Filling Polygons

Dr Nicolas Holzschuch
University of Cape Town
e-mail: holzschu@cs.uct.ac.za
Map of the lecture

• Filling rectangles
  – algorithm
  – problems and solutions

• Filling polygons:
  – algorithm
  – problems and solutions
  – algorithm details: active-edge table
Filling rectangles

• Rectangle defined by: \((x_{\text{min}}, x_{\text{max}}) \times (y_{\text{min}}, y_{\text{max}})\)
• Fill it using scan-line algorithm:

```plaintext
for y = y_{\text{min}} to y_{\text{max}}
    for x = x_{\text{min}} to x_{\text{max}}
        LightPixel(x, y)
    end_for
end_for
```
Problems and solutions

- Two rectangles sharing an edge:
  - the edge will be drawn twice
- Solution: revised algorithm
  
  ```
  for y = ymin to ymax-1
      for x = xmin to xmax-1
          LightPixel(x,y)
      end_for
  end_for
  ```

- Only draw if it’s below or on the left
Filling Polygons

• Main algorithm:
  ```plaintext
  for y = 0 to height_screen
    find intersection polygon/scanline
    fill the intersection
  end_for
  ```

• Intersection polygon-scanline:
  – the algorithm in a moment
  – the specifications now
Filling Polygons: example

Polygon drawn so far

Current scanline
Filling Polygons: example (2)

Extremities, computed using Bresenham-like alg.

- What happens with two neighbouring polygons?
Filling Polygons: example (3)

Integer intersections: do as we did with rectangles

Keep the extremities inside
Filling Polygons: inside/outside

• **Even/odd:**
  – for each scanline, count number of edges encountered so far:
    • even: outside
    • odd: inside

• **Edge orientation:**
  – the edge is oriented, so is the scanline
  – scanline entering: add one to the counter
  – scanline leaving: remove one
Inside/outside: example

Even/Odd

Edge orientation (1)

Edge orientation (2)
Computing the extremities

• Scanline-edge intersection:
  – not exactly Bresenham algorithm
  – requirements are more relaxed

• Active-edge table:
  – list of edges
  – ordered for maximum efficiency
We don’t need Bresenham

• Something simpler may suffice:

Bresenham
**Scanline-edge intersection**

- Moving from one scanline to the next:
  
  \[ x += \frac{1}{m} \]
  
  - with \( m \), the slope of the edge:
    
    \[ m = \frac{y_{\text{max}} - y_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} \]
    
    - therefore, \( x \) can always be expressed as:
      
      \[ x = a + \frac{b}{y_{\text{max}} - y_{\text{min}}} \]
      
      \((a \text{ and } b \text{ are integers})\)
Scanline-edge intersection (2)

• Keep $x$ as two integers $(a,b)$
• moving to the next scanline:
  
  ```
  writePixel(a,y)
  b += (x_{\text{max}} - x_{\text{min}})
  
  while (b >= (y_{\text{max}} - y_{\text{min}})) {
    
    b -= y_{\text{max}} - y_{\text{min}}
    a ++
  }
  ```
Scanline-edge intersection (3)

- Rounding-up:
  - avoid lighting exterior pixels
  - draw pixel \((a,y)\) if it is a right-edge
  - draw pixel \((a+1,y)\) if it is a left-edge
Edge Table

• Keep bucket list of all edges
  – one bucket per scanline
• Edges inserted at bucket of their $y_{\text{min}}$
• Within a bucket:
  – sorted by order of $x$ coordinate at $y_{\text{min}}$
• Entries contain:
  – $y_{\text{max}}$, $x$ value at $y_{\text{min}}$, and $1/m$
Edge Table: example

\begin{align*}
  \text{y}=0 & \quad \text{y}=h \\
  y_{\max} & \quad 3 \\
  9 & \\
  3/2 & \\
  x_{\min} & \quad 9 \\
  3 & \\
  1/4 & \\
  1/m & \quad 3/2 \\
  11 & \\
  7 & \\
  1/2 & \\
  19 & \\
  6 & \\
  4 & \\
  1/2 & \\
  13 & \\
  11 & \\
  9/2 & \\
\end{align*}
Active Edge Table

• Keep list of edges that are intersected by the scanline
• Use Edge Table
• Update at each scanline
• Start with $y$ at smallest non-empty bucket
• Initialize AET to be empty
Active Edge Table (2)

• For each $y$ value:
  – move bucket $y$ content from ET to AET
  – sort AET on $x$ values
  – fill in desired pixels on the scanline using AET
  – remove from AET edges with $y_{\text{max}} = y$
  – for each edge in the AET, update $x$ for the next scanline
Active Edge Table: example

- Sample AET:

- Draw from 3 to 7, then 11 to 17
Drawing polygons: summary

• A simple algorithm — in theory
• Difficult to implement, in practice
• Everything is in the data structure
  – ET
  – AET
• Cornerstone for other algorithms:
  – visible-surface determination
  – shading (Gouraud shading, Phong shading)
Special case: triangles

• In a triangle, there are only two edges on a given scanline

• Simpler to draw:
  – no need for ET/AET

• Some softwares prefer to cut into triangles, then fill those triangles:
  – easier for hardware and assembly
  – efficiency linked to number of triangles