Scan Conversion

Thomas Funkhouser
Princeton University
C0S 426, Fall 1999

Where Are We Now?

- Part I: Image processing

  ← You are here
- Part II: Rendering
- Part III: Modeling
- Part IV: Animation

Rendering

- Generate an image from geometric primitives

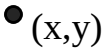
2D Geometric Primitives

- How are these shapes described in a computer?
  - Point
  - Vector
  - Line
  - Ray
  - Triangle
  - Polygon
  - Quadric
  - Spline
  - etc.

2D Point

- Specifies a location
  - Represented by two coordinates
  - Infinitely small

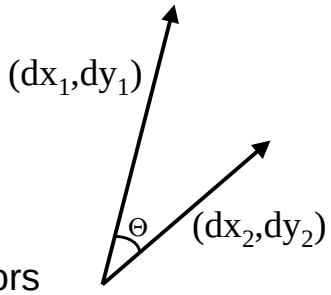
```
typedef struct {
    Coordinate x;
    Coordinate y;
} Point;
```



2D Vector

- Specifies a direction and a magnitude
  - Represented by two coordinates
  - Magnitude ||V|| = sqrt(dxdx + dydy)
  - Has no location

```
typedef struct {
    Coordinate dx;
    Coordinate dy;
} Vector;
```

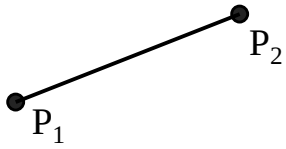


- Dot product of two 2D vectors
  - $V_1 \cdot V_2 = dx_1dx_2 + dy_1dy_2$
  - $V_1 \cdot V_2 = ||V_1|| \, ||V_2|| \cos(\Theta)$

2D Line Segment

- Specifies a linear combination of two points
  - Parametric representation:
    - » $P = P_1 + t\,(P_2 - P_1), \quad (0 \le t \le 1)$

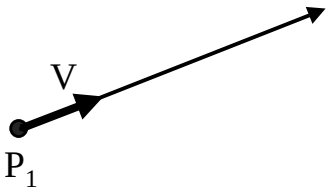
```
typedef struct {
    Point P1;
    Point P2;
} Segment;
```



2D Ray

- Line segment with one endpoint at infinity
  - Parametric representation:
    - » $P = P_1 + t\,V, \quad (0 <= t < \infty)$

```
typedef struct {
    Point P1;
    Vector V;
} Line;
```



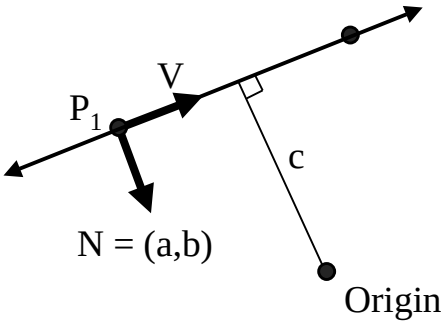
2D Line

- Line segment with both endpoints at infinity
  - Parametric representation:
    - » $P = P_1 + t\,V, \quad (-\infty < t < \infty)$

```
typedef struct {
    Point P1;
    Vector V;
} Line;
```

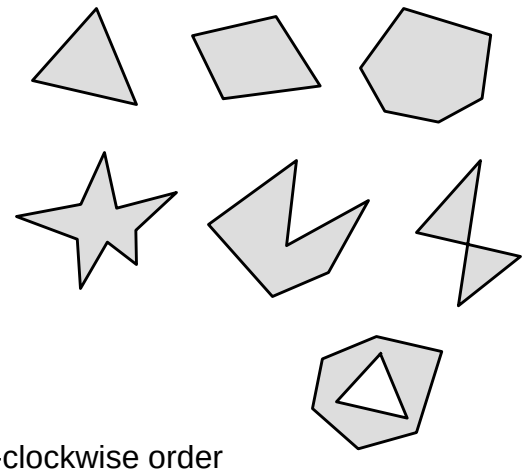
  - Implicit representation:
    - » $P \cdot N + c = 0$, or
    - » $ax + by + c = 0$

```
typedef struct {
    Vector N;
    Distance c;
} Line;
```



2D Polygon

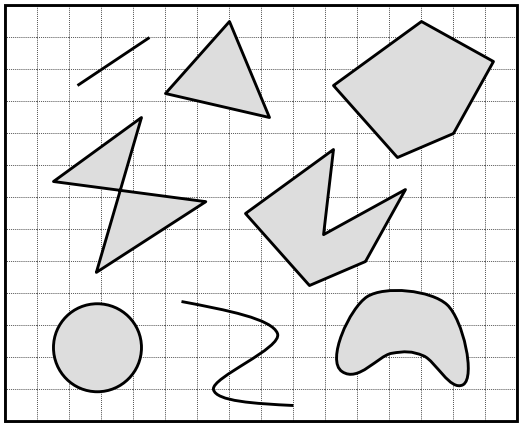
- Area “inside” a sequence of points
  - Triangle
  - Quadrilateral
  - Convex
  - Star-shaped
  - Concave
  - Self-intersecting
  - Holes



```
typedef struct {
    Point *points;
    int npoints;
} Polygon;
```

Points are in counter-clockwise order

2D Rendering



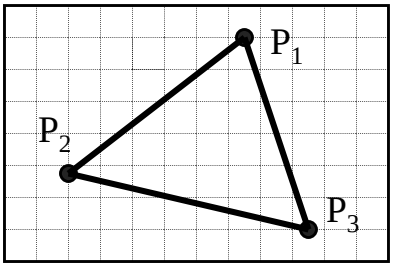
- Create an image from a set of 2D geometric primitives

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

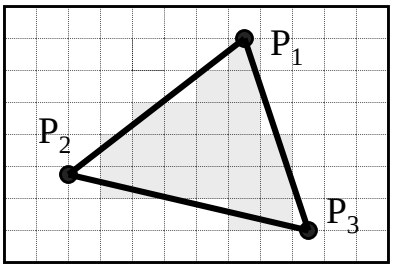


Scan Conversion

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

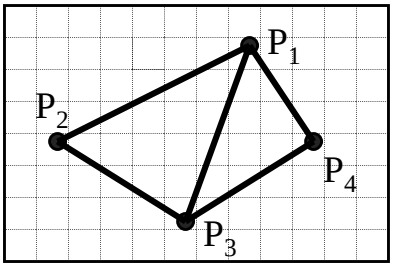
```
void SetPixel(int x, int y, Color rgba)
```

- Example: Filling the inside of a triangle



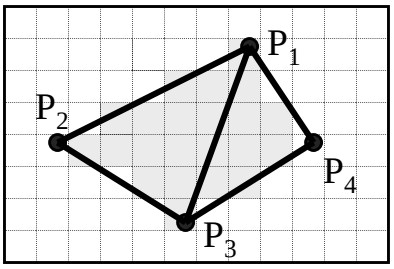
Triangle Scan Conversion

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



Triangle Scan Conversion

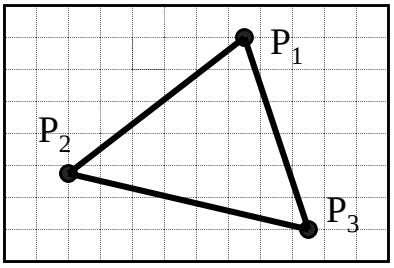
- Properties of a good algorithm
  - Symmetric
  - Straight edges
  - Antialiased edges
  - No cracks between adjacent primitives
  - MUST BE FAST!



Simple Algorithm

- Color all pixels inside triangle

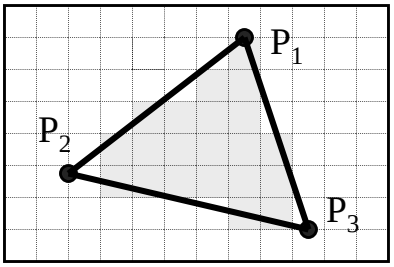
```
void ScanTriangle(Triangle T, Color rgba){
  for each pixel P at (x,y){
     if (Inside(T, P))
        SetPixel(x, y, rgba);
  }
}
```



Simple Algorithm

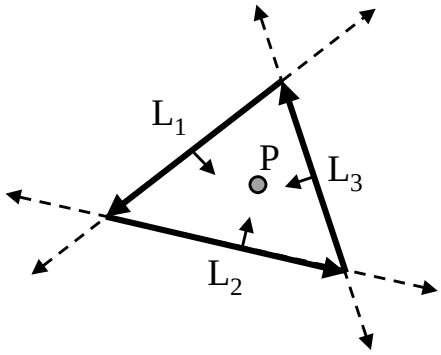
- Color all pixels inside triangle

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



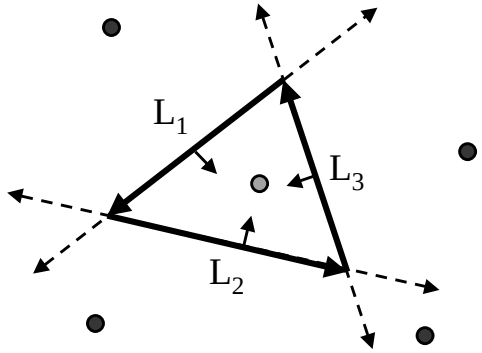
Inside Triangle Test

- 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

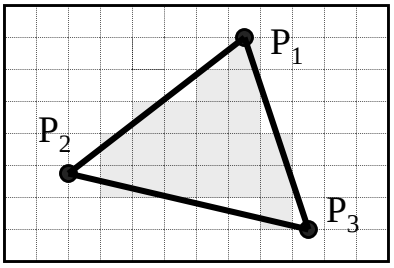

```
Boolean Inside(Triangle T, Point P)
{
  for each boundary line L of T {
     Scalar d = L.a*P.x + L.b*P.y + L.c;
     if (d < 0.0) return FALSE;
  }
  return TRUE;
}
```



Simple Algorithm

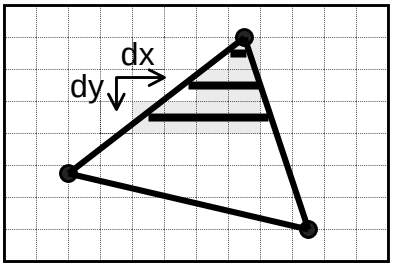
- What is bad about this algorithm?

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



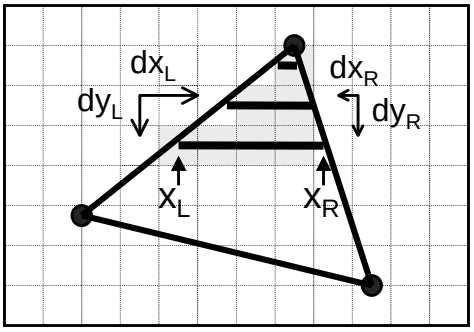
Triangle Sweep-Line Algorithm

- 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



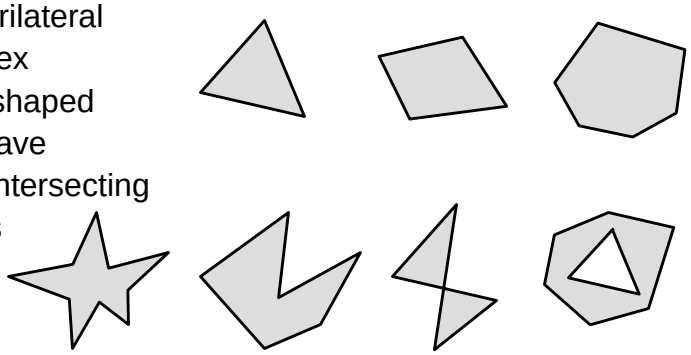
Triangle Sweep-Line Algorithm

```
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;
  }
}
```



Polygon Scan Conversion

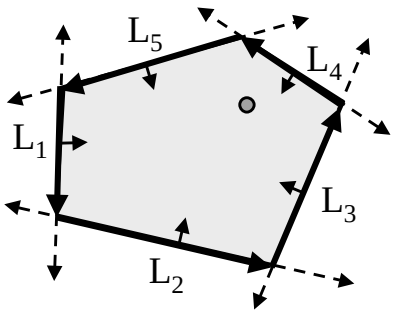
- Fill pixels inside a polygon
  - Triangle
  - Quadrilateral
  - Convex
  - Star-shaped
  - Concave
  - Self-intersecting
  - Holes



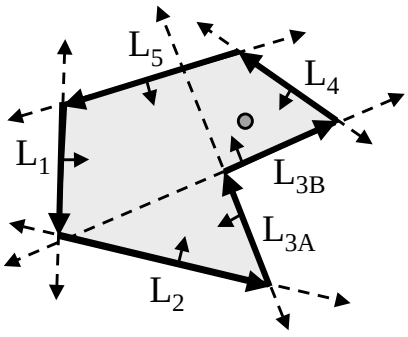
What problems do we encounter with arbitrary polygons?

Polygon Scan Conversion

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

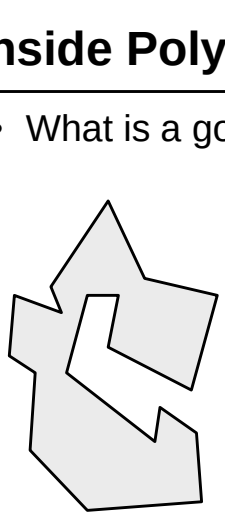


Convex Polygon

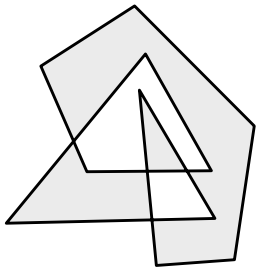


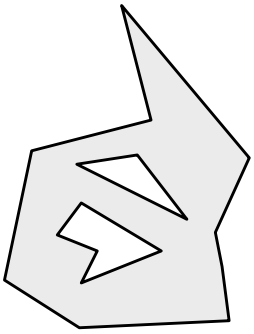
Concave Polygon

Inside Polygon Rule



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





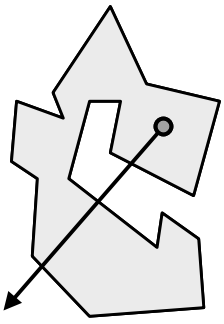
Concave

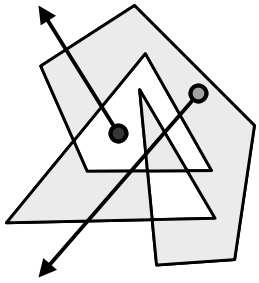
Self-Intersecting

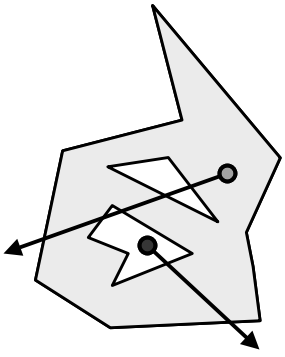
With Holes

Inside Polygon Rule

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







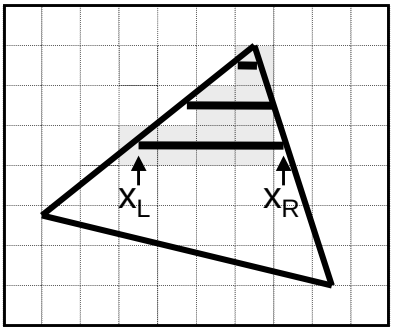
Concave

Self-Intersecting

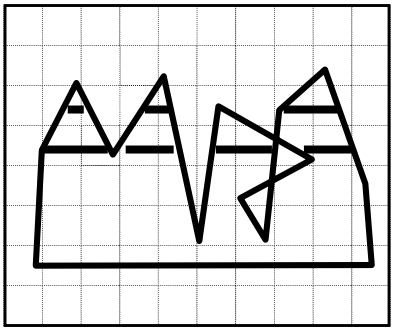
With Holes

Polygon Sweep-Line Algorithm

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



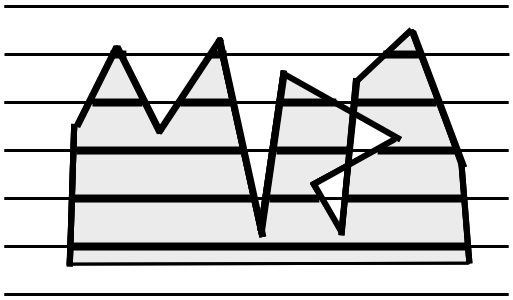
Triangle



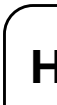
Polygon

Polygon Sweep-Line Algorithm

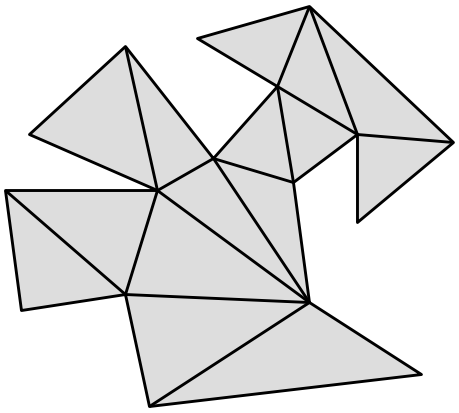
```
void ScanPolygon(Triangle T, Color rgba){
  sort edges by maxy
  make empty "active edge list"
  for each scanline (top-to-bottom) {
     insert/remove edges from "active edge list"
     update x coordinate of every active edge
     sort active edges by x coordinate
     for each pair of active edges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



Hardware Scan Conversion

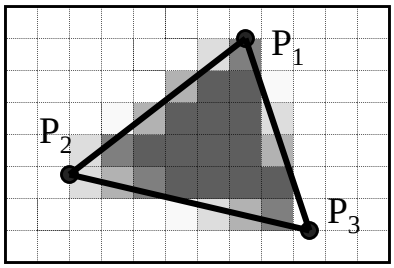


- Convert everything into triangles
  - Scan convert the triangles



Antialiasing

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



Summary

- 2D geometric primitives
  - Point, vector, ray, line, polygon
  - Similar scan conversion algorithm for each one
- 2D polygon scan conversion
  - Paint pixels inside polygon
  - Sweep-line algorithm
    - » Key idea: scanline coherence
  - Straight-forward for triangles