Midpoint Ellipse Algorithm

- Use symmetry of ellipse
- Divide the quadrant into two regions
  - the boundary of two regions is the point at which the curve has a slope of -1.
  - Process by taking unit steps in the $x$ direction to the point $P$, then taking unit steps in the $y$ direction
- Apply midpoint algorithm.

![Diagram showing the midpoint ellipse algorithm with points $(-x, y)$, $(x, y)$, $(-x, -y)$, $(x, -y)$ and the line with slope $-1$.]
Midpoint Ellipse Algorithm

\[
\left( \frac{x - x_c}{r_x} \right)^2 + \left( \frac{y - y_c}{r_y} \right)^2 = 1
\]

\[f_{\text{ellipse}}(x, y) = r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2\]

\[
\begin{cases} 
< 0 & \text{inside the ellipse boundary} \\
= 0 & \text{on the ellipse boundary} \\
> 0 & \text{outside the ellipse boundary}
\end{cases}
\]

\[p_{1k} = f_{\text{ellipse}}(x_k + 1, y_k - \frac{1}{2})\]

\[p_{2k} = f_{\text{ellipse}}(x_k + \frac{1}{2}, y_k - 1)\]
Maintaining Geometric Properties

- When line drawing,
  - exclude the last point
- When drawing an enclosed area,
  - display the area using only those pixels that are interior to the object boundaries

**FIGURE 3-36** Line path and corresponding pixel display for grid endpoint coordinates (20, 10) and (30, 18).

**FIGURE 3-37** Conversion of rectangle (a) with vertices at screen coordinates (0, 0), (4, 0), (4, 3), and (0, 3) into display (b) that includes the right and top boundaries and into display (c) that maintains geometric magnitudes.
Inside-Outside Tests

Odd-even Rule (Odd-parity Rule)

- For each pixel, determine if it is inside or outside of a given polygon.

- **Approach**
  - From any point \( P \) being tested, cast a ray to a distant point in an arbitrary direction.
  - If the number of crossings is *odd*, then \( P \) is an *interior* point.
  - If the number of crossings is *even*, then \( P \) is an *exterior* point.
Odd-Even Rule

- Be sure that the line path does not intersect any line-segment endpoints.
Odd-Even Rule

- Edge Crossing Rules
  - an upward edge includes its starting endpoint, and excludes its final endpoint;
  - a downward edge excludes its starting endpoint, and include its final endpoint;
  - horizontal edges are excluded;
Odd-Even Rule

- Very fragile algorithm
  - Ray crosses a vertex
  - Ray is coincident with an edge
- Commonly used in ECAD
- Suitable for H/W
Inside-Outside Tests

Nonzero Winding Number Rule

- A winding number
  - the # of times the boundary of an object “winds” around a particular point \( P \) in the counterclockwise direction
  - *Non-zero* values: *interior* points
  - *Zero* values: *exterior* points
Nonzero Winding Number Rule

- Winding num: initialized to 0
- From any point \( P \) being tested, cast a ray to a distant point in an arbitrary direction
  - +1: edge crossing the line from \textit{right to left}
  - -1: \textit{left to right}
- Use the sign of the cross product of the line and edge vectors
  - Non-zero: \textit{interior}
  - Zero: \textit{exterior}
- Be sure that the line path does not pass through any line-segment endpoints.
How to decide interior

Vertices are numbered: 0 1 2 3 4 5 6 7 8 9
Polygon Tables

- An object
  - described as a set of polygon surface facets
- Geometric data
  - vertex table
  - edge table
  - surface-facet table
- Other parameters
  - Color, transparency, light-reflection properties
Area Filling

How to generate a solid color/patterned polygon area

- Which pixels?
- What value?

Scan-line fill algorithm
Scan-Line Fill Algorithm

- For each scan line
  - (1) Find intersections (the extrema of spans)
    - Use Bresenham's line-scan algorithm
    - Note that in a line drawing algorithm there is no difference between interior and exterior pixels
  - (2) Sort intersections (increasing x order)
  - (3) Fill in between pair of intersections
Scan-Line fill algorithm

Find intersections

- \( x_{k+1} = x_k + \Delta x / \Delta y \)
- example (left edge)
  - \( m = 5/2 \)
  - \( x_{\text{min}} = 3 \)
- the sequence of \( x \) values
  - \( 3, 3+2/5, 3+4/5, 3+5/6=4+1/5 \)

<table>
<thead>
<tr>
<th>y</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
<td>3+ 2/5</td>
<td>3+ 4/5</td>
<td>4+ 1/5</td>
</tr>
<tr>
<td>pixel</td>
<td>(3,1)</td>
<td>(3,2)</td>
<td>(4,3)</td>
<td>(4,4)</td>
</tr>
</tbody>
</table>
A standard convention is to say that a point on a left or bottom edge is inside, and a point on a right or top edge is outside.
Scan-line Fill Algorithm

- Not quite simple
  - A scan line passed through a vertex, it intersects *two polygon edges at that point.*
Scan-line Fill Algorithm

Scan line $y'$
- intersects an even number of edges
- Two pairs of intersection points correctly identify the interior span

Scan line $y$
- intersects an odd number (5) of edges
- Must count the vertex intersections \textit{as only one point}
Scan-line Fill Algorithm

How to distinguish these cases

- **Scan line $y$**
  - Two intersecting edges are on opposite sides of the scan line
  - *Counted as just one boundary intersection point*

- **Scan line $y’$**
  - Two intersecting edges are both above the scan line

- **By tracing around the boundary,**
  - If three endpoint $y$ values of two consecutive edges monotonically increase or decrease
    $\rightarrow$ count the shared vertex as *a single intersection*
  - Otherwise, (a local minimum or maximum)
    $\rightarrow$ *add the two edge intersections* with the scan line
Scan-line Fill Algorithm

Implementation

- To shorten some polygon edges to split those vertices that should be counted as one intersection
- While processing non-horizontal edges in (counter)clockwise
  - Check each edge whether the edge & next edges have either monotonically increasing or decreasing endpoints
  - If so, *shorten the lower edge*

![Scan-line Fill Algorithm Diagram](image)
Scan-line Fill Algorithm

- Coherence properties
  - span coherence - all pixels on a span are set to the same value
  - scan-line coherence - consecutive scan lines are identical
  - edge coherence - edges intersected by scan line $i$ are also intersected by scan line $i+1$
Scan-line Fill Algorithm

- Use edge coherence and the scan-line algorithm
  
  - Sorted Edge Table
    - Contains all the non-horizontal edges.
    - Edges are sorted by their smaller $y$ coordinates.
    - Table entry:
      - maximum $y$ value, $x$-intercept value, inverse slope
    - For each scan line, edges are sorted from left to right

- Active Edge List
  
  - While processing scan lines from bottom to top, product *active edge list* for each scan line.
  - Contains edges which intersect the current scan line.
  - Edges are sorted on their $x$ intersection values.
Scan-line Fill Algorithm
Area Filling (Scan line method)

Scan line 9

Scan line 10
Span Rules

- intersection at integer coordinate
  - leftmost: interior
  - rightmost: exterior

- shared vertices
  - count parity at $y_{min}$ vertices only
  - shorten edges

- horizontal edges
  - do not count vertices

---

A standard convention is to say that a point on a left or bottom edge is inside, and a point on a right or top edge is outside.
## Area Filling (Filling Methods)

- **Pixel Adjacency**

<table>
<thead>
<tr>
<th>8-connected</th>
<th>4-connected</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Boundary-Fill Algorithm**
  - If the boundary is specified in a single color, fill the interior, pixel by pixel, until the boundary color is encountered.
  - Starts from an interior point and paints the interior in a specified color or intensity.
procedure boundaryFill4(x, y: integer; borderColor: color; fillColor: color);
  var c: color;
  begin
    c := readPixel(x, y);
    if (c ≠ borderColor and c ≠ fillColor) then
      begin
        writePixel(x, y, newValue);
        boundaryFill4(x, y - 1, borderColor, fillColor);
        boundaryFill4(x, y + 1, borderColor, fillColor);
        boundaryFill4(x - 1, y, borderColor, fillColor);
        boundaryFill4(x + 1, y, borderColor, fillColor);
      end
  end;

Boundary-Fill Algorithm
There is the following problem with boundary_fill4:

- Involve heavy duty recursion which may consume memory and time

Solve with 8-connected
Boundary-Fill Algo.

Efficiency in space!

- Starting from the initial interior point, first fill in the span on the starting scan line
- Locate and stack starting positions for spans on the adjacent scan lines
  - in down to up order
  - in left to right order
- All upper scan lines processed ⇒ All lower scan lines processed
Flood-Fill Algorithm

- Flood-Fill Algorithm
  - When filling an area that is not defined within a single color boundary...
  - start from an interior point and reassign all pixel values that are currently set to a given interior color with the desired fill color.
procedure flood_fill4(
   x,y: integer  // starting point in region
   oldValue: color      // value that defines interior
   newvalue: color); replacement value
begin
   if readPixel(x,y) = oldValue then
      begin
         writePixel(x,y,newValue);
         flood_fill4(x,y-1,oldValue,newValue);
         flood_fill4(x,y+1,oldValue,newValue);
         flood_fill4(x-1,y,oldValue,newValue);
         flood_fill4(x+1,y,oldValue,newValue);
      end
   end
end;
Patterned Lines

- Patterned line from \( P \) to \( Q \) is not same as patterned line from \( Q \) to \( P \).

- Patterns can be geometric or cosmetic
  - Cosmetic: Texture applied after transformations
  - Geometric: Pattern subject to transformations
Character, Symbols

- Bitmap font (Raster font)
  - Set up a pattern of binary values on a rectangular grid
  - The simplest to define and display
  - But, more storage: each variation (size and format) must be saved
Character, Symbols

Outline font

- Use straight-line and curve sections
- More flexible method
- Can be increased in size without distorting the character shapes
  - By manipulating curve definitions for the character outlines.
  - More time, since they must be scan converted into the frame buffer
- Less storage
## Comparison of Methods

<table>
<thead>
<tr>
<th>Outline font</th>
<th>Bitmap font</th>
</tr>
</thead>
<tbody>
<tr>
<td>easy to rotate</td>
<td>rotate by multiples of 90°</td>
</tr>
<tr>
<td>easy to scale</td>
<td>scale by powers of 2</td>
</tr>
<tr>
<td>variable length storage</td>
<td>fixed length storage</td>
</tr>
<tr>
<td>scan convert lines</td>
<td>scan convert points</td>
</tr>
<tr>
<td>fill if polygons</td>
<td>draw as filled or outline</td>
</tr>
<tr>
<td>may be anti-aliased or smoothed via curve fitting</td>
<td>may be pre-anti-aliased</td>
</tr>
</tbody>
</table>
Line Attributes

Butt cap

Round cap

Projecting square cap

Miter join

Round Join

Bevel join
Aliasing in CG

Which is better?
Digital technology can only *approximate* analog signals through a process known as *sampling*.

Aliasing: the distortion of information due to low-frequency sampling (undersampling).

Choosing an appropriate *sampling rate* depends on data size restraints, need for accuracy, the cost per sample...

Errors caused by aliasing are called *artifacts*. Common aliasing artifacts in computer graphics include jagged profiles, disappearing or improperly rendered fine detail, and disintegrating textures.