COMP175: Computer Graphics

Lecture 5
Rasterization:
Raster-Based Fill Algorithms
Review

Region filling: The process of “coloring in” a region of an image

Requirements:

1. A digital representation of a closed shape
2. A test for determining if a point is inside or outside of the shape
3. A rule or procedure for determining the colors of each point inside the shape
Review

Seed Fill Algorithms
- Start with an interior seed point and grow
- Pixel-based descriptions
- Boundary fill
- Flood fill

Raster-Based Filling
- Fill the interior one raster scan line at a time
- Geometric descriptions
- Even-odd parity rule
- Non-zero winding number rule
Raster-based filling of polygons

Polygon: A sequence of vertices connected by edges
Assume that vertices have been rasterized
Raster-based filling of polygons

For each scan line
  • Determine points where the scan line intersects the polygon
Raster-based filling of polygons

For each scan line

- Determine points where the scan line intersects the polygon
- Set pixels between intersection points (using a fill rule)
  - Even-odd parity rule: set pixels between pairs of intersections
  - Non-zero winding rule: set pixels according to the winding number
Efficient raster-based fill algorithms

We will consider three implementations:

- X-intersection array algorithm
- Edge list algorithm
- Pineda’s algorithm
X-intersection Array Algorithm
X-intersection array algorithm

Basic approach:

1. Create a 2D array of x-intersections.

2. Compute the x-intersections between each edge and the scan lines of the array.

3. For each scan line, sort the x-intersections left to right.

4. For each scan line, render spans of pixels between pairs of x-intersections.
X-intersection array algorithm

A two pass approach

First consider each edge of the polygon (object order)
  • Determine x-intersections

Then consider each scan line (image order)
  • Fill interior spans
1. Create a 2D array of x-intersections

MxN array: Large enough to hold all possible x-intersections for each scan line intersecting the polygon

- M = the number of scan lines
- N = the maximum number of x-intersections for any scan line
2. Insert x-intersections into the array

For each edge

a. Determine the x-intersections between the edge and each scan line
b. Place the x-intersection into the edge crossings array

![Diagram of intersections and scan lines]

<table>
<thead>
<tr>
<th>Scan line</th>
<th>( x_0 )</th>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( \ldots )</th>
<th>( x_{N-1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( j_0+1 )</td>
<td>( x_{01} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( j_0+2 )</td>
<td>( x_{02} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( j_0+3 )</td>
<td>( x_{03} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( j_0+M-1 )</td>
<td>( x_{04} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Intersections for edge \( e_0 \)

Add x-intersections for edge \( e_0 \)
2. Insert x-intersections into the array

For each edge

a. Determine the x-intersections between the edge and each scan line

b. Place the x-intersection into the edge crossings array
2. Insert x-intersections into the array

For each edge
   a. Determine the x-intersections between the edge and each scan line
   b. Place the x-intersection into the edge crossings array
2. Insert x-intersections into the array

For each edge
   a. Determine the x-intersections between the edge and each scan line
   b. Place the x-intersection into the edge crossings array
3. Sort the x-intersections

For each row, sort the x-intersections from left to right
4. Render interior spans

Using even-odd parity rule, fill pixels in each row between pairs of intersection points

First two spans from x-intersection array

Final x-intersection array
Exercise

Fill this polygon using the non-zero winding rule

- For each edge, compute the \( x \)-intersections
- Also store Winding Number Increment (WNI = +1 or -1)
Exercise

Fill this polygon using the non-zero winding rule
- For each edge, compute the x-intersections
- Also store Winding Number Increment (WNI = +1 or -1)

Scan line | x | WNI | x | WNI | x | WNI
---|---|---|---|---|---|---
7 | $x_{07}$ | -1 | | $x_{37}$ | +1 | 
6 | $x_{16}$ | -1 | | $x_{36}$ | +1 | 
5 | $x_{15}$ | -1 | | $x_{35}$ | +1 | 
4 | $x_{14}$ | -1 | | $x_{34}$ | +1 | 
3 | $x_{13}$ | -1 | | $x_{33}$ | +1 | 
2 | $x_{12}$ | -1 | | $x_{22}$ | +1 | 
1 | $x_{11}$ | -1 | | $x_{21}$ | +1 | 
0 | $x_{10}$ | -1 | | $x_{20}$ | +1 | 

Final x-intersection array
Exercise

Sort edge intersections from left to right
Exercise

For each scan line, fill pixels according to the winding number

![Diagram showing winding numbers and scan lines]

<table>
<thead>
<tr>
<th>Scan line</th>
<th>x</th>
<th>WNI</th>
<th>x</th>
<th>WNI</th>
<th>x</th>
<th>WNI</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>x_{07}</td>
<td>-1</td>
<td>x_{37}</td>
<td>+1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>x_{16}</td>
<td>-1</td>
<td>x_{36}</td>
<td>+1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>x_{15}</td>
<td>-1</td>
<td>x_{35}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>x_{34}</td>
<td>+1</td>
<td>x_{14}</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>x_{33}</td>
<td>+1</td>
<td>x_{13}</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>x_{22}</td>
<td>+1</td>
<td>x_{12}</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>x_{21}</td>
<td>+1</td>
<td>x_{11}</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>x_{20}</td>
<td>+1</td>
<td>x_{10}</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Final x-intersection array
Memory considerations

A large x-intersection array is needed for
- General or concave polygons (M could be large)
- High resolution images (N large)

Convex polygon
- A maximum of two x-intersections per scan line
  - M = 2
  - Sorting is easy

Concave polygon
- Subdivide to convex polygons
To compute $x$-intersections efficiently, use DDA:

1. Sample an edge once for each row it crosses.
Digital Differential Analyzer (DDA)

To compute $x$-intersections efficiently, use DDA:
1. Sample an edge once for each row it crosses.
2. Order vertices from bottom to top.
Digital Differential Analyzer (DDA)

To compute $x$-intersections efficiently, use DDA:
1. Sample an edge once for each row it crosses.
2. Order vertices from bottom to top.
3. Determine the change in $x$-intersection between rows.

\[ mInv = \frac{dx}{dy} = \frac{x_1 - x_0}{y_1 - y_0} \]
To compute $x$-intersections efficiently, use DDA:

1. Sample an edge once for each row it crosses.
2. Order vertices from bottom to top.
3. Determine the change in $x$-intersection between rows.
4. Determine the first intersection point $(x, j_0)$

$$m_{\text{Inv}} = \frac{dx}{dy} = \frac{x_1 - x_0}{y_1 - y_0}$$

$$j_0 = \begin{cases} y_0 & \text{if } y_0 \text{ is an integer} \\ (\text{int})y_0 + 1 & \text{otherwise} \end{cases}$$

$$x = x_0 + m_{\text{Inv}} \times (j_0 - y_0)$$
Digital Differential Analyzer (DDA)

To compute $x$-intersections efficiently, use DDA:

\[
\begin{align*}
\text{if } (y_0 \text{ is an integer}) \\
& \quad j = y_0 \\
\text{else} \\
& \quad j = \text{int}(y_0) + 1 \\
\text{end if} \\
\end{align*}
\]

\[
\begin{align*}
m\text{Inv} &= \frac{(x_1 - x_0)}{(y_1 - y_0)} \\
x &= x_0 + m\text{Inv} \times (j - y_0) \\
\end{align*}
\]

\[
\begin{align*}
\text{while } (j \leq y_1) \{ \\
& \quad x\text{Intercept} = x \\
& \quad j = j + 1 \\
& \quad x = x + m\text{Inv} \\
& \quad \}
\end{align*}
\]
Limitations of x-intersection array

A two-pass approach

1. Determine and sort the x-intersections (process each edge)
2. Render the spans (process each scan line)

Can we do it in one pass?

• For each scan line, determine the x-intersections in left to right order and fill spans as we go -> Edge List Algorithm
Edge List Algorithm
Edge list algorithm

Basic approach:

1. Create a global list of all polygon edges.
   • Pre-process edges to facilitate incremental updates of x-intersections between scan lines.
   • Order edges from bottom-left to top-right.

2. Create an active edge list.
   • Keeps track of edges that intersect the current scan line.
   • Order active edges left to right.

3. Process each scan line.
   a. Determine x-intersections of active edges in left to right order.
   b. Fill spans between appropriate x-intersections using a fill rule.
   c. Update active edge list.
1. Create a global list of polygon edges

Order edges from bottom-left to top-right.

<table>
<thead>
<tr>
<th>Edge</th>
<th>Edge data (dx/dy, j₀, x, y₁)</th>
</tr>
</thead>
<tbody>
<tr>
<td>e₀</td>
<td></td>
</tr>
<tr>
<td>e₁</td>
<td></td>
</tr>
<tr>
<td>e₂</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
2. Create an active edge list

List edges that cross the current scan line. Order active edges from left to right.

Active Edge List

<table>
<thead>
<tr>
<th>Edge</th>
<th>Index into the global edge list</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_0$</td>
<td>4 (index to edge $e_4$)</td>
</tr>
<tr>
<td>$a_1$</td>
<td>5</td>
</tr>
<tr>
<td>$a_2$</td>
<td>6</td>
</tr>
<tr>
<td>$a_3$</td>
<td>3</td>
</tr>
</tbody>
</table>
3. Process each scan line

a. Determine $x$-intersections of active edges in left to right order.

b. Fill spans between appropriate $x$-intersections using a fill rule.

c. Update active edge list and move to the next scan line.
An example

Create a global edge list:

General polygon
An example

Create a global edge list:

- Pre-process each edge for fast intersection computation
  - Pre-compute the data needed:
    \[ m_{\text{Inv}} = \frac{dx}{dy}, j_0, x(j_0), y_1, \] and the edge’s winding number increment, if using the non-zero winding rule
- Store pre-processed data in the global edge list

<table>
<thead>
<tr>
<th>Edge</th>
<th>Edge data</th>
</tr>
</thead>
<tbody>
<tr>
<td>(e_0)</td>
<td>(m_{\text{Inv}}, j_{00}, x(j_{00}), y_{10}, \ldots)</td>
</tr>
<tr>
<td>(e_1)</td>
<td>(m_{\text{Inv}}, j_{01}, x(j_{01}), y_{11}, \ldots)</td>
</tr>
<tr>
<td>(e_2)</td>
<td>(m_{\text{Inv}}, j_{02}, x(j_{02}), y_{12}, \ldots)</td>
</tr>
<tr>
<td>(e_3)</td>
<td>(m_{\text{Inv}}, j_{03}, x(j_{03}), y_{13}, \ldots)</td>
</tr>
<tr>
<td>(e_4)</td>
<td>(m_{\text{Inv}}, j_{04}, x(j_{04}), y_{14}, \ldots)</td>
</tr>
<tr>
<td>(e_5)</td>
<td>(m_{\text{Inv}}, j_{05}, x(j_{05}), y_{15}, \ldots)</td>
</tr>
<tr>
<td>(e_6)</td>
<td>(m_{\text{Inv}}, j_{06}, x(j_{06}), y_{16}, \ldots)</td>
</tr>
<tr>
<td>(e_7)</td>
<td>(m_{\text{Inv}}, j_{07}, x(j_{07}), y_{17}, \ldots)</td>
</tr>
</tbody>
</table>

The global edge list has one entry for each edge. This list is initially unsorted.
Create a global edge list:

- Pre-process each edge for fast intersection computation
  - Pre-compute the data needed: \( m_{\text{Inv}} = \frac{dx}{dy}, j_0, x(j_0), y_1, \) and the edge’s winding number increment, if using the non-zero winding rule
- Store pre-processed data in the global edge list
- Sort the global edge list according to each edge’s bottom vertex

<table>
<thead>
<tr>
<th>Edge</th>
<th>Edge data</th>
</tr>
</thead>
<tbody>
<tr>
<td>( e_0 )</td>
<td>( m_{\text{Inv}}<em>0, j</em>{00}, x(j_{00}), y_{10}, \ldots )</td>
</tr>
<tr>
<td>( e_7 )</td>
<td>( m_{\text{Inv}}<em>7, j</em>{07}, x(j_{07}), y_{17}, \ldots )</td>
</tr>
<tr>
<td>( e_6 )</td>
<td>( m_{\text{Inv}}<em>6, j</em>{06}, x(j_{06}), y_{16}, \ldots )</td>
</tr>
<tr>
<td>( e_2 )</td>
<td>( m_{\text{Inv}}<em>2, j</em>{02}, x(j_{02}), y_{12}, \ldots )</td>
</tr>
<tr>
<td>( e_3 )</td>
<td>( m_{\text{Inv}}<em>4, j</em>{04}, x(j_{04}), y_{14}, \ldots )</td>
</tr>
<tr>
<td>( e_1 )</td>
<td>( m_{\text{Inv}}<em>1, j</em>{01}, x(j_{01}), y_{11}, \ldots )</td>
</tr>
<tr>
<td>( e_4 )</td>
<td>( m_{\text{Inv}}<em>3, j</em>{03}, x(j_{03}), y_{13}, \ldots )</td>
</tr>
<tr>
<td>( e_5 )</td>
<td>( m_{\text{Inv}}<em>5, j</em>{05}, x(j_{05}), y_{15}, \ldots )</td>
</tr>
</tbody>
</table>
An example

Keeps track of which edges are intersected by the current scan line

- Initialize the active edge list with edges intersected by the lowest scan line
- Active edges are sorted from left to right
**An example**

**Process each scan line**
- Determine the x-intersections for each active edge using DDA
- Set pixels in the spans between pairs of x-intersections using the fill rule (even-odd parity rule or non-zero winding rule)

![General polygon](image)

<table>
<thead>
<tr>
<th>Edge</th>
<th>Edge data</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_0$</td>
<td>$\text{minv}<em>0, j</em>{00}, x_{(i_0)0}, y_{10}, \ldots$</td>
</tr>
<tr>
<td>$e_7$</td>
<td>$\text{minv}<em>7, j</em>{07}, x_{(i_0)7}, y_{17}, \ldots$</td>
</tr>
<tr>
<td>$e_6$</td>
<td>$\text{minv}<em>6, j</em>{06}, x_{(i_0)6}, y_{16}, \ldots$</td>
</tr>
<tr>
<td>$e_2$</td>
<td>$\text{minv}<em>2, j</em>{02}, x_{(i_0)2}, y_{12}, \ldots$</td>
</tr>
<tr>
<td>$e_3$</td>
<td>$\text{minv}<em>4, j</em>{04}, x_{(i_0)4}, y_{14}, \ldots$</td>
</tr>
<tr>
<td>$e_1$</td>
<td>$\text{minv}<em>1, j</em>{01}, x_{(i_0)1}, y_{11}, \ldots$</td>
</tr>
<tr>
<td>$e_4$</td>
<td>$\text{minv}<em>3, j</em>{03}, x_{(i_0)3}, y_{13}, \ldots$</td>
</tr>
<tr>
<td>$e_5$</td>
<td>$\text{minv}<em>5, j</em>{05}, x_{(i_0)5}, y_{15}, \ldots$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Edge</th>
<th>Edge in global list</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_0$</td>
<td>$e_0$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$e_7$</td>
</tr>
</tbody>
</table>

**Active edge list for scan line $j$**

**Sorted global edge list**

1. **General polygon**
2. **Sorted global edge list**
3. **Active edge list for scan line $j$**
An example

Update the active edge list

- Remove edges whose top vertex is on or below the next scan line
- Add edges from the global edge list whose bottom vertex lies between this scan line and the next scan line or on the next scan line
- Re-sort the edges in the active edge list from left to right
An example

Update the active edge list

- Remove edges whose top vertex is on or below the next scan line
- Add edges from the global edge list whose bottom vertex lies between this scan line and the next scan line or on the next scan line
- Re-sort the edges in the active edge list from left to right
Horizontal edges

If a horizontal edge lies between scan lines
• It does not intersect any scan line

If it lies on a scan line
• The even odd parity and the winding number will not change along the edge
• The vertices of edges connected to the horizontal edge will determine the parity and the winding number

Horizontal edges can be safely ignored
Shared vertices

If the scan line intersects a shared vertex
- The edge crossing could be counted more than once
- Yields incorrect, inconsistent even-odd parity and winding numbers

Scan line pierces the outline
=> Should count one crossing

Scan line grazes the outline
=> Should count 0 or 2 crossings
Shared vertices

One approach:

Ensure that the scan line does not intersect a vertex
Preprocess: Add a small y-offset to a vertex if it falls on a scan line
Shared vertices

Another approach:

Use a convention to include/exclude vertices
- Include the bottom vertex of each edge if it falls on a scan line
- Do not include the top vertex of each edge if it falls on a scan line
Shared vertices

Another approach:

Use a convention to include/exclude vertices

- Include the bottom vertex of each edge if it falls on a scan line
- Do not include the top vertex of each edge if it falls on a scan line

Counts 1 crossing when the scan line pierces the polygon at a shared vertex.
Shared vertices

Another approach:

Use a convention to include/exclude vertices
  - Include the bottom vertex of each edge if it falls on a scan line
  - Do not include the top vertex of each edge if it falls on a scan line

Counts 0 crossings when the scan line grazes the polygon at a shared top vertex.
Shared vertices

Another approach:

Use a convention to include/exclude vertices

- Include the bottom vertex of each edge if it falls on a scan line
- Do not include the top vertex of each edge if it falls on a scan line

Counts 2 crossings when the scan line grazes the polygon at a shared bottom vertex.
Sliver polygons

So thin that some scan lines have only one pixel or no pixels

No simple solution
Avoid long thin triangles
Want triangles with a good aspect ratio

Example of an aliasing problem
(more on these next week)
Implementing edge lists

Recall the basic algorithm:

1. Create a global edge list.

2. Create an active edge list.

3. Process each scan line.
   a. Determine x-intersections of active edges in left to right order.
   b. Fill spans between appropriate x-intersections using a fill rule.
   c. Update active edge list.
What information is required?

To create the global edge list:

- Vertices ordered from bottom to top (ignore horizontal edges)
- Data required to compute intersections between edges and scan lines
- Data for the fill rule

![Diagram of edges and vertices](image.png)

<table>
<thead>
<tr>
<th>Edge</th>
<th>(x₀, y₀)</th>
<th>(x₁, y₁)</th>
</tr>
</thead>
<tbody>
<tr>
<td>e₀</td>
<td>(0, 0.5)</td>
<td>(0, 4.5)</td>
</tr>
<tr>
<td>e₁</td>
<td>(2, 2)</td>
<td>(0, 4.5)</td>
</tr>
<tr>
<td>e₂</td>
<td>(2, 2)</td>
<td>(0, 5.5)</td>
</tr>
<tr>
<td>e₃</td>
<td>(4, 0.5)</td>
<td>(4, 5.5)</td>
</tr>
<tr>
<td>e₄</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

Vertices are listed bottom to top
What information is required?

To create the active edge list:

- Which edges are active for a given scan line
- How to update the active edge list for the next scan line

![Active Edges](image)
What information is required?

To determine x-intersections for current scan line:

• Data for determining the x-intersection for each scan line
  • e.g., if using DDA, need previous x-intersection and dx/dy
What information is required?

To fill spans between x-intersections:

• X-intersections for each active edge, ordered left to right
• Even-odd parity bit or winding number
• Edge direction if using non-zero winding rule
  • (i.e., the winding number increment, +1 or -1)
What information is required?

To update the active edge list:

- Test to determine when to remove an edge
  - Compare next scan line with max height of each edge
  - For a closed polygon, always add a new edge when an old edge is removed (until the last edge has been processed)
  - New edges should be initialized in the pre-processing step so that it doesn’t have to be done here

- Data for updating x-intersections of each active edge
Data structures

EdgeData: For each edge,

• \( j_0 \) = bottom-most scan line intersected by the edge

• \( j_1 \) = top-most scan line intersected by the edge

• \( x \) = the value of the \( x \)-intersection with the current scan line, initialized to \( x(j_0) \)

• \( dx \) = the change in the value of the \( x \)-intersection for the next scan line

• \( \text{dir} \) = the direction of the edge (1 if the original edge is upward, -1 if the original edge is downward)
Data structures

EdgeData: For each edge,

- \( j_0 \) = bottom-most scan line intersected by the edge
- \( j_1 \) = top-most scan line intersected by the edge
- \( x \) = the value of the x-intersection with the current scan line, initialized to \( x(j_0) \)
- \( dx \) = the change in the value of the x-intersection for the next scan line
- \( dir \) = the direction of the edge (1 if the original edge is upward, -1 if the original edge is downward)

Compute the line data
if \( y_0 \) is an integer \( j_0 = y_0 \)
else \( j_0 = (\text{int}) y_0 + 1 \)

if \( y_1 \) is an integer \( j_1 = y_1 - 1 \)
else \( j_1 = (\text{int}) y_1 \)

\( dx = (x_1 - x_0) / (y_1 - y_0) \)
\( x = x_0 + dx \times (j_0 - y_0) \)
\( dir = 1 \)
Data structures

typedef struct {
    int j0;   //Bottom-most scan line intersecting edge
    int j1;   //Top-most scan line intersecting edge
    float x;  //x-intersection with current scan line
    float dx; //Displacement of x-intersect for next scan line
    float dir; //1 if edge is upward, -1 if edge is downward
} EdgeData;
Global edge list

Allocate an array big enough to hold all the edge data

```c
EdgeData *global = (edgeData *) malloc(nEdges * sizeof(EdgeData));
```

<table>
<thead>
<tr>
<th>Edge</th>
<th>Data ((j_0, j_1, x, dx, dir))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(e_0)</td>
<td></td>
</tr>
<tr>
<td>(e_1)</td>
<td></td>
</tr>
<tr>
<td>(e_2)</td>
<td></td>
</tr>
<tr>
<td>(e_3)</td>
<td></td>
</tr>
<tr>
<td>(e_4)</td>
<td></td>
</tr>
</tbody>
</table>
Global edge list

Compute the data for each edge and insert it in the global edge list

<table>
<thead>
<tr>
<th>Edge</th>
<th>Data $\langle j_0, j_1, x, dx, dir \rangle$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_0$</td>
<td>1, 4, 0, 0, 1</td>
</tr>
<tr>
<td>$e_1$</td>
<td>3, 4, 1.2, -0.8, -1</td>
</tr>
<tr>
<td>$e_2$</td>
<td>3, 5, 2.71, 0.71, 1</td>
</tr>
<tr>
<td>$e_3$</td>
<td>1, 5, 4.5, 0, -1</td>
</tr>
<tr>
<td>$e_4$</td>
<td>1, 0, 0, 0, 0</td>
</tr>
</tbody>
</table>
Global edge list

Compute the data for each edge and insert it in the global edge list
Note that we can ignore horizontal edges ($j_1 = j_0$)
Global edge list

Order the edges in the global edge list bottom-to-top, left-to-right
- Use a standard sorting method, e.g., insertion sort

<table>
<thead>
<tr>
<th>Edge</th>
<th>Data (j₀, j₁, x, dx, dir)</th>
</tr>
</thead>
<tbody>
<tr>
<td>e₀</td>
<td>1, 4, 0, 0, 1</td>
</tr>
<tr>
<td>e₁</td>
<td>3, 4, 1.2, -0.8, -1</td>
</tr>
<tr>
<td>e₂</td>
<td>3, 5, 2.71, 0.71, 1</td>
</tr>
<tr>
<td>e₃</td>
<td>1, 5, 4.5, 0, -1</td>
</tr>
</tbody>
</table>
Sorting the global edge list

Start at the head of the list
Search remaining elements in the list for smallest \( j_0 \) (if two edges have equal \( j_0 \), choose the one with the smallest \( x \)-intersection)
Swap the current edge with the one with the smallest \( j_0 \)
Start at the next edge in the list, search the remaining elements and swap, continuing until the last edge of the list is reached

<table>
<thead>
<tr>
<th>Edge</th>
<th>Data ((j_0, j_1, x, dx, \text{dir}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( e_0 )</td>
<td>( 1, 4, 0, 0, 1 )</td>
</tr>
<tr>
<td>( e_1 )</td>
<td>( 3, 4, 1.2, -0.8, -1 )</td>
</tr>
<tr>
<td>( e_2 )</td>
<td>( 3, 5, 2.71, 0.71, 1 )</td>
</tr>
<tr>
<td>( e_3 )</td>
<td>( 1, 5, 4.5, 0, -1 )</td>
</tr>
</tbody>
</table>

Global edge list
Sorting the global edge list

Start at the head of the list
Search remaining elements in the list for smallest $j_0$ (if two edges have equal $j_0$, choose the one with the smallest x-intersection)
Swap the current edge with the one with the smallest $j_0$
Start at the next edge in the list, search the remaining elements and swap, continuing until the last edge of the list is reached

e_0$ is the lowest, leftmost edge: no swap is needed

<table>
<thead>
<tr>
<th>Edge</th>
<th>Data $(j_0, j_1, x, dx, dir)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_0</td>
<td>1, 4, 0, 0, 1</td>
</tr>
<tr>
<td>e_1</td>
<td>3, 4, 1.2, -0.8, -1</td>
</tr>
<tr>
<td>e_2</td>
<td>3, 5, 2.71, 0.71, 1</td>
</tr>
<tr>
<td>e_3</td>
<td>1, 5, 4.5, 0, -1</td>
</tr>
</tbody>
</table>

Global edge list
Sorting the global edge list

Start at the head of the list
Search remaining elements in the list for smallest $j_0$ (if two edges have equal $j_0$, choose the one with the smallest $x$-intersection)
Swap the current edge with the one with the smallest $j_0$
Start at the next edge in the list, search the remaining elements and swap, continuing until the last edge of the list is reached

<table>
<thead>
<tr>
<th>Edge</th>
<th>Data ($j_0$, $j_1$, $x$, $dx$, dir)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_0$</td>
<td>1, 4, 0, 0, 1</td>
</tr>
<tr>
<td>$e_1$</td>
<td>3, 4, 1.2, -0.8, -1</td>
</tr>
<tr>
<td>$e_2$</td>
<td>3, 5, 2.71, 0.71, 1</td>
</tr>
<tr>
<td>$e_3$</td>
<td>1, 5, 4.5, 0, -1</td>
</tr>
</tbody>
</table>

$e_3$ is the next lowest, leftmost edge: swap with current edge
Sorting the global edge list

Start at the head of the list
Search remaining elements in the list for smallest $j_0$ (if two edges have equal $j_0$, choose the one with the smallest $x$-intersection)
Swap the current edge with the one with the smallest $j_0$
Start at the next edge in the list, search the remaining elements and swap, continuing until the last edge of the list is reached

<table>
<thead>
<tr>
<th>Edge</th>
<th>Data ($j_0$, $j_1$, $x$, $dx$, dir)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_0$</td>
<td>1, 4, 0, 0, 1</td>
</tr>
<tr>
<td>$e_3$</td>
<td>1, 5, 4.5, 0, -1</td>
</tr>
<tr>
<td>$e_2$</td>
<td>3, 5, 2.71, 0.71, 1</td>
</tr>
<tr>
<td>$e_1$</td>
<td>3, 4, 1.2, -0.8, -1</td>
</tr>
</tbody>
</table>

$e_3$ is the next lowest, leftmost edge: swap with current edge

Global edge list
Sorting the global edge list

Start at the head of the list
Search remaining elements in the list for smallest \( j_0 \) (if two edges have equal \( j_0 \), choose the one with the smallest \( x \)-intersection)
Swap the current edge with the one with the smallest \( j_0 \)
Start at the next edge in the list, search the remaining elements and swap, continuing until the last edge of the list is reached

<table>
<thead>
<tr>
<th>Edge</th>
<th>Data ((j_0, j_1, x, dx, \text{dir}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( e_0 )</td>
<td>1, 4, 0, 0, 1</td>
</tr>
<tr>
<td>( e_3 )</td>
<td>1, 5, 4.5, 0, -1</td>
</tr>
<tr>
<td>( e_2 )</td>
<td>3, 5, 2.71, 0.71, 1</td>
</tr>
<tr>
<td>( e_1 )</td>
<td>3, 4, 1.2, -0.8, -1</td>
</tr>
</tbody>
</table>

\( e_1 \) is the next lowest, leftmost edge: swap with current edge

Global edge list
Sorting the global edge list

Start at the head of the list
Search remaining elements in the list for smallest $j_0$ (if two edges have equal $j_0$, choose the one with the smallest $x$-intersection)
Swap the current edge with the one with the smallest $j_0$
Start at the next edge in the list, search the remaining elements and swap, continuing until the last edge of the list is reached

<table>
<thead>
<tr>
<th>Edge</th>
<th>Data ($j_0$, $j_1$, $x$, $dx$, $dir$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_0$</td>
<td>1, 4, 0, 0, 1</td>
</tr>
<tr>
<td>$e_3$</td>
<td>1, 5, 4.5, 0, -1</td>
</tr>
<tr>
<td>$e_1$</td>
<td>3, 4, 1.2, -0.8, -1</td>
</tr>
<tr>
<td>$e_2$</td>
<td>3, 5, 2.71, 0.71, 1</td>
</tr>
</tbody>
</table>

$e_1$ is the next lowest, leftmost edge: swap with current edge
## Sorting the global edge list

Start at the head of the list
Search remaining elements in the list for smallest $j_0$ (if two edges have equal $j_0$, choose the one with the smallest $x$-intersection)
Swap the current edge with the one with the smallest $j_0$
Start at the next edge in the list, search the remaining elements and swap, continuing until the last edge of the list is reached

<table>
<thead>
<tr>
<th>Edge</th>
<th>Data ($j_0$, $j_1$, $x$, $dx$, dir)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_0$</td>
<td>1, 4, 0, 0, 1</td>
</tr>
<tr>
<td>$e_3$</td>
<td>1, 5, 4.5, 0, -1</td>
</tr>
<tr>
<td>$e_1$</td>
<td>3, 4, 1.2, -0.8, -1</td>
</tr>
<tr>
<td>$e_2$</td>
<td>3, 5, 2.71, 0.71, 1</td>
</tr>
</tbody>
</table>

Last edge: sort is complete

Global edge list
Global edge list

Now have a sorted global edge list

<table>
<thead>
<tr>
<th>Edge</th>
<th>Data $(j_0, j_1, x, dx, dir)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_0$</td>
<td>1, 4, 0, 0, 1</td>
</tr>
<tr>
<td>$e_3$</td>
<td>1, 5, 4.5, 0, -1</td>
</tr>
<tr>
<td>$e_1$</td>
<td>3, 4, 1.2, -0.8, -1</td>
</tr>
<tr>
<td>$e_2$</td>
<td>3, 5, 2.71, 0.71, 1</td>
</tr>
</tbody>
</table>
Active edge list

Keeps track of which edges in the global edge list are intersected by the current scan line

Use to fill spans between x-intersections in the current scan line

Must update the active list with each new scan line
Active edge list

There are several ways to implement it
Could use linked lists:
  • Keep a linked list of edge indices
  • Each element in the linked list contains an index to the corresponding edge in the global edge array and pointer to the next and previous active edges
  • To update the active edge list, update the pointers

typedef struct {
    int edgeIdx;               // Edge index in the global edge array
    _ActiveEdge *prev;        // Pointer to previous active edge
    _ActiveEdge *next;        // Pointer to next active edge
} ActiveEdge;
Active edge list

There are several ways to implement it
Could use an active edge array

```c
int nActive; // # active edges
int *active = (int) malloc (nEdges * sizeof(int)); // Edge indices
```

- Initialize the active list with indices of edges that intersect the first scan line
- Update the active list with each new scan line
  - Remove edges if \( j_1 \) is below the new scan line
  - Update \( x \) for each edge remaining in the active edge list
  - Add edges whose \( j_0 \) falls between the previous and the new scan line
  - Sort edges in the active edge list from left to right (e.g., insertion sort)
Exercise

A. Fill in the active edge table with the ordered active edges at each scan line

B. Use non-zero winding to determine which ranges of x values are inside the polygon

<table>
<thead>
<tr>
<th>Edge</th>
<th>Data $(j_0, j_1, x, dx, dir)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_0$</td>
<td>$1, 4, 0, 0, 1$</td>
</tr>
<tr>
<td>$e_3$</td>
<td>$1, 5, 4.5, 0, -1$</td>
</tr>
<tr>
<td>$e_1$</td>
<td>$3, 4, 1.2, -0.8, -1$</td>
</tr>
<tr>
<td>$e_2$</td>
<td>$3, 5, 2.71, 0.71, 1$</td>
</tr>
</tbody>
</table>

Global edge list

<table>
<thead>
<tr>
<th>Scan line</th>
<th>Edge index</th>
</tr>
</thead>
<tbody>
<tr>
<td>$j = 1$</td>
<td>0, 3</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Active edge list
Summary

Raster-based filling of polygons

X-intersection array algorithm
  Improve efficiency with DDA

Edge list algorithm
  Special considerations: horizontal edges, shared vertices, slivers
  Implementation with active edge array

Both approaches find limits of interior spans, fill pixels between the limits
Next time

Coloring and texturing polygons

Third polygon filling approach: Pineda’s algorithm
  Uses coherence of square tiles rather than rows of pixels
  Computes color/texture interpolation