

CM0304 Graphics

III Real-Time Rendering

III.5 Polygon Drawing

F. C. Langbein

School of Computer Science
Cardiff University



Version 2.3

Overview

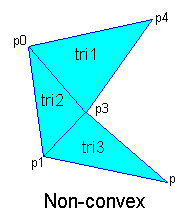
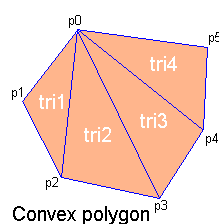
- Inside polygon test
- Triangle/polygon scan-line algorithm
- Micropolygons and Warnock's algorithm

F. C. Langbein, CM0304 Graphics – III Real-Time Rendering; III.5 Polygon Drawing

1

Polygons and Triangles

- Polygons can *approximate* any 2D shape (3D surface)
 - Number of polygons determines quality
- Any polygon can be *decomposed into triangles*
 - Simple for convex polygons
 - Harder for non-convex polygons (may have to introduce additional vertices)

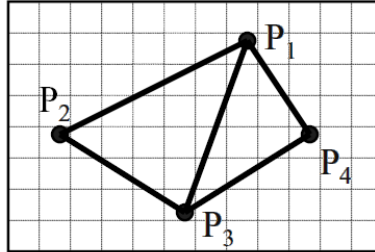


F. C. Langbein, CM0304 Graphics – III Real-Time Rendering; III.5 Polygon Drawing

2

Desirable Properties

- Properties of a good algorithm for drawing polygons
 - *Symmetric* — independent of orientation
 - *Straight* edges
 - *No cracks* between adjacent primitives
 - *FAST*



A Simple Approach

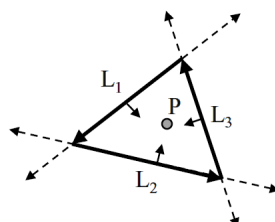
- Colour all pixels inside polygon

```
public void draw_polygon (Polygon P, Color c) {
    for each pixel (x,y) (in bounding box) {
        if (P.inside (x,y))
            raster.set_pixel (c.getRGB (), x, y);
    }
}
```

- Need *inside polygon test*
- Expensive!
 - Have to find a way to *reduce* number of tested pixels

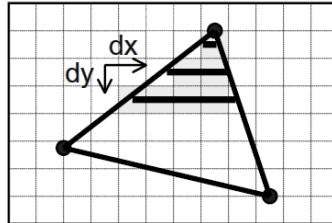
Inside Triangle Test

- A point is inside a triangle if it is in the *positive half-space* of all three boundary lines
 - Triangle vertices are ordered *counter-clockwise*
 - Point must be on the *left side* of every boundary line
- *Line equation*: $L_k^t p - d_k = 0$ in 2D (similar to plane in 3D)
 - Half-space test: $L_k^t p - d_k > 0$



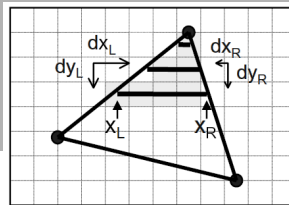
Triangle Scan-Line Algorithm

- 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 x coordinates incrementally
- Note, all triangles are convex



Triangle Scan-Line Algorithm

```
public void scan_triangle (Triangle T, Color c) {
    for each edge pair of T {
        initialise xL, xR;
        compute dxL / dyL and dxR / dyR
        for each scan-line at y
            for (int x = xL; x <= xR; ++x)
                raster.set_pixel (c, x, y);
        xL += dxL / dyL;
        xR += dxR / dyR;
    }
}
```

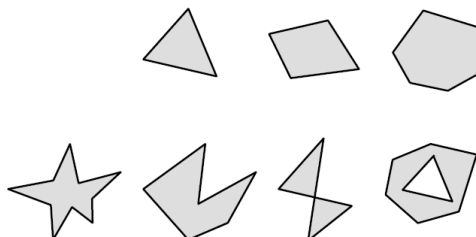


- Similar to drawing two lines
- Can be optimised! (see Bresenham)

General 2D Polygons

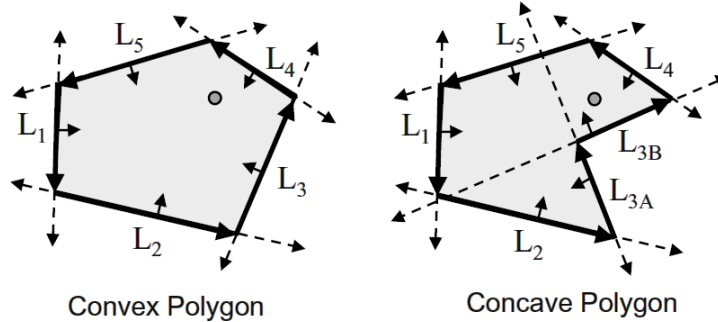
- Fill pixels inside a polygon

- Triangle
- Quadrilateral
- Convex
- Star-shaped
- Concave
- Self-intersecting
- Holes



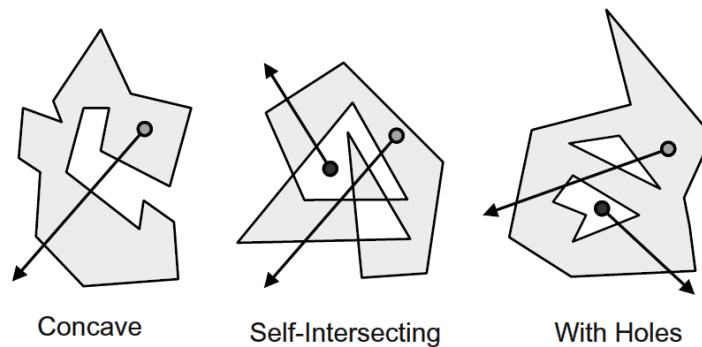
Inside Polygon Test

- Need a different test for points inside polygon
 - Triangle method works *only for convex polygons*



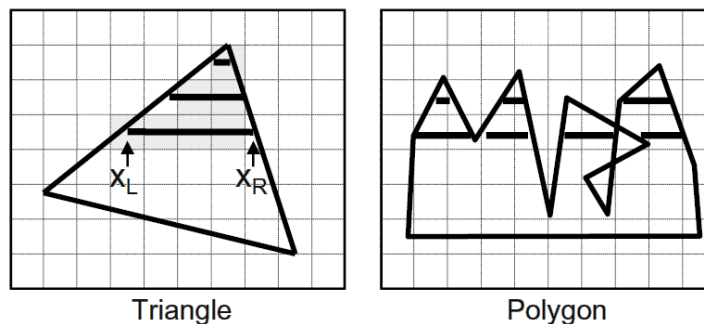
Inside Polygon Rule

- *Odd-parity rule*
 - Any ray from point inside polygon to infinity *crosses odd number of edges*



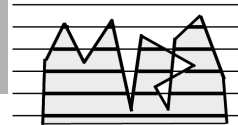
Polygon Scan-Line Algorithm

- Incremental algorithm to find *spans*
 - Determine insideness with odd parity rule
 - Take advantage of *scan-line coherence*



Polygon Scan-Line Algorithm

```
void scan_polygon (Polygon P, Color c) {
    sort edges by maximum y coordinate
    create empty "active edge list" A
    for each scan-line s = top to bottom {
        insert edges with y_max = s into A
        remove edges with y_min > s from A
        update x coordinate of every active edge
            (see DDA, Bresenham, ...)
        sort active edges by x coordinate
        for each pair of active edges (left-to-right)
            draw_span (x_k, x_{k+1}, y, c)
    }
}
```

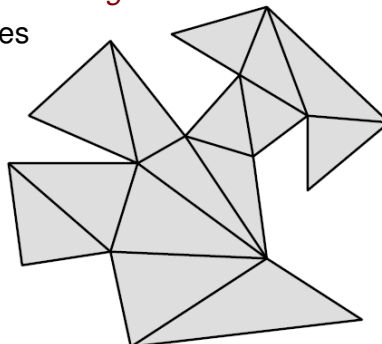


Polygon Scan-Line Notes

- If scan-line goes through a vertex of the polygon
 - *Vertices at extrema* have to be counted twice
(at extremum vertex: y direction of both edges is either up or down)
- *y direction coherence*
 - Active edges only change when y of scan-line is at y of an edge vertex
 - Intersection position of edge with scan-line can be computed similar to Bresenham

Hardware Scan Conversion

- OpenGL polygons are convex
- For hardware scan conversion:
 - Convert everything into *triangles*
 - Scan convert triangles

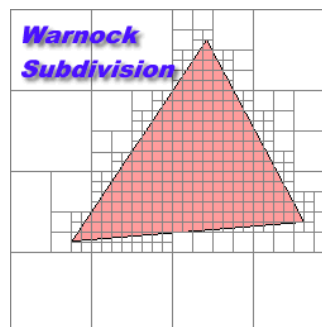
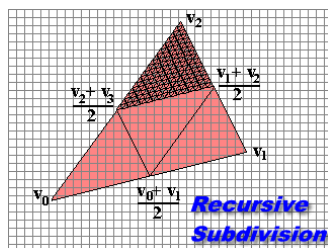


Scan-Line and Visible Surfaces

- *Combine* scan-line with visible surface detection
 - For each scan line consider all polygons intersecting it
 - Whenever more than one polygon is active over a span:
 - Do depth computations to determine polygon visibility
- Can take advantage of y-coherence between scan-lines:
 - Use a Z buffer for a single scan-line
- Full Z buffer
 - Can draw polygons in arbitrary sequence
 - with standard scan-line algorithm and depth computations / comparisons as in z-buffer algorithm

Alternatives

- Alternatives to scan-line approach
 - Recursive primitive subdivision (*micropolygons*)
 - Recursive screen subdivision (*Warnock's algorithm*)



Summary

- For what can polygons and triangles be used? What is the advantage of triangles over general polygons?
- Describe two tests to determine whether a pixel is inside a convex polygon. Describe a test to determine whether a pixel is inside a general polygon.
- Describe the scan-line algorithm for triangles / general polygons.
- How can we combine the scan-line algorithm with visible surface detection?