Intro to Polygons

Objectives

• Introduce polygon and terminology

• Polygon data structures

• Filling polygons on the 2D image plane

• Polygons in 3D
Polygons

• Multi-sided planar element composed of edges and vertices.

• Vertices (singular vertex) are represented by points

• Edges connect vertices as line segments

Simple Non-simple

• Simple polygons - no edges cross
• Non-simple - some edges cross
Polygons

- **Convex Polygon** - a polygon that has no included angles larger than 180 degrees
- **Concave Polygon** - has at least one included angle greater than 180 degrees

Test for convexity: any line segment drawn between two points inside a polygon must remain inside the polygon if the polygon is convex.
Polygons

- A concave polygon can be broken up into two or more convex polygons

---

Polygons and OpenGL

OpenGL assumes all polygons are simple and convex

- Types in GL for single polygons
  - GL_POLYGON, GL_QUAD, GL_TRIANGLES

- Types in GL for groups of triangles and quads
  - GL_TRIANGLE_STRIP, GL_QUAD_STRIP, GL_TRIANGLE_FAN

- Strips and fans reduce the number of vertices specified for the same number of basic elements
Filling Polygons in 2D

• Assume that all polygons lie on the image plane (2D), then scan conversion is to fill in pixels lying within a closed polygon and to do so efficiently

• To do this, algorithm needs to determine if a pixel is inside or outside a polygon

• How do we do this?

Filling Polygons

• Scan Line Approaches:
  - Inside test: Point P is inside a polygon iff a scanline intersects polygon edges an odd number of times moving from P in either direction
Filling Polygons

• **Inside test** special cases:
  - Horizontal edges can be ignored
  - Use **Min-max test** when scanline passes thru vertex

Min-max:

count twice if slope changes sign  
count once otherwise

---

Filling Polygons

• **Inside test**: To fill polygons, one scanline at a time
  determine which pixels are inside and set "on"
Filling Polygons

• Faster is to fill polygons, one scanline at a time
determine which pixels intersect and sort by edge

• Scanline fill algorithm
  For each scanline:
  Find intersections
  Sort intersections
  Fill in pixels between
  pairs of intersections

But, intersection calculations are expensive

Filling Polygons

• Faster still, use edge coherence - Many edges that
intersect scanline $s$ intersect scanline $s+1$

• Compute scanline intersection:
  $y = mx + b$ and $y_s = s$, what is $x_s$?
  $s = mx_s + b$ or $x_s = (s-b)/m$

• Compute difference in next scanline for intersection
  So, for $y_{s+1} = s+1$,
  $s+1 = mx_{s+1} + b$ or $x_{s+1} = (s +1 - b)/m$

  and, $x_{s+1} = x_s + 1/m$  <= USE THIS TO UPDATE
**Edge Tables**

- **Edge coherence**: using an edge table as a convenient data structure to take advantage of coherence
- **Edge table** is a linked-list for each appropriate scanline corresponding to the lower vertex, and storing:
  - \(y_{\text{upper}}\), last scanline edge
  - \(x_{\text{lower}}\), start x coordinate of edge
  - \(1/m\), increment for x between scanlines

---

**Edge Table (ET) for given polygon example**

![Diagram of an edge table and a polygon example](image)
• During scan conversion, we use a second structure, the Active Edge List - (AEL) linked-list of edges on the current scanline, y. Edge info included:
  - $y_{\text{upper}}$ for last scanline to consider
  - $x$ for current value of intersection
  - $1/m$ to increment $x$

**Active Edge Table** for polygon example, given scanline is at $y = 6$
• Using these data structures, the scan conversion algorithm becomes:

For each scanline s
   add edges from ET for s to AEL
   if AEL is not empty
      sort AEL by x
      fill pixels between pairs of edges
      delete finished edges
      update x values for edges in AEL

Edge tables are commonly used in graphics pipeline and may be implemented in the graphics card.

But, scanline approaches are not the only method for performing polygon fills.

Other approaches, such as flood-fill, have fallen out of favor but are easy to implement.
Flood Fill

- Alternative, to scan-line approaches, flood fill starts with scan-converted lines drawn around the polygon

```
flood_fill (x,y)
    if(pixel(x,y) is off)
        turn pixel is on
        flood_fill(x+1,y)
        flood_fill(x-1,y)
        flood_fill(x,y+1)
        flood_fill(x,y-1)
    }
```

Polygons in 3D

- Mathematically define polygons
  - Vectors, vector operations, normals

- Polygons in 3D
Defining 3D Polygons

• Polygons are 2D, truncated planes in 3-space

\[ Ax + By + Cz + D = 0 \]

• How do we express it given the graphical elements?

Start with the generic form of a plane
Ax + By + Cz + D = 0

Defining 3D Polygons

• Vertices and edges are ordered, CW or CCW, usually counter clockwise
• Therefore, edges are directed line segments and may be treated as vectors
Vectors

• Definition: a vector is a quantity with two attributes
  - Direction
  - Magnitude

• Examples include
  - Force
  - Velocity
  - Directed line segments
    - Most important example for geometry

Vector Operations

• Every vector has an inverse
  - Same magnitude but points in opposite direction
• Every vector can be multiplied by a scalar
• Special case: The zero vector
  - Zero magnitude, undefined direction
• The sum of any two vectors is a vector
  - Use head-to-tail axiom
Vector Operations

**dot product**
- also called *inner product*

\[
\begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix} \cdot \begin{bmatrix}
  a \\
  b \\
  c
\end{bmatrix} = x \cdot a + y \cdot b + z \cdot c
\]

Note, dot product results in a scalar

Use it to find the length of a vector:

\[ \sqrt{v \cdot v} = ||v|| \]

Or to *normalize* (set length to one) as in \( v' = v / ||v|| \)

---

Vector Operations

Dot product can also be used to define angles:
For example, assume \( u \) & \( v \) are unit length

\[
\text{projection } u \cdot v
\]

\[ \theta = \cos^{-1}( u \cdot v ) \]

*Also, can be generalized for non-normal vectors*
Vector Operations

Cross Product

- Cross product results in a vector
- Apply the right hand rule:
  (curl fingers from $u$ to $v$; thumb points to $u \times v$)

Vectors and planes

- Every plane has a vector $n$ normal (perpendicular, orthogonal) to it
- From two vectors that line on the plane, we can use the cross product to find the normal, $n = u \times v$
**Vectors and planes**

- In our plane equation: \( Ax + By + Cz + D = 0 \), \([A,B,C]^T\) = the normal of the plane
- From the edges find two vectors
- Use the cross product to find the normal, giving us A, B, C

Then, to find D, plug in any vertex into the equation and solve for D

---

**3D Polygons**

- Once we know our plane equation: \( Ax + By + Cz + D = 0 \),
  we still need to manage the truncation which leads to the polygon itself

Functionally, we will need to do this to know if a point lies in a polygon or not, for example
3D Polygons

• How do we know if \((x_i,y_i,z_i)\) is in the polygon?

To do this, we can project the 3D polygon into 2D and see if the point is in the 2d using the inside test.
3D Polygons

• Project 3d to 2d based on largest of A,B,C

This example:
Z (or C) is principal component of \( N \), normal so project on to xy-plane
BONUS: More 2D Polygon tests

- Recall the *Inside* test: Point P is inside a polygon iff a scanline intersects polygon edges an odd number of times moving from P in either direction.

BUT, for Convex Polygons we can test *in other ways*.

BONUS: More 2D Polygon tests

- Another *convex* polygon test: Point P is inside a convex polygon (defined in CW or CCW order) iff: for each edge, the *cross-product* of the vector formed by the edge and vector from the second vertex (of the edge) to P is in the same direction as all other edges' same cross-product.
**BONUS: More 2D Polygon tests**

- **Convex polygons**  Point P is also inside a convex polygon (defined in CW or CCW order) iff:

  The area of the polygon = $\sum$ the areas of the triangles formed by each edge and point P