

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

Triangle   Polygon

## Polygon Sweep-Line Algorithm

```
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);
   }
}
```
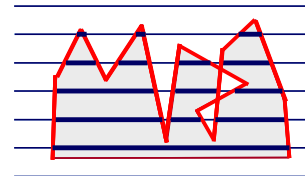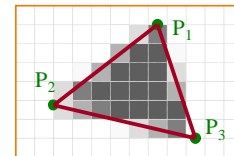
Hardware Scan Conversion

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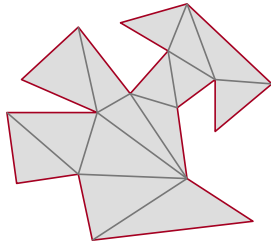
Hardware Antialiasing

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

P_1
P_2
P_3

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?

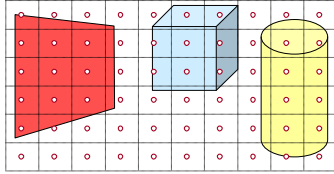Emphasis on methods that can be implemented in hardware

COP
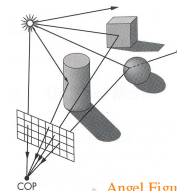
- Angel Figure 6.34

4

Ray Casting

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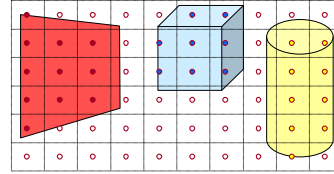 - When is this unnecessary?

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

Polygon Shading

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

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

Polygon Shading Algorithms

- Flat Shading
- Gouraud Shading
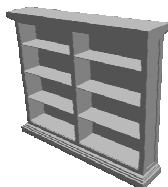- Phong Shading

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
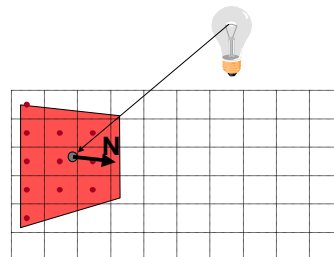

$$I = I_E + K_A I_{AL} + \sum_i (K_D (N \bullet L_i) I_i + K_S (V \bullet R_i)^n I_i)$$
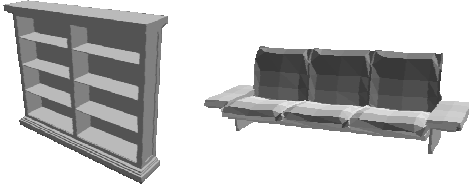
Flat Shading

- 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
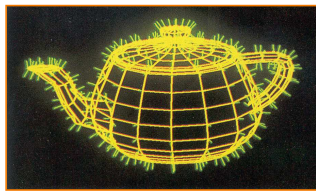 - Not so good for smooth surfaces

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?
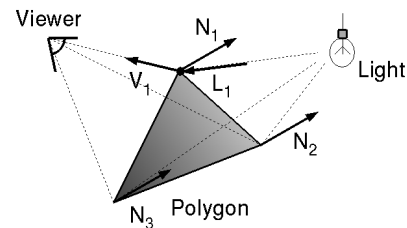

Watt Plate 7

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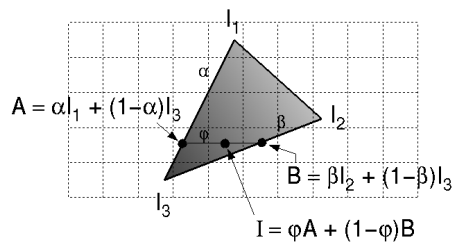
Gouraud Shading

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

Viewer, N_1, V_1, L_1, Light, N_2, N_3, Polygon

Gouraud Shading

- Bilinearly interpolate colors at vertices down and across scan lines

I_1

α

$A = \alpha I_1 + (1-\alpha) I_3$

φ β I_2

$B = \beta I_2 + (1-\beta) I_3$

I_3

$I = \varphi A + (1-\varphi) B$

Gouraud Shading

- Smooth shading over adjacent polygons
  - Curved surfaces
  - Illumination highlights
  - Soft shadows
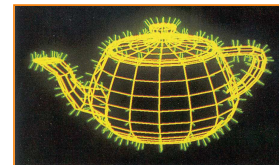
Mesh with shared normals at vertices

Watt Plate 7

Gouraud Shading

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

Flat Shading   Gouraud Shading
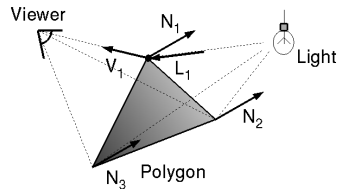
Polygon Shading Algorithms

- Flat Shading
- Gouraud Shading
- **Phong Shading**

Phong Shading

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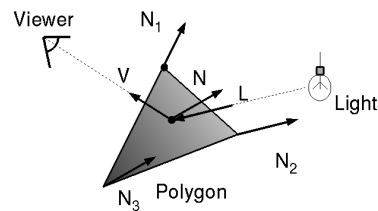
Viewer

N_1

V_1 L_1 Light

N_2

N_3 Polygon

$$I = I_E + K_A I_{AL} + \sum_i (K_D (N \bullet L_i) I_i + K_S (V \bullet R_i)^n I_i)$$
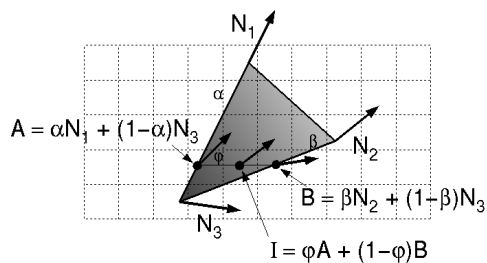
Phong Shading

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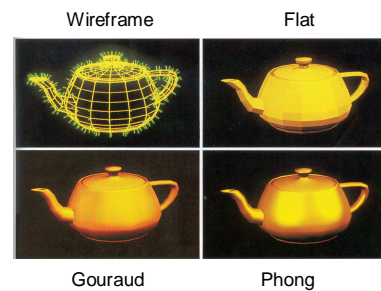 - Approximate surface normals for points inside polygons by bilinear interpolation of normals from vertices

Viewer

N_1

V N L Light

N_2

N_3 Polygon

Phong Shading

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

N_1

α

$A = \alpha N_1 + (1-\alpha)N_3$

φ β N_2

$B = \beta N_2 + (1-\beta)N_3$

N_3

$I = \varphi A + (1-\varphi)B$

Polygon Shading Algorithms

Wireframe   Flat

Gouraud   Phong
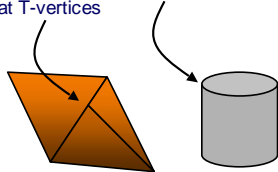
Shading Issues

- 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   ↑ Less expensive
  - Gouraud
  - Phong
  - Ray casting   ↓ More accurate
- Key ideas:
  - Sampling and reconstruction
  - Spatial coherence