Outline

- Line drawing
- Digital differential analyzer
- Bresenham’s algorithm
- Cohen-Sutherland line clipping
- Parametric line clipping

Scan-Conversion Algorithms

- Scan-Conversion: Computing pixel coordinates for ideal line on 2D raster grid
- Pixels best visualized as circles/dots
  - Why? Monitor hardware

Digital Differential Analyzer (DDA)

- If slope is less then \(_x = 1\) (otherwise \(_y = 1\))
- Compute corresponding \(_y(x)\)
- \(x_{k+1} = x_k + \_x\)
- \(y_{k+1} = y_k + \_y\)
- Issues:
  - Avoiding floating point ops, multiplications, and real numbers

Bresenham’s Algorithm

- 1965 @ IBM
- Basic Idea:
  - Only integer arithmetic
  - Incremental
- Consider the implicit equation for a line: \(f(x,y) = ax + by + c = 0\)

Bresenham’s Algorithm

Given:

*implicit line equation: \(f(x,y) = ax + by + c = 0\)*

Let: \(d_x = r_x - q_x\), \(d_y = r_y - q_y\)

where \(r\) and \(q\) are points on the line

Then: \(a = d_y\), \(b = -d_x\), \(c = -(q_xr_y - r_xq_y)\)

Observe that all of these are integers and: \(f(x,y) > 0\) for points above the line

\(f(x,y) < 0\) for points below the line

Now......
**Bresenham’s Algorithm**

- Suppose we just finished \((p_x, p_y)\)
  - (assume \(0<\text{slope}<1\))
  - other cases symmetric
- Which pixel next?
  - \(E\) or \(NE\)

  \[
  \text{East (} E = (p_x + 1, p_y) \text{)}
  \]
  \[
  \text{NorthEast (} NE = (p_x + 1, p_y + 1) \text{)}
  \]

**Assume:**

- \(q\) = exact \(y\) value at \(x = p_x + 1\)
- \(y\) midway between \(E\) and \(NE\): \(m = p_y + 1/2\)

**Observe:**

- If \(q < m\), then pick \(E\)
- Else pick \(NE\)

- If \(q = m\), it doesn’t matter

**Case I: When \(E\) is next**

- What increment for computing an new \(D\)?

  \[
  D_{\text{new}} = f(p_x + 2, p_y + (1/2))
  \]
  \[
  = 2a(p_x + 2) + 2b\left(p_y + \frac{1}{2}\right) + 2c
  \]
  \[
  = 2ap_x + 2bp_y + (4a + b + 2c)
  \]
  \[
  = 2ap_x + 2bp_y + (2a + b + 2c) + 2a
  \]
  \[
  = D + 2a = D + 2d_x
  \]
  - Hence, increment by: \(2d_x\)

**Case II: When \(NE\) is next**

- What increment for computing an new \(D\)?

  \[
  D_{\text{new}} = f(p_x + 2, p_y + (1 + (1/2))
  \]
  \[
  = 2a(p_x + 2) + 2b\left(p_y + \frac{3}{2}\right) + 2c
  \]
  \[
  = 2ap_x + 2bp_y + (4a + 3b + 2c)
  \]
  \[
  = 2ap_x + 2bp_y + (2a + b + 2c) + (2a + 2b)
  \]
  \[
  = D + 2(a + b) = D + 2(d_x - d_y)
  \]
  - Hence, increment by: \(2(d_x - d_y)\)
How to get an initial value for $D$?

- Suppose we start at: $(q_x, q_y)$
- Initial midpoint is: $(q_x + 1, q_y + 1/2)$

Then:

$$D_{init} = f(q_x + 1, q_y + 1/2)$$

$$= 2a(q_x + 1) + 2b\left(q_y + \frac{1}{2}\right) + 2c$$

$$= (2aq_x + 2bq_y + 2c) + (2a + b)$$

$$= 0 + 2a + b$$

$$= 2d_x - d_y$$

---

The Algorithm

```c
void bresenham(int dx, dy, int x, int y) {
    int dx2 = dx * dx, dy2 = dy * dy;
    int D = dy2 - dx2;
    // Initial decision value
    int y2 = dy;    // Start at (x, y)

    for (int x = x; x <= x + dx; x++) {
        D += y2;
        if (D > dx2) {
            y2 += dy2;
        } else {
            y2 -= dy2;
        }
    }
}
```

Assumptions: $q_x \leq r_x$

Pre-computed: $2d_y, 2(d_x - d_y)$

---

Bresenham's Algorithm: Example

- Initial midpoint: $(0, 0)$
- Decision value: $D = 2d_y - 2d_x$
- Starting point: $(0, 0)$
- Next point: $(1, 1)$
- Decision value: $D = 2(1) - 2(0) = 2$
- Next point: $(2, 2)$
- Decision value: $D = 2(2) - 2(1) = 2$
- Next point: $(3, 3)$
- Decision value: $D = 2(3) - 2(2) = 2$
- Next point: $(4, 4)$
- Decision value: $D = 2(4) - 2(3) = 2$
- Next point: $(5, 5)$
- Decision value: $D = 2(5) - 2(4) = 2$
- Next point: $(6, 6)$
- Decision value: $D = 2(6) - 2(5) = 2$
- Next point: $(7, 7)$
- Decision value: $D = 2(7) - 2(6) = 2$

---

Bresenham's Algorithm: Example

- Initial midpoint: $(0, 0)$
- Decision value: $D = 2d_y - 2d_x$
- Starting point: $(0, 0)$
- Next point: $(1, 1)$
- Decision value: $D = 2(1) - 2(0) = 2$
- Next point: $(2, 2)$
- Decision value: $D = 2(2) - 2(1) = 2$
- Next point: $(3, 3)$
- Decision value: $D = 2(3) - 2(2) = 2$
- Next point: $(4, 4)$
- Decision value: $D = 2(4) - 2(3) = 2$
- Next point: $(5, 5)$
- Decision value: $D = 2(5) - 2(4) = 2$
- Next point: $(6, 6)$
- Decision value: $D = 2(6) - 2(5) = 2$
- Next point: $(7, 7)$
- Decision value: $D = 2(7) - 2(6) = 2$

---

Bresenham's Algorithm: Example

- Initial midpoint: $(0, 0)$
- Decision value: $D = 2d_y - 2d_x$
- Starting point: $(0, 0)$
- Next point: $(1, 1)$
- Decision value: $D = 2(1) - 2(0) = 2$
- Next point: $(2, 2)$
- Decision value: $D = 2(2) - 2(1) = 2$
- Next point: $(3, 3)$
- Decision value: $D = 2(3) - 2(2) = 2$
- Next point: $(4, 4)$
- Decision value: $D = 2(4) - 2(3) = 2$
- Next point: $(5, 5)$
- Decision value: $D = 2(5) - 2(4) = 2$
- Next point: $(6, 6)$
- Decision value: $D = 2(6) - 2(5) = 2$
- Next point: $(7, 7)$
- Decision value: $D = 2(7) - 2(6) = 2$
Some issues with Bresenham’s Algorithms

- Pixel ‘density’ varies based on slope
  - straight lines look darker, more pixels per CM

- Endpoint order
  - Line from P1 to P2 should match P2 to P1
  - Always choose E when hitting M, regardless of direction

References:
- 1994 Foley/VanDam/Finer/Huges/Phillips ICG
Some issues with Bresenham’s Algorithms

- How to handle the line when it hits the clip window?
- Vertical intersections
  - Could change line slope
  - Need to change init cond.
- Horizontal intersections
  - Again, changes in the boundary conditions
  - Can’t just intersect the line w/ the box

Scissoring Clipping

Performed during scan conversion of the line (circle, polygon)

Compute the next point \((x,y)\)

If \(x_{\min} \leq x \leq x_{\max}\) and \(y_{\min} \leq y \leq y_{\max}\)

Then output the point.

Else do nothing

- Issues with scissoring:
  - Too slow
  - Does more work then necessary
  - We are not going to use it.

The Cohen-Sutherland Line Clipping Algorithm

- How to clip lines to fit in windows?
  - Easy to tell if whole line falls w/in window
  - Harder to tell what part falls inside
- Consider a straight line \(P_0 = (x_0,y_0)\) and \(P_1 = (x_1,y_1)\)
- And window: \(WT\), \(WB\), \(WL\) and \(WR\)

Cohen-Sutherland

Basic Idea:

- **First**, do easy test
  - Completely inside or outside the box?
- If no, we need a more complex test
  - **Note**: we will also need to figure out how scan line meets the box

Cohen-Sutherland

- The Easy Test:
  - Compute 4-bit code based on endpoints \(P_0\) and \(P_1\)
  - Window \(0000\)
  - \(WT\) \(0001\)
  - \(WB\) \(0010\)
  - \(WL\) \(0100\)
  - \(WR\) \(0110\)

  Bit 1: 1 if point is above window, \(i.e. y > WT\)
  Bit 2: 1 if point is below window, \(i.e. y < WB\)
  Bit 3: 1 if point is right of window, \(i.e. x > WL\)
  Bit 4: 1 if point is left of window, \(i.e. x < WR\)

- Line is completely visible iff both code values of endpoints are \(0\), \(i.e. C_0 \cap C_1 = 0\)

- If line segments are completely outside the window, then \(C_0 \cap C_1 \neq 0\)
Cohen-Sutherland

Otherwise, we clip the lines:
- We know that there is a bit flip, w.r.t. assume its \((x_0, y_0)\)
- Which bit? Try `em all!
  - Suppose its bit 4
  - Then \(x_0 < WL\) and we know that \(x_1 > WL\)
- We need to find the point: \((x_c, y_c)\)

\[
\begin{align*}
wl_{x_0} & \equiv WL-x_0 \\
y_0 & = \frac{WL-x_0}{x_1-x_0} (y_1-y_0) + y_0
\end{align*}
\]

Cohen-Sutherland

- Replace \((x_0, y_0)\) with \((x_c, y_c)\)
- Re-compute values
- Continue until all points are inside the clip window

QED

Parametric Line Clipping

- Developed by Cyrus and Beck in 1978
- Used to clip 2D/3D lines against convex polygon/polyhedron
- Liang and Barsky (1984) algorithm efficient in clipping upright 2D/3D clipping regions
- First we will follow original Cyrus-Beck development to introduce parametric clipping
- Then we will reduce Cyrus-Beck to more efficient Liang-Barsky case

The Cyrus-Beck Technique

- Cohen-Sutherland algorithm computes \((x, y)\) intersections of the line and clipping edge
- Cyrus-Beck finds a value of parameter \(t\) for intersections of the line and clipping edges
- Simple comparisons used to find actual intersection points
- Liang-Barsky optimizes it by examining \(t\) values as they generated to reject some line segments immediately

Finding the Intersection Points

\[
\begin{align*}
P(t) &= P_0 + t(P_1-P_0) \\
N \cdot [P_0-P_1] &= 0 \\
N \cdot [P_1-P_0] &= 0 \\
N \cdot [P_1-P_0] &= 0 \\
D &= \frac{[P_0-P_1]}{[N \cdot D]}
\end{align*}
\]

Make sure:
1. \(D = 0\) or \(P_1 = P_0\)
2. \(N \cdot D = 0\) lines are not parallel
Finding the Line Segment

- Classify point as potentially entering (PE) or leaving (PL)
- PE if crosses edge into inside half plane => angle \( P_0P_1 \) and \( N_i \) greater 90° => \( N_i \cdot D < 0 \)
- PL otherwise.
- Find \( T_e = \max(t_e) \)
- Find \( T_l = \min(t_l) \)
- Discard if \( T_e > T_l \)
  - If \( T_e < 0 \) \( T_e = 0 \)
  - If \( T_l > 1 \) \( T_l = 1 \)
- Use \( T_e, T_l \) to compute intersection coordinates \((x_e, y_e), (x_l, y_l)\)