CS-184: Computer Graphics

Lecture #9: Scan Conversion

Prof. James O’Brien
University of California, Berkeley
Today

- 2D Scan Conversion
  - Drawing Lines
  - Drawing Curves
  - Filled Polygons
  - Filling Algorithms
Drawing a Line

- Basically, it's easy... but for the details
- Lines are a basic primitive that needs to be done well...
Drawing a Line

- Basically, it's easy... but for the details
- Lines are a basic primitive that needs to be done well...

From "A Procedural Approach to Style for NPR Line Drawing from 3D models," by Grabli, Durand, Turquin, Sillion
Drawing a Line

\[ p_1 = (x_1, y_1) \]

\[ p_2 = (x_2, y_2) \]
Drawing a Line

\[ p_1 = (x_1, y_1) \]

\[ p_2 = (x_2, y_2) \]
Drawing a Line

- Some things to consider
  - How thick are lines?
  - How should they join up?
  - Which pixels are the right ones?

For example:
Drawing a Line

Inclusive Endpoints

$p_1 = (x_1, y_1)$

$p_2 = (x_2, y_2)$
Drawing a Line

\[ y = m \cdot x + b, x \in [x_1, x_2] \]

\[ m = \frac{y_2 - y_1}{x_2 - x_1} \]

\[ b = y_1 - m \cdot x_1 \]
### Drawing a Line

\[ \Delta x = 1 \]
\[ \Delta y = m \cdot \Delta x \]

\[ x = x_1 \]
\[ y = y_1 \]
while \( x \leq x_2 \)
  plot \( (x, y) \)
  \( x++ \)
  \( y += Dy \)
Drawing a Line

\[ \Delta x = 1 \]
\[ \Delta y = m \cdot \Delta x \]

After rounding
Drawing a Line

\[ \Delta x = 1 \]
\[ \Delta y = m \cdot \Delta x \]

\[ y_2 = y_1 + \Delta y \]

Accumulation of roundoff errors

How slow is float-to-int conversion?
Drawing a Line

\[ |m| \leq 1 \]

\[ |m| > 1 \]

\{ Gap \}
void drawLine_Error1(int x1, x2, int y1, y2)

  float m = float(y2 - y1) / (x2 - x1)
  int x = x1
  float y = y1

  while (x <= x2)
    
    setPixel(x, round(y), PIXEL_ON)
    x += 1
    y += m

Not exact math

Accumulates errors
void drawLine-Error2(int x1, x2, int y1, y2)

float m = float(y2-y1)/(x2-x1)
int x = x1
int y = y1
float e = 0.0

while (x <= x2)

  setPixel(x, \textcolor{red}{y}, PIXEL\_ON)

  x += 1
  e += m
  if (e >= 0.5)
    y+=1
    e-=1.0

No more rounding
void drawLine-Error3(int x1,x2, int y1,y2)

    int x = x1
    int y = y1
    float e = -0.5

    while (x <= x2)

        setPixel(x,y,PIXEL_ON)
        x += 1
        e += float(y2-y1)/(x2-x1)
        if (e >= 0.0)
            y+=1
            e-=1.0
void drawLine_Error4(int x1, x2, int y1, y2)

int x = x1
int y = y1
float e = -0.5*(x2-x1) // was -0.5

while (x <= x2)

  setPixel(x, y, PIXEL_ON)

  x += 1
  e += y2-y1 // was /(x2-x1)
  if (e >= 0.0) // no change
    y+=1
  e-=(x2-x1) // was 1.0
void drawLine-Error5(int x1, x2, int y1, y2)

    int x = x1
    int y = y1
    int e = -(x2-x1)              // removed *0.5

    while (x <= x2)

        setPixel(x,y,PIXEL_ON)

        x += 1
        e += 2*(y2-y1)              // added 2*
        if (e >= 0.0)               // no change
            y+=1
            e-=2*(x2-x1)              // added 2*
        

    // no change
void drawLine-Bresenham(int x1, x2, int y1, y2)

int x = x1
int y = y1
int e = -(x2-x1)

while (x <= x2)

  setPixel(x, y, PIXEL_ON)

  x += 1
  e += 2*(y2-y1)
  if (e >= 0.0)
    y += 1
    e -= 2*(x2-x1)

Faster
Not wrong

|m| ≤ 1
x₁ ≤ x₂
Drawing a Line

- How thick?

- Ends?

  - Butt
  - Round
  - Square
Drawing a Line

- Joining?

Ugly  Bevel  Round  Miter
Drawing Curves

\[ y = f(x) \]

Only one value of \( y \) for each value of \( x \)...

[Graph of a sine wave]
Drawing Curves

- **Parametric curves**
  - Both $x$ and $y$ are a function of some third parameter

\[
x = f(u) \\
y = f(u)
\]

\[
x = f(u) \\
u \in [u_0 \ldots u_1]
\]
Drawing Curves

\[ x = f(u) \quad u \in [u_0 \ldots u_1] \]
Drawing Curves

- Draw curves by drawing line segments
  - Must take care in computing end points for lines
  - How long should each line segment be?

\[ x = f(u) \quad u \in [u_0 \ldots u_1] \]
Drawing Curves

- Draw curves by drawing line segments
  - Must take care in computing end points for lines
  - How long should each line segment be?
  - Variable spaced points

\[ x = f(u) \quad u \in [u_0 \ldots u_1] \]
Drawing Curves

- Midpoint-test subdivision

\[ |f(u_{mid}) - l(0.5)| \]
Drawing Curves

- **Midpoint-test subdivision**

\[ |f(u_{mid}) - l(0.5)| \]
Drawing Curves

- Midpoint-test subdivision

\[ |f(u_{mid}) - l(0.5)| \]
Drawing Curves

- **Midpoint-test subdivision**
  - Not perfect
  - We need more information for a guarantee...

\[ |f(u_{mid}) - l(0.5)| \]
Filled Polygons
Filled Polygons
Filled Polygons

Toggle inside/outside flag to "INSIDE"
Filled Polygons

Toggle inside/outside flag to "OUTSIDE"
Filled Polygons

What happens at these locations?
Filled Polygons

If we count ONCE...
If we count TWICE...
Filled Polygons

Treat \((\text{scan } y = \text{vertex } y)\) as \((\text{scan } y > \text{vertex } y)\)
Filled Polygons

Horizontal edges
Filled Polygons

Horizontal edges
Filled Polygons

- “Equality Removal” applies to all vertices
- Both $x$ and $y$ coordinates
Filled Polygons

- Final result:
Filled Polygons

- Who does this pixel belong to?
Inside/Outside Testing

The Polygon

Non-exterior

Non-zero winding

Parity
Optimize for Triangles

- Split triangle into two parts
  - Two edges per part
  - Y-span is monotonic
- For each row
  - Interpolate span
- Interpolate barycentric coordinates
Flood Fill
Flood Fill

(a)

(b)

Start Position

Filled Pixel Spans

Stacked Positions

2
3
4
5
6
1
1
1
1
1