Image Processing and Computer Graphics

Rasterization

Matthias Teschner

Computer Science Department
University of Freiburg
Motivation

- rasterization is
  - the transformation of geometric primitives (line segments, circles, polygons) into a raster image representation, i.e. pixel positions
  - the estimation of an appropriate set of pixel positions to represent a geometric primitive
- the rendering pipeline
  - processes vertices (transformations and lighting)
  - assembles primitives from vertices in window space and topology information
  - rasterizes primitives, i.e. converts primitives to fragments with interpolated attributes
  - processes fragments, updates the framebuffer
Motivation

- computation of pixel positions that represent a primitive

Line (segment) rasterization

Circle rasterization

Polygon rasterization

[Wikipedia: Rasterung von Linien, Rasterung von Polygonen, Rasterung von Kreisen]
Outline

- lines
- circles
- polygons
General Setting

- components of start and end point of a line are integer values \( p_b = (x_b, y_b) \quad p_e = (x_e, y_e) \)
- lines are represented as
  \[ y = mx + b \quad \text{or} \quad F(x, y) = ax + by + c = 0 \]
- algorithms are often restricted to \( 0 \leq m \leq 1 \)
  - arbitrary lines are handled by employing symmetries
- algorithms consist of an initialization step and a loop
  - efficiency of a particular algorithm depends on the line length
A Simple Algorithm

- \[ y = \frac{\Delta y}{\Delta x} (x - x_b) + y_b = \frac{y_e - y_b}{x_e - x_b} (x - x_b) + y_b \]

- for each \( x_b \leq x_i \leq x_e \)
  compute \( y_i = \text{round}(y(x_i)) \)
  set \( p_i = (x_i, y_i) \)

- efficient incremental update
  \[
y(x_{i+1}) - y(x_i) = m(x_{i+1} - x_b) + y_b - (m(x_i - x_b) + y_b) = m(x_{i+1} - x_i) = m \\
y(x_{i+1}) = y(x_i) + m
  \]
Generalization

-1 \leq m \leq 1

- x_b < x_e: increment \, x_i, compute (x_i, \text{round}(y(x_i)))
- x_e < x_b: decrement \, x_i, compute (x_i, \text{round}(y(x_i)))

m > 1 \text{ or } m < -1

- y_b < y_e: increment \, y_i, compute (\text{round}(x(y_i)), y_i)
- y_e < y_b: decrement \, y_i, compute (\text{round}(x(y_i)), y_i)
**Bresenham Algorithm**
*(Midpoint Algorithm)*

- explicit form of a line
  \[ y = \frac{y_e - y_b}{x_e - x_b} (x - x_b) + y_b \]

- implicit form of a line
  \[ 0 = \frac{\Delta y}{\Delta x} x - y + y_b - \frac{\Delta y}{\Delta x} x_b = \Delta y \cdot x - \Delta x \cdot y + \Delta x \cdot y_b - \Delta y \cdot x_b \]

- implicit form of a line
  - for all points \((x, y)\) on a line
    \[ F(x, y) = \Delta y \cdot x - \Delta x \cdot y + \Delta x \cdot y_b - \Delta y \cdot x_b = 0 \]
  - all points with \(F(x, y) > 0\) are on one side of the line
  - all points with \(F(x, y) < 0\) are on the other side
Bresenham Algorithm

- for incremented values of x, the algorithm decides whether to increment y or not
- based on the current pixel \((x_i, y_i)\), the algorithm decides whether to choose \((x_i + 1, y_i)\) or \((x_i + 1, y_i + 1)\) (E east, NE north east)
- F is evaluated at the next midpoint \(F(x_i + 1, y_i + \frac{1}{2})\)
- \(F(x_i + 1, y_i + \frac{1}{2}) \geq 0 \Rightarrow\) choose NE
- \(F(x_i + 1, y_i + \frac{1}{2}) < 0 \Rightarrow\) choose E

[Wikipedia: Rasterung von Linien]
Incremental Update of the Decision Variable

- decision variable \( d_i = F(x_i + 1, y_i + \frac{1}{2}) \)
- incremental update from \( d_i \) to \( d_{i+1} \) depending on \( d_i \)
- \( d_i \geq 0 \Rightarrow \) choose NE, \( d_{i+1} = F(x_i + 2, y_i + 1 + \frac{1}{2}) \)
- \( d_i < 0 \Rightarrow \) choose E, \( d_{i+1} = F(x_i + 2, y_i + \frac{1}{2}) \)
- in case of \( d_i \geq 0 \) :
  \[ \Delta_{NE} = d_{i+1} - d_i = \Delta y \cdot (x_i + 2) - \Delta x \cdot (y_i + \frac{3}{2}) + c - (\Delta y \cdot (x_i + 1) - \Delta x \cdot (y_i + \frac{1}{2}) + c) \]
  \[ \Delta_{NE} = \Delta y - \Delta x \]
- in case of \( d_i < 0 \) :
  \[ \Delta_{E} = d_{i+1} - d_i = \Delta y \cdot (x_i + 2) - \Delta x \cdot (y_i + \frac{1}{2}) + c - (\Delta y \cdot (x_i + 1) - \Delta x \cdot (y_i + \frac{1}{2}) + c) \]
  \[ \Delta_{E} = \Delta y \]
Bresenham Algorithm
Initialization

- for start point \( \mathbf{p}_b = (x_b, y_b) \),
  decision variable \( d_1 \) can be initialized as
  \[
  d_1 = F(x_b + 1, y_b + \frac{1}{2}) = \Delta y \cdot (x_b + 1) - \Delta x \cdot (y_b + \frac{1}{2}) + c
  \]
  \[
  = \Delta y \cdot x_b - \Delta x \cdot y_b + c + \Delta y - \frac{1}{2} \Delta x
  \]
  \[
  = F(x_b, y_b) + \Delta y - \frac{1}{2} \Delta x
  \]
  \[
  = \Delta y - \frac{1}{2} \Delta x
  \]

- floating-point arithmetic is avoided by considering \( 2 \cdot F(x,y) \):
  \[
  d_1 = 2\Delta y - \Delta x
  \]
  \[
  \Delta_E = 2\Delta y
  \]
  \[
  \Delta_{NE} = 2\Delta y - 2\Delta x
  \]
Bresenham Algorithm
Implementation

```c
void BresenhamLine(int xb, int yb, int xe, int ye) {

        int dx, dy, incE, incNE, d, x, y;

dx = xe - xb; dy = ye - yb;
d = 2*dy - dx;
incE = 2*dy;
incNE = 2*(dy - dx);
x = xb; y = yb;
WritePixel(x, y); /* write start pixel */
while (x < xe) {
        x++;
        if (d < 0)    /* choose E */
                d += incE;
        else {
                d += incNE; /* choose NE */
                y++;
        }
        WritePixel(x, y);
}
```

University of Freiburg – Computer Science Department – Computer Graphics - 12
Bresenham Algorithm
Decision Variable

[Wikipedia: Rasterung von Linien]
**Generalization**

- ** incr y by 1**
- ** incr x by 0 or -1**

increment x by -1
increment y by 0 or 1

increment x by 1
increment y by 0 or 1

increment x by 1
increment y by 0 or -1

increment x by -1
increment y by 0 or -1

increment x by 0 or -1
increment y by -1

increment x by 0 or 1
increment y by -1
Run Length Slices

- estimate x-values where the y-value is incremented

\[ x_i \] is the (floating-point) intersection of the line with the line defined by \((x_b, y_b+i+0.5)\) and \((x_e, y_b+i+0.5)\)

- increment y, compute \(x_i\), draw pixels with the same y-value up to \(\lfloor x_i \rfloor\)

[Wikipedia: Rasterung von Linien]
Run Length Slices

- line: \( y = \frac{\Delta y}{\Delta x} (x - x_b) + y_b \)
  \[
x = \frac{\Delta x}{\Delta y} (y - y_b) + x_b
  \]

- x-components of the intersection at \( y = y_b + i + \frac{1}{2} \):
  \[
x_i = \frac{\Delta x}{\Delta y} (y_b + i + \frac{1}{2} - y_b) + x_b
  \]

- differential update using \( x_{i+1} - x_i = \frac{\Delta x}{\Delta y} \)

- initialization: \( x_1 = \frac{3\Delta x}{2\Delta y} + x_b \)
- loop: \( x_{i+1} = x_i + \frac{\Delta x}{\Delta y} \)
Issues / Limitations

- aliasing
  - stair-case artifacts, varying line intensity
- clipping
  - artifacts due to round-off of clipped end points

same number of pixels for lines with different length

aliasing can be addressed by rendering thick lines with varying pixel intensities

no anti-aliasing

with anti-aliasing

[Wikipedia: Antialiasing, Rasterung von Linien]

University of Freiburg – Computer Science Department – Computer Graphics - 17
Summary - Lines

- line rasterization algorithms are usually described for a subset of lines and generalized using symmetries
- incremental updates are often employed
- Bresenham avoids floating-point arithmetic
- improved algorithms address aliasing / clipping artifacts
- note that the algorithms do not compute all pixels that are intersected by the line
Outline

- lines
- circles
- polygons
General Setting

- circle with center at (0,0) and radius r
- implicit representation
  \[ F(x, y) = x^2 + y^2 - r^2 = 0 \]
- algorithms compute only one eighth of a circle
  - if (x, y) is on the circle, then (±x, ±y) and (± y, ± x) are on the circle
Metzger Algorithm

- if \((x_i, y_i)\) is on the circle, the algorithm decides whether \(R_O = (x_i, y_i + 1)\) or \(R_I = (x_i - 1, y_i + 1)\) is the next point on the circle
- the point with the shortest distance to the circle is chosen

\[
d_I = r - \|R_I\| = r - \sqrt{(x_i - 1)^2 + (y_i + 1)^2}
\]

\[
d_O = \|R_O\| - r = \sqrt{x_i^2 + (y_i + 1)^2} - r
\]

- if \(d_I \leq d_O \Rightarrow R_I\)
- if \(d_I > d_O \Rightarrow R_O\)
Horn Algorithm

- The algorithm checks whether \((x_i - \frac{1}{2}, y_i + 1)\) is outside
  - if so, it chooses \((x_i - 1, y_i + 1)\)
  - if not, it chooses \((x_i, y_i + 1)\)
- Decision variable
  \[d_i = (x_i - \frac{1}{2})^2 + y_i^2 - r^2\]
- Incremental update
  - if \(d_i < 0\) \(\Rightarrow (x_{i+1}, y_{i+1}) = (x_i, y_i + 1)\)
    \[d_{i+1} = (x_i - \frac{1}{2})^2 + (y_i + 1)^2 - r^2\]
    \[d_{i+1} = d_i + 2y_i + 1\]
  - if \(d_i \geq 0\) \(\Rightarrow (x_{i+1}, y_{i+1}) = (x_i - 1, y_i + 1)\)
    \[d_{i+1} = d_i + 2y_i + 1 - 2x_i + 2\]

[Wikipedia: Rasterung von Kreisen]
Horn Algorithm Implementation

```c
void HornCircle(int r) {

  int d, x, y;

  d = -r;
  x = r;
  y = 0;

  while (y > x) {
    WritePixel(x, y); /* and symmetric pixels */
    d += 2*y + 1;
    y += 1;
    if (d >= 0) {
      d += -2*x + 2;
      x += -1;
    }
  }
}
```
Bresenham Algorithm
(Midpoint Algorithm)

- \( F(x, y) = x^2 + y^2 - r^2 = 0 \Rightarrow (x, y) \) is on the circle
- Based on the current pixel \((x_i, y_i)\), the algorithm decides whether to choose \((x_i + 1, y_i)\) or \((x_i + 1, y_i - 1)\) (E east, SE southeast)
- \( F \) is evaluated at the next midpoint
  - \( F(x_i + 1, y_i - \frac{1}{2}) \)
  - \( F(x_i + 1, y_i - \frac{1}{2}) \geq 0 \Rightarrow \) choose SE
  - \( F(x_i + 1, y_i - \frac{1}{2}) < 0 \Rightarrow \) choose E

[Wikipedia: Rasterung von Kreisen]
Incremental Update of the Decision Variable

- decision variable $d_i = F(x_i + 1, y_i - \frac{1}{2})$
- incremental update from $d_i$ to $d_{i+1}$ depending on $d_i$
  - $d_i \geq 0 \Rightarrow$ choose SE, $d_{i+1} = F(x_i + 2, y_i - 1 - \frac{1}{2})$
  - $d_i < 0 \Rightarrow$ choose E, $d_{i+1} = F(x_i + 2, y_i - \frac{1}{2})$

- in case of $d_i \geq 0$:
  $\Delta_{SE} = 2x_i - 2y_i + 5$

- in case of $d_i < 0$:
  $\Delta_E = 2x_i + 3$
Incremental Update of the Increments

- four patterns of a set of three adjacent points
  - (1) \((x_i, y_i), (x_i + 1, y_i), (x_i + 2, y_i)\)
  - (2) \((x_i, y_i), (x_i + 1, y_i), (x_i + 2, y_i - 1)\)
  - (3) \((x_i, y_i), (x_i + 1, y_i - 1), (x_i + 2, y_i - 1)\)
  - (4) \((x_i, y_i), (x_i + 1, y_i - 1), (x_i + 2, y_i - 2)\)

- increments \(\Delta_{E,i} = 2x_i + 3\) \(\Delta_{SE,i} = 2x_i - 2y_i + 5\)
  - if the algorithm moves towards E
  - (1) \(\Delta_{E,i+1} = 2(x_i + 1) + 3 \Rightarrow \Delta_{E,i+1} = \Delta_{E,i} + 2\)
  - (2) \(\Delta_{SE,i+1} = 2(x_i + 1) - 2y_i + 5 \Rightarrow \Delta_{SE,i+1} = \Delta_{SE,i} + 2\)
  - if the algorithms moves towards SE
  - (3) \(\Delta_{E,i+1} = 2(x_i + 1) + 3 \Rightarrow \Delta_{E,i+1} = \Delta_{E,i} + 2\)
  - (4) \(\Delta_{SE,i+1} = 2(x_i + 1) - 2(y_i - 1) + 5 \Rightarrow \Delta_{SE,i+1} = \Delta_{SE,i} + 4\)
Incremental Update of Increments

- point \((x_i, y_i)\) is on the circle
- if next point is E,
  - \(d_i = d_i + \Delta_E,i\)
  - \(\Delta_E,i = \Delta_E,i + 2\)
  - \(\Delta_{SE,i} = \Delta_{SE,i} + 2\)
- if next point is SE,
  - \(d_i = d_i + \Delta_{SE,i}\)
  - \(\Delta_E,i = \Delta_E,i + 2\)
  - \(\Delta_{SE,i} = \Delta_{SE,i} + 4\)
Bresenham Algorithm
Initialization

- at point \((0, r)\)
  
  \[
  d_1 = F(0 + 1, r - \frac{1}{2}) = 1 + (r - \frac{1}{2})^2 - r^2 = \frac{5}{4} - r
  \]
  
  \[
  \Delta_{SE} = 2r + 5
  \]
  
  \[
  \Delta_E = 3
  \]

- as \(d\) is incremented only by integer values, \(d_1 = 1 - r\)
Bresenham Algorithm
Implementation

```c
void BresenhamCircle (int r) {
    int x, y, d, deltaE, deltaSE;

    x = 0; y = r; d = 1 - r; deltaE = 3; deltaSE = -2*r + 5;

    WritePixel(x, y); /* and symmetric points */
    while (y > x) {
        if (d < 0) {
            /* choose E */
            d += deltaE;
            deltaE += 2;
            deltaSE += 2;
        }
        else {
            /* choose SE */
            d += deltaSE;
            deltaE += 2;
            deltaSE += 4;
            y--;
        }
        x++;
        WritePixel(x, y); /* and symmetric points */
    }
}
```
Summary - Circles

- circle rasterization algorithms are usually described for one eighth of a circle and generalized using symmetries
- incremental updates are often employed
- floating-point arithmetic is avoided
Outline

- lines
- circles
- polygons
General Setting

- a polygon is defined by edges
- the polygon should be closed to allow inside / outside classification
- rasterization estimates all pixel positions inside a polygon
- in general simple, but
  - if adjacent polygons share an edge, each pixel on the edge should belong to exactly one polygon
  - no pixel along the edge should be missed
  - no pixel along the edge should be rasterized twice
Edge List Algorithms

- compute intersections of non-horizontal polygon edges with lines (scanlines)
- intersections are computed for $y = y_i + 0.5$
- fill pixel positions in-between two intersection points
  - scan from left to right
  - enter the polygon at the first intersection, leave the polygon at the next intersection

[Wikipedia: Rasterung von Polygonen]
Edge Fill Algorithms

- for each polygon edge
  - process all scanlines intersected by the edge
  - invert all pixels with an x-component larger than the intersection point

[Wikipedia: Rasterung von Polygonen]
Fence Fill Algorithm

- for each polygon edge
  - process all scanlines intersected by the edge
  - if $x_{\text{intersection}} \geq x_{\text{fence}}$ invert all pixels with $x_{\text{fence}} \leq x_{\text{pixel}} < x_{\text{intersection}}$
  - if $x_{\text{intersection}} < x_{\text{fence}}$ invert all pixels with $x_{\text{intersection}} \leq x_{\text{pixel}} < x_{\text{fence}}$

[Wikipedia: Rasterung von Polygonen]
Summary - Polygons

- polygon rasterization algorithms work for closed polygons
  - inside / outside classification
- rasterization estimates all pixel positions inside a polygon
- processing of edges has to consider that pixels on shared edges should be rasterized exactly once
Summary

- Vertices in window space and topology information are used to assemble primitives.
- Rasterization converts primitives to fragments with interpolated attributes.
  - Rasterization of lines
  - Rasterization of circles
  - Rasterization of polygons
- Rasterized pixel positions with interpolated attributes are further processed in the rendering pipeline.