Polygon Scan Conversion and Z-Buffering
• Rasterization takes shapes like triangles and determines which pixels to fill.
First approach:

1. Polygon Scan-Conversion
   • Rasterize a polygon scan line by scan line, determining which pixels to fill on each line.
Filling Polygons

Second Approach:

2. Polygon Fill
   - Select a pixel inside the polygon. Grow outward until the whole polygon is filled.
Polygon Types

Convex

For every pair of points within a convex polygon, the line segment connecting them is also completely enclosed within the polygon.

Not Convex
Why Polygons?

• You can approximate practically any surface if you have enough polygons.
• Graphics hardware is optimized for polygons (and especially triangles)
Which Pixels to Fill?

- Pixels entirely inside polygon → Yes
- Pixels partially inside → ?
- Convention: only fill pixels whose centers are within the polygon
  - This is an imperfect solution.
Spatial Coherence

- Spatial Coherence:
  - If one pixel is filled, its neighbors likely are also.
- Use spatial coherence to construct an efficient algorithm.
- Process each scan line in order from left to right.
Coherence

Scan-line

Edge

Span
Polygon Scan Conversion

- Intersection Points
- Other points in the span
Polygon Scan Conversion

- Process each scan line
  1. Find the intersections of the scan line with all polygon edges.
  2. Sort the intersections by $x$ coordinate.
  3. Fill in pixels between pairs of intersections using an odd-parity rule.
     - Set parity even initially.
     - Each intersection flips the parity.
     - Draw when parity is odd.
Special Cases: Vertices

• How do we count the intersecting vertex in the parity computation?

• Count it zero or two times.
Special Cases: Vertices

• What about:

• Here it counts once.
Special Cases: Vertices

• Each edge has two vertices.
  • The $y_{min}$ vertex is the vertex with the lower $y$ coordinate.
• Vertices only count when they are the lower $y_{min}$ vertex for an edge.
Special Cases: Vertices

- Both cases now work correctly
Horizonal Edges

- What if neither vertex is lower? Which is $y_{min}$?

Don’t count either vertex in the parity calculation!
Top Spans of Polygons

- Effect of only counting $y_{\text{min}}$:
  - Top spans of polygons are not drawn

- If two polygons share this edge, it is not a problem.
Shared Polygon Edges

• What if two polygons share an edge?

• Solution:
  • Span is closed on left and open on right  \( (x_{\text{min}} \leq x < x_{\text{max}}) \)
  • Scan lines closed on bottom and open on top  \( (y_{\text{min}} \leq y < y_{\text{max}}) \)
Polygon Scan-Conversion

• Process for scan converting polygons
  • Process one polygon at a time.
  • Store information about every polygon edge.
  • Compute spans for each scan line.
  • Draw the pixels between the spans.
• This can be optimized using an edge table.
• One entry in the edge table per polygon edge.
• Perform scan conversion one scan line at a time.
• For each new scan line, consult the edge table to determine if it intersects any new edges.

This is the beginning of your code for project 1
Computing Intersections

• For each scan line, we need to know if it intersects the polygon edges.
• It is expensive to compute a complete line-line intersection computation for each scan line.
• After computing the intersection between a scan line and an edge, we can use that information in the next scan line.
Scan Line Intersection

Intersection points needed

Previous scan line $y_i$

Intersection points from previous scan line

Current scan line $y_{i+1}$

Polygon Edges
Scan Line Intersection

Use edge coherence to incrementally update the $x$ intersections.

We know $y = mx + b$, $m = \frac{y_{\text{max}} - y_{\text{min}}}{x_{@\text{ymax}} - x_{@\text{ymin}}}$.

Each new scan line is 1 greater in $y$

$$y_{i+1} = y_i + 1$$

We need to compute $x$ for a given scan line,

$$x = \frac{y - b}{m}$$
So, \[ x_i = \frac{y_i - b}{m}, \quad x_{i+1} = \frac{y_{i+1} - b}{m} \]

and \[ y_{i+1} = y_i + 1 \]

then \[ x_{i+1} = \frac{y_i + 1 - b}{m} = \frac{y_i - b}{m} + \frac{1}{m} = x_i + \frac{1}{m} \]

This is a more efficient way to compute \( x_{i+1} \).
Edge Tables

- We will use two different edge tables:
  - Active Edge Table (AET)
    - Stores all edges that intersect the current scan line.
  - Global Edge Table (GET)
    - Stores all polygon edges.
    - Used to update the AET.
Active Edge Table

- Table contains one entry per edge intersected by the current scan line.
- At each new scan line:
  - Compute new intersections for all edges using the formula: $x_{i+1} = x_i + \frac{1}{m}$
  - Add any new edges intersected.
  - Remove any edges no longer intersected.
- To efficiently update the AET, we will maintain a global edge table (GET)
Global Edge Table

- Table contains information about all polygon edges.
- It has one bucket for each scan line.
  - Each bucket stores a list of edges whose $y_{\text{min}}$ value is the scan line $y$ value.
  - Each edge is found in only one bucket.
- Each entry in the GET contains
  - The edge’s $y_{\text{max}}$ value.
  - The $x_{@y_{\text{min}}}$ value (the $x$ value at the $y_{\text{min}}$ point)
  - The $x$ increment value ($1/m$)
**Global Edge Table Example**

- **GET Entries** \( (y_{\text{max}}, x_{\text{ymin}}, 1/m) \):
  - \( AB \rightarrow (5, 2, \frac{1}{4}) \)
  - \( BC \rightarrow (6, 3, 3) \)
  - \( CD \rightarrow (8, 6, -\frac{3}{2}) \)
  - \( DE \rightarrow (8, 0, \frac{3}{4}) \)
  - \( EA \rightarrow (4, 2, -\frac{2}{3}) \)

- **Edges**:
  - \( AB = (2, 1), (3, 5) \)
  - \( BC = (3, 5), (6, 6) \)
  - \( CD = (6, 6), (3, 8) \)
  - \( DE = (3, 8), (0, 4) \)
  - \( EA = (0, 4), (2, 1) \)

Why don’t we store \( y_{\text{min}} \)?
Global Edge Table Example

Indexed by scan line

Place entries into the GET based on the $y_{\text{min}}$ values.

\begin{align*}
\text{AB} &= (2, 1), (3, 5) \\
\text{BC} &= (3, 5), (6, 6) \\
\text{CD} &= (6, 6), (3, 8) \\
\text{DE} &= (3, 8), (0, 4) \\
\text{EA} &= (0, 4), (2, 1)
\end{align*}
Active Edge Table Example

- The active edge table stores information about the edges that intersect the current scan line.
- Entries are
  - The $y_{\text{max}}$ value of the edge
  - The $x$ value where the edge intersects the current scan line.
  - The $x$ increment value ($1/m$)
- Entries are sorted by $x$ values.
Active Edge Table Example

1. The $y_{\text{max}}$ value for that edge
2. The $x$ value where the scan line intersects the edge.
3. The $x$ increment value (1/m)

AB = (2, 1), (3, 5)
BC = (3, 5), (6, 6)
CD = (6, 6), (3, 8)
DE = (3, 8), (0, 4)
EA = (0, 4), (2, 1)

(y_{\text{max}}, x, 1/m)

AET

4, 2/3, -2/3

5, 5/2, 1/4

EA

AB
Active Edge Table Example

1. The $y_{\text{max}}$ value for that edge
2. The x value where the scan line intersects the edge.
3. The x increment value ($1/m$)

\[
\begin{align*}
\text{AET} & \quad (y_{\text{max}}, x, 1/m) \\
\begin{array}{c}
\text{DE} \\
\text{AB}
\end{array} & \quad \begin{array}{c}
4.82/3, 3/43 \\
5.15/4, 1/4
\end{array}
\end{align*}
\]

In the GET, edge DE is stored in bucket 4, so it gets removed from the AET.

New x value for AB is $5/2 + 1/4 = 11/4$

\[
\begin{align*}
\text{AB} & = (2, 1), (3, 5) \\
\text{BC} & = (3, 5), (6, 6) \\
\text{CD} & = (6, 6), (3, 8) \\
\text{DE} & = (3, 8), (0, 4) \\
\text{EA} & = (0, 4), (2, 1)
\end{align*}
\]
Active Edge Table Example

1. The $y_{\text{max}}$ value for that edge
2. The $x$ value where the scan line intersects the edge.
3. The $x$ increment value ($1/m$)

\[
\begin{array}{l}
\text{AB} = (2, 1), (3, 5) \\
\text{BC} = (3, 5), (6, 6) \\
\text{CD} = (6, 6), (3, 8) \\
\text{DE} = (3, 8), (0, 4) \\
\text{EA} = (0, 4), (2, 1)
\end{array}
\]

Scan Line 5

\[
\begin{array}{c}
8, 0, 3/4 \\
5, 11/4, 1/4 \\
8, 3/4, 3/4 \\
\end{array}
\]

Increment $x = 0 + 3/4$

Remove AB and BC since it is in the GET at $y_{\text{min}} = 5$. 

$y_{\text{max}}$, $x$, $1/m$
Active Edge Table Example

Scan Line 6 – Your Turn

AET

AB = (2, 1), (3, 5)
BC = (3, 5), (6, 6)
CD = (6, 6), (3, 8)
DE = (3, 8), (0, 4)
EA = (0, 4), (2, 1)
One More Example

• What is the global edge table for this polygon?
  • 1. Find Edges

\[
\begin{align*}
AB &= ? \\
BC &= ? \\
CA &= ?
\end{align*}
\]
One More Example

- What is the global edge table for this polygon?
  - 2. Compute GET Entries

\[
\begin{align*}
AB &= (3, 3), (7, 8) \rightarrow (8, 3, 4/5) \\
BC &= (7, 8), (1, 6) \rightarrow (8, 1, 3) \\
CA &= (1, 6), (3, 3) \rightarrow (6, 3, -2/3)
\end{align*}
\]
One More Example

- What is the global edge table for this polygon?
  - 3. Place entries into GET

\[
\begin{align*}
AB &= (3, 3), (7, 8) \rightarrow (8, 3, 4/5) \\
BC &= (7, 8), (1, 6) \rightarrow (8, 1, 3) \\
CA &= (1, 6), (3, 3) \rightarrow (6, 3, -2/3)
\end{align*}
\]
One More Example

- What is the global edge table for this polygon?

3. Place entries into GET

\[
\begin{align*}
AB &= (3, 3), (7, 8) \rightarrow (8, 3, 4/5) \\
BC &= (7, 8), (1, 6) \rightarrow (8, 1, 3) \\
CA &= (1, 6), (3, 3) \rightarrow (6, 3, -2/3)
\end{align*}
\]
Scan Line Algorithm

1. Add polygon edges to the Global Edge Table (GET)
2. Set $y$ to the smallest $y$ coordinate in the GET.
3. Initially, set the Active Edge Table (AET) to be empty.
4. Repeat until the AET and GET are empty:
   a. Add edges from the GET to the AET in which $y_{\text{min}} = y$.
   b. Remove edges from the AET in which $y_{\text{max}} = y$.
   c. Sort AET on $x$.
   d. Fill pixels between pairs of intersections in the AET
   e. For each edge in the AET, replace $x$ with $x + 1/m$
   f. Set $y = y + 1$ to move to the next scan line
Filling Techniques

• A second approach to drawing polygons is to use a filling technique, rather than scan conversion.
• Pick a point inside the polygon, then fill neighboring pixels until the polygon boundary is reached.
• Boundary Fill Approach:
  • Draw polygon boundary.
  • Determine an interior point.
  • Starting at the interior point, do
    • If the point is not the boundary color or the fill color
      • Set this pixel to the fill color.
      • Propagate to the pixel’s neighbors and continue.
Propagating to Neighbors

- Most frequently used approaches:
  - 4-connected area
  - 8-connected area
Fill Problems

- Fill algorithms have potential problems
- E.g., 4-connected area fill:

Starting point

Fill complete
Fill Problems

- Similarly, 8-connected can “leak” over to another polygon

![Diagram showing starting point and fill complete](image)

- Another problem: the algorithm is highly recursive
  - Can use a stack of spans to reduce amount of recursion
Pattern Filling

- Often we want to fill a region with a pattern, not just a color
- Define an $n$ by $m$ pixmap (or bitmap) that we wish to replicate across the region

5x4 pixmap

Object to be patterned

Final patterned object
Pattern Filling

• How do we determine which color to color a point in the object?
• Use the MOD function to tile the pattern across the polygon
• For point \((x, y)\)
  • Use the pattern color located at \((x \mod m, y \mod n)\)
Triangles

• Always convex: No matter how you rotate a triangle, it only has one span per scan line.

• Any polygon can be decomposed into triangles.

Convex

Concave
Triangles (cont.)

- Rasterization algorithms can take advantage of triangle properties.
- Graphics hardware is optimized for triangles.
- Because triangle drawing is so fast, many systems will subdivide polygons into triangles prior to scan conversion.
Z-Buffering

- Visible Surface Determination Algorithm:
  - Determine which object is visible at each pixel.
  - Order of polygons is not critical.
  - Works for dynamic scenes.

- Basic idea:
  - Rasterize (scan-convert) each polygon, one at a time
  - Keep track of a $z$ value at each pixel
    - Interpolate $z$ value of vertices during rasterization.
  - Replace pixel with new color if $z$ value is greater.
    (i.e., if object is closer to eye)
Example

Goal is to figure out which polygon to draw based on which is in front of what.
Z-buffering

- Need to maintain:
  - Frame buffer
    - contains color values for each pixel
  - Z-buffer
    - contains the current value of $z$ for each pixel
- The two buffers have the same width and height.
- No object/object intersections.
- No sorting of objects required.
- Additional memory is required for the z-buffer.
  - In the early days, this was a problem.
allocate z-buffer;

// Initialize color and depth

for each pixel (x,y)
   color [x][y] = backgroundColor;
   zBuffer[x][y] = farPlane;

// Draw polygons (in any order).

for each polygon p
   for each pixel (x,y) in p  // Rasterize polygon.
      p_z = p’s z-value at (x,y);  // Interpolate polygon’s z-value
      if p_z > zBuffer[x][y]  // If new depth is closer:
         color[x][y] = newColor;  // Write new color & new depth
         zBuffer[x][y] = p_z;
Scan convert the following two polygons. The number inside the pixel represents its z-value.

Does order matter?
Z-Buffering: Example

\[
\begin{array}{cccc}
-\infty & -\infty & -\infty & -\infty \\
-\infty & -\infty & -\infty & -\infty \\
-\infty & -\infty & -\infty & -\infty \\
-\infty & -\infty & -\infty & -\infty \\
\end{array}
\]

\[
\begin{array}{cccc}
-1 \\
-2 \\
-3 \\
-4 \\
\end{array}
\]

\[
\begin{array}{cccc}
-3 & -4 & -5 \\
-4 & -5 & -6 & -7 \\
\end{array}
\]

\[
\begin{array}{cccc}
-1 & -2 & -3 & -\infty \\
-2 & -3 & -\infty & -\infty \\
-3 & -4 & -\infty & -\infty \\
-4 & -5 & -\infty & -\infty \\
\end{array}
\]

\[
\begin{array}{cccc}
-1 \\
-3 & -2 \\
-5 & -4 & -3 \\
-7 & -6 & -5 & -4 \\
\end{array}
\]

\[
\begin{array}{cccc}
-1 & -\infty & -\infty & -1 \\
-2 & -3 & -3 & -2 \\
-3 & -4 & -3 & -3 \\
-4 & -5 & -5 & -4 \\
\end{array}
\]

\[
\begin{array}{cccc}
-1 \\
-2 \\
-3 \\
-4 \\
\end{array}
\]

\[
\begin{array}{cccc}
-3 & -2 \\
-5 & -4 & -3 \\
-7 & -6 & -5 & -4 \\
\end{array}
\]

\[
\begin{array}{cccc}
-1 & -\infty & -\infty & -1 \\
-2 & -3 & -3 & -2 \\
-3 & -4 & -3 & -3 \\
-4 & -5 & -5 & -4 \\
\end{array}
\]

\[
\begin{array}{cccc}
-1 \\
-2 \\
-3 \\
-4 \\
\end{array}
\]

\[
\begin{array}{cccc}
-3 & -2 \\
-5 & -4 & -3 \\
-7 & -6 & -5 & -4 \\
\end{array}
\]

\[
\begin{array}{cccc}
-1 & -\infty & -\infty & -1 \\
-2 & -3 & -3 & -2 \\
-3 & -4 & -3 & -3 \\
-4 & -5 & -5 & -4 \\
\end{array}
\]
Z-Buffering: Computing Z

• How do you compute the $z$ value at a given pixel?
  • Interpolate between vertices

\[
\begin{align*}
    z_a &= z_1 + (z_2 - z_1) \frac{y_1 - y_s}{y_1 - y_2} \\
    z_b &= z_1 + (z_3 - z_1) \frac{y_1 - y_s}{y_1 - y_3} \\
    z_s &= z_b + (z_a - z_b) \frac{x_b - x_s}{x_b - x_a}
\end{align*}
\]

How do we compute $x_a$ and $x_b$?
Z-buffer Implementation

• Modify the 2D polygon algorithm slightly.
  • When projected onto the screen 3D polygons look like 2D polygons (don’t sweat the projection, yet).

• Compute Z values to figure out what’s in front.

• Modifications to polygon scan converter
  • Need to keep track of z value in GET and AET.
  • Before drawing a pixel, compare the current z value to the z-buffer.
  • If you color the pixel, update the z-buffer.
  • For optimization:
    • Maintain a horizontal z-increment for each new pixel.
    • Maintain a vertical z-increment for each new scanline.
GET Entries Updated for Z-buffering

• GET Entries before Z-buffering

\[
\begin{array}{|c|c|c|}
\hline
y_{\text{max}} & x @ y_{\text{min}} & 1/m \\
\hline
\end{array}
\]

• With Z-buffering:

\[
\begin{array}{|c|c|c|c|}
\hline
y_{\text{max}} & x @ y_{\text{min}} & 1/m & z @ y_{\text{min}} & \text{vertZ} \\
\hline
\end{array}
\]

Vertical Z Increment
Computing the Vertical Z Increment

- This value is the increment in $z$ each time we move to a new scan line

$$vertZ = \frac{z_1 - z_0}{y_1 - y_0}$$
Horizontal Z Increment

- We can also compute a \( \text{horizontalZ} \) increment for the x direction.
- As we move horizontally between pixels, we increment z by \( \text{horizontalZ} \).
- Given the current z values of the two edges of a span, \( \text{horizontalZ} \) is given by

\[
\text{horizontalZ} = \frac{z_b - z_a}{x_b - x_a}
\]
Horizontal Increment of a Span

\[ p_a = (x_a, y_a, z_a) \]

\[ p_b = (x_b, y_b, z_b) \]
AET Entries Updated for Z-buffering

- AET Entries before Z-buffering:

<table>
<thead>
<tr>
<th>y_max</th>
<th>x @ current y</th>
<th>1/m</th>
</tr>
</thead>
</table>

- With Z-buffering:

<table>
<thead>
<tr>
<th>y_max</th>
<th>x @ current y</th>
<th>1/m</th>
<th>vertZ</th>
<th>z @ current x,y</th>
</tr>
</thead>
</table>

- Note: *horizontalZ* doesn’t need to be stored in the AET – just computed each iteration.
Z-Buffering: Recap

- Create a z-buffer which is the same size as the frame-buffer.
- Initialize frame-buffer to background.
- Initialize z-buffer to far plane.
- Scan convert polygons one at a time, just as before.
- Maintain z-increment values in the edge tables.
- At each pixel, compare the current z-value to the value stored in the z-buffer at the pixel location.
  - If the current z-value is greater
    - Color the pixel the color for this point in the polygon.
    - Update the z-buffer.
Z-Buffering: Summary

• Advantages:
  • Easy to implement
  • Fast with hardware support \(\Rightarrow\) Fast depth buffer memory
  • On most hardware
  • No sorting of objects
  • Shadows are easy (later)

• Disadvantages:
  • Extra memory required for z-buffer:
    • Integer depth values
    • Scan-line algorithm
  • Prone to aliasing
    • Super-sampling