Lecture 4
Rasterization:
Region Filling
Drawing 2D shapes

Rasterization of graphics primitives
Midpoint algorithms: useful for drawing outlines

- Line segments
- Circles
- More general curves
Filling 2D shapes

Solid fill

Pattern fill

Texture fill
Region filling

The process of “coloring in” a region of an image

Requirements:

1. A digital representation of the shape
   a. The shape must be closed.
   b. It must have a well defined inside and outside.

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
Describing a region: Bounding pixels

“Inside” is determined by the boundary and a seed point

All points connected to the seed point are inside
Describing a region: Color range

“Inside” is determined by a color or color range

Original image

Pink pixels have been filled yellow
Describing a region: Geometrically

“Inside-ness” is determined by evaluating an inequality

Points satisfying the inequality are inside

\[ x^2 + y^2 < R^2 \]

The inside of a circle of radius R

\[ 0 < x < 1 \]
\[ 0 < y < 1 \]

The inside of a unit square
Describing a region: Geometrically

A set of edge equations and a rule for determining the interior

Inside is determined by testing the rule for each edge

- Example:
  - A list of directed edges:
    - line from (0,0) to (1, 0)
    - line from (1,0) to (1, 1)
    - line from (1,1) to (0, 1)
    - line from (0, 1) to (0,0)
  - Rule for interior points:
    - interior points lie to the right of all of the edges
Pixel-level vs. geometric descriptions

Pixel-level: Describe a region in terms of
  • its bounding pixels (boundary-defined), or
  • all of the pixels that comprise it (interior-defined)

Geometric: Define a region by connected lines and curves
Approaches to filling regions

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

Raster-Based Filling
- Fill the interior one raster scan line at a time
- Geometric descriptions
Seed Fill Algorithms
Seed Fill

1. Select a seed point inside a region
2. Move outwards from the seed point
   a. If pixel is not set, set pixel
   b. Process each neighbor of pixel that is inside the region

1. Select a seed point
2. Move outwards to neighbors.
Stop when the region is filled.
Selecting the seed point

Difficult to place the seed point automatically

What is the inside of this shape?
Seed Fill algorithm

user selects seedPixel
initialize a fillList to contain seedPixel

while (fillList not empty)
{
    pixel ← next pixel from fillList
    setPixel(pixel)

    for (each of pixel’s neighbors)
    {
        if (neighbor is inside region && neighbor not set)
            add neighbor to fillList
    }
}
Determining neighbors

4-connected region

8-connected region
Determining neighbors

Different fill results for 4-connected and 8-connected regions

Original boundary

Magnified area

Fill using 4-connected neighbors

Fill using 8-connected neighbors
Determining “inside-ness” of neighbors

Boundary-defined region
  • Use **boundary fill**
  • Set all connected pixels within the boundary

Interior-defined region
  • Use **flood fill**
  • Set all connected pixels that have similar color
Boundary Fill

1. Region described by a set of bounding pixels
2. A seed pixel is set inside the boundary
Boundary Fill

1. Region described by a set of bounding pixels
2. A seed pixel is set inside the boundary
3. Check if this pixel is a bounding pixel or has already been filled
4. If no to both, fill it and make neighbors new seeds
Boundary Fill

1. Region described by a set of bounding pixels
2. A seed pixel is set inside the boundary
3. Check if this pixel is a bounding pixel or has already been filled
4. If no to both, fill it and make neighbors new seeds
Boundary Fill

1. Region described by a set of bounding pixels
2. A seed pixel is set inside the boundary
3. Check if this pixel is a bounding pixel or has already been filled
4. If no to both, fill it and make neighbors new seeds
Boundary Fill

1. Region described by a set of bounding pixels
2. A seed pixel is set inside the boundary
3. Check if this pixel is a bounding pixel or has already been filled
4. If no to both, fill it and make neighbors new seeds

![Image after 4-connected boundary fill](image-url)
Flood Fill

1. Region is a patch of like-colored pixels
2. A seed pixel is set and a range of colors is defined
Flood Fill

1. Region is a patch of like-colored pixels
2. A seed pixel is set and a range of colors is defined
3. Check if the pixel is in the color range
4. If yes, fill it and make the neighbors news seeds
Flood Fill

1. Region is a patch of like-colored pixels
2. A seed pixel is set and a range of colors is defined
3. Check if the pixel is in the color range
4. If yes, fill it and make the neighbors news seeds
Flood Fill

1. Region is a patch of like-colored pixels
2. A seed pixel is set and a range of colors is defined
3. Check if the pixel is in the color range
4. If yes, fill it and make the neighbors news seeds
Flood Fill

1. Region is a patch of like-colored pixels
2. A seed pixel is set and a range of colors is defined
3. Check if the pixel is in the color range
4. If yes, fill it and make the neighbors news seeds

Image after 4-connected flood fill
Improving Seed Fill

Shortcomings of boundary fill and flood fill:

• Time
  • Large number of recursive calls
  • Pixels might be considered more than once (test if set, test if inside)

• Memory
  • Don’t know how big the fill list should be
  • Could be all of the image pixels
  • More if our algorithm allows us to consider a pixel more than once
Improving Seed Fill

Goal: Improve performance and reduce memory

Strategy: Exploit coherence
  • Structure the order in which neighboring pixels are processed
  • Reduces the number of recursive calls
Neighbor coherence

Neighboring pixels tend to be in the same region
Span coherence

Neighboring pixels on a scan line tend to be in the same region.
Scan line coherence

The filling patterns of adjacent scan lines tend to be similar.
Span-Based Seed Fill Algorithm

1. Start from the seed point
Span-Based Seed Fill Algorithm

1. Start from the seed point
2. Fill the entire horizontal span of pixels inside the region
Span-Based Seed Fill Algorithm

1. Start from the seed point
2. Fill the entire horizontal span of pixels inside the region
3. Determine spans of pixels in the rows above and below the current row that are connected to the current span
4. Add the left-most pixel of these spans to the fill list
Span-Based Seed Fill Algorithm

1. Start from the seed point
2. Fill the entire horizontal span of pixels inside the region
3. Determine spans of pixels in the rows above and below the current row that are connected to the current span
4. Add the left-most pixel of these spans to the fill list
5. Repeat until the fill list is empty
Span-Based Seed Fill Algorithm

1. Start from the seed point
2. Fill the entire horizontal span of pixels inside the region
3. Determine spans of pixels in the rows above and below the current row that are connected to the current span
4. Add the left-most pixel of these spans to the fill list
5. Repeat until the fill list is empty
Span-Based Seed Fill Algorithm

1. Start from the seed point
2. Fill the entire horizontal span of pixels inside the region
3. Determine spans of pixels in the rows above and below the current row that are connected to the current span
4. Add the left-most pixel of these spans to the fill list
5. Repeat until the fill list is empty
Span-Based Seed Fill Algorithm

1. Start from the seed point
2. Fill the entire horizontal span of pixels inside the region
3. Determine spans of pixels in the rows above and below the current row that are connected to the current span
4. Add the left-most pixel of these spans to the fill list
5. Repeat until the fill list is empty
Span-Based Seed Fill Algorithm

1. Start from the seed point
2. Fill the entire horizontal span of pixels inside the region
3. Determine spans of pixels in the rows above and below the current row that are connected to the current span
4. Add the left-most pixel of these spans to the fill list
5. Repeat until the fill list is empty
Span-Based Seed Fill Algorithm

1. Start from the seed point
2. Fill the entire horizontal span of pixels inside the region
3. Determine spans of pixels in the rows above and below the current row that are connected to the current span
4. Add the left-most pixel of these spans to the fill list
5. Repeat until the fill list is empty
Span-Based Seed Fill Algorithm

1. Start from the seed point
2. Fill the entire horizontal span of pixels inside the region
3. Determine spans of pixels in the rows above and below the current row that are connected to the current span
4. Add the left-most pixel of these spans to the fill list
5. Repeat until the fill list is empty
Span-Based Seed Fill Algorithm

1. Start from the seed point
2. Fill the entire horizontal span of pixels inside the region
3. Determine spans of pixels in the rows above and below the current row that are connected to the current span
4. Add the left-most pixel of these spans to the fill list
5. Repeat until the fill list is empty
Approaches to filling regions

**Seed Fill Algorithms**
- Start with an interior seed point and grow
- Pixel-based descriptions

**Raster-Based Filling**
- Fill the interior one raster scan line at a time
- Geometric descriptions
Raster-Based Filling
Axis-aligned rectangle

Defined by its corner points
Axis-aligned rectangle

Filled in a straightforward manner

```cpp
for (j = y_min; j < y_max; j++) {
    for (i = x_min; i < x_max; i++) {
        setPixel(i, j, fillColor)
    }
}
```
Polygon

Described geometrically as a list of connected line segments
  • These edges must form a closed shape
Filling general polygons

Simple approach: Boundary fill

1. Use a line drawing algorithm to draw edges of the polygon with a boundary color
2. Set a seed pixel inside the boundary
Filling general polygons

Simple approach: Boundary fill
1. Use a line drawing algorithm to draw edges of the polygon with a boundary color
2. Set a seed pixel inside the boundary
3. Inside pixels are connected to the seed pixel via other inside pixels
Problems with boundary fill

Pixels are drawn on both sides of the line
  • The polygon contains pixels outside of the outline
  • Polygons with shared edges will have overlapping pixels

Efficiency
  • Would be more efficient to combine edge drawing and filling in one step
Edge pixels

Adjacent polygons share edges
When rendered, some pixels along the edges are shared
Need to know what color to use for shared edge pixels
Edge pixels

If we draw all edge pixels for each polygon...

- Shared pixels will be rendered more than once
- If setPixel() \textit{overwrites} the current pixel, the last polygons drawn will look larger

Green triangle written last
Edge pixels

If we draw all edge pixels for each polygon...

- Shared pixels will be rendered more than once
- If setPixel() overwrites the current pixel, the last polygons drawn will look larger

Blue triangle written last
Edge pixels

If we draw all edge pixels for each polygon...

- Shared pixels will be rendered more than once
- If `setPixel()` overwrites the current pixel, the last polygons drawn will look larger
- If `setPixel()` *blends* the background color with the foreground color, shared edge pixels will have a blended color

Edge color different than either triangle
Edge pixels

If we do not draw the edge pixels...

- Only interior pixels are drawn
- Gaps appear between polygons and the background shows through

Gaps between adjacent triangles
Edge pixels

Solution: Only draw some of the edges for each polygon
  • Follow convention to determine which edge to draw when edge pixels are shared
  • e.g., draw the polygon’s left edges and horizontal bottom edges
Polygon

Described geometrically as a list of connected line segments
  • These edges must form a closed shape

Could use a seed fill algorithm
  • Rasterize the edges to get a bounding pixel description
  • Apply boundary fill

Can do better by using information about the edges
Raster-Based Filling of polygons

Approach:
Fill polygons in raster-scan order
Fill spans of pixels inside the polygon along each scan line
Polygon

A sequence of vertices connected by edges
Assume that vertices have been rasterized

For each point encountered in the scan, determine whether it is inside the polygon -- a **fill rule**:
- **Even-odd parity rule**
- **Non-zero winding number rule**
Even-Odd Parity Rule

Inside-outside test for a point P:

1. Draw line from P to infinity
   - Any direction
   - Does not go through any vertex

2. Count the number of times the line crosses an edge
   - If the number of crossings is odd, P is inside
   - If the number of crossings is even, P is outside
Non-Zero Winding Number Rule

The outline of the shape must be directed

- The line segments must have a consistent direction so that they form a continuous, closed path
Non-Zero Winding Number Rule

Inside-outside test:

1. Determine the \textbf{winding number} $W$ of $P$
   a. Initialize $W$ to zero and draw a line from $P$ to infinity
   b. If the line crosses an edge directed from bottom to top, $W++$
   c. If the line crosses an edge directed from top to bottom, $W--$

2. If the $W = 0$, $P$ is outside
3. Otherwise, $P$ is inside
Vertices

Check the vertex type

• Line to infinity **pierces** the edge
  • The vertex connects two upwards or two downwards edges
  • Process a single edge crossing

• Line to infinity **grazes** the edge
  • The vertex connects an upwards and a downwards edge
  • Don’t process any edge crossings
Vertices

An alternative is to ensure that the line doesn’t intersect a vertex

Either use a different line if the first line intersects a vertex
  • Could be costly if you have to try several lines

Or preprocess edge vertices to ensure that none of them fall on a scan line
  • Add a small floating point value to each vertex y-position
Standard polygons

Do not self intersect and do not contain holes

The even-odd parity rule and the non-zero winding number rule give the same results for standard polygons
General polygons

Can be self intersecting
Can have interior holes

The non-zero winding number rule and the even-odd parity rule can give different results for general polygons
Even-Odd Parity

Non-Zero Winding
Raster-Based Filling

Fill polygons in raster-scan order
  • Fill spans of pixels inside the polygon along each horizontal scan line
    • More efficient addressing by accessing spans of pixels
    • Only test pixels at the span endpoints
Raster-Based Filling

For each scan line
  • Determine points where the scan line intersects the polygon
Raster-Based Filling

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
Raster-Based Filling: Using even-odd parity rule

for (each scan line j)
{
    find the intersections between j and each edge
    sort the intersections by increasing x-value

    //Use the even-odd parity rule to set interior points
    for (each pair of x-intersections \((x_1, x_2)\))
    {
        while \((x_1 \leq i < x_2)\)
            setPixel(i, j, fillColor)
    }
}
Raster-Based Filling: Using even-odd parity rule

for (each scan line j)
{
    find the intersections between j and each edge
    sort the intersections by increasing x-value

    //Use the even-odd parity rule to set interior points
    for (each pair of x-intersections \( (x_1, x_2) \))
    {
        while \( (x_1 \leq i < x_2) \)
            setPixel(i, j, fillColor)
    }
}
Edge pixels

Fill pixels with centers in between pairs of intersections of the scan line with the polygon edges

Convention: If an intersection point lies at a pixel center
- The pixel is included if it is a leftmost edge
- The pixel is not included if it is a rightmost edge
for (each scan line j)
{
    //Determine points where j intersects the edges

    //Set pixels according to the winding number

}
Summary

Region descriptions

Seed fill algorithms (pixel-based descriptions)
   Boundary fill (boundary-based descriptions)
   Flood fill (interior-based descriptions)
   Handling of shared edges

Raster-based fill (geometric descriptions)
   Fill rules
      Even-odd parity
      Non-zero winding number
   Handling of shared vertices
Next time

Efficient raster-based fill algorithms
  x-intersection array
  Edge lists
  Pineda’s algorithm
Related ideas

Color replacement

Source: digiretus.com Photoshop tutorials
Related ideas

Color replacement

Source: gimp.org tutorials
Related ideas

Colorization of black-and-white stills and videos

Related ideas

Inpainting