Chapter 7

Scan Conversion – Drawing on Raster Display (part 1 - Lines)
The Rendering Pipeline

1. Geometry Database
3. Lighting
4. Perspective Transform.
5. Clipping
6. Scan Conversion
7. Texturing
8. Depth Test
9. Blending
10. Frame-buffer
Scan Conversion - Rasterization

- Convert continuous rendering primitives into discrete fragments/pixels
  - Lines
    - Bresenham
  - Triangles
    - Flood Fill
    - Scanline
    - Implicit formulation
Scan Conversion - Lines

- Given segment equation fill in the pixels
- In drawings below - grid points = centers of pixels
Lines and Curves

- Explicit - one coordinate as function of the others
  \[ y = f(x) \]
  \[ z = f(x, y) \]

- line
  \[ y = mx + b \]
  \[ y = \frac{(y_2 - y_1)}{(x_2 - x_1)}(x - x_1) + y_1 \]

- circle
  \[ y = \pm \sqrt{r^2 - x^2} \]
Lines and Curves

- Parametric – all coordinates as functions of common parameter

\[
(x, y) = (f_1(t), f_2(t)) \\
(x, y, z) = (f_1(u, v), f_2(u, v), f_3(u, v))
\]

**line**

\[
\begin{align*}
x(t) &= x_1 + t(x_2 - x_1) \\
y(t) &= y_1 + t(y_2 - y_1)
\end{align*}
\]

\[t \in [0, 1]\]

**circle**

\[
\begin{align*}
x(\theta) &= r \cos(\theta) \\
y(\theta) &= r \sin(\theta)
\end{align*}
\]

\[\theta \in [0, 2\pi]\]
Lines and Curves

- **Implicit** - define as “zero set” of function of all the parameters

\[
\{(x, y) : F(x, y) = 0\}
\]

\[
\{(x, y, z) : F(x, y, z) = 0\}
\]

- Defines partition of space

\[
\{(x, y) : F(x, y) > 0\}, \{(x, y) : F(x, y) = 0\}, \{(x, y) : F(x, y) < 0\}
\]
Lines and Curves - Implicits

**line**

\[
\begin{align*}
dy &= y_2 - y_1 \\
dx &= x_2 - x_1 \\
F(x, y) &= (x - x_1)dy - (y - y_1)dx
\end{align*}
\]

- \(F(x, y) = 0\): \((x, y)\) is on line
- \(F(x, y) > 0\): \((x, y)\) is below line
- \(F(x, y) < 0\): \((x, y)\) is above line

**circle**

\[
F(x, y) = x^2 + y^2 - r^2
\]

- \(F(x, y) = 0\): \((x, y)\) is on circle
- \(F(x, y) > 0\): \((x, y)\) is outside
- \(F(x, y) < 0\): \((x, y)\) is inside

\[
F(x, y) = xdy - ydx + (y_1dx - x_1dy)
\]
Basic Line Drawing

Assume $x_1 < x_2$ & line slope absolute value is $\leq 1$

```
Line ( x1, y1, x2, y2 )
begin
  float dx, dy, x, y, slope ;
  dx <= x2 - x1;
  dy <= y2 - y1;
  slope <= dy / dx ;
  y <= y1
  for x from x1 to x2 do
    begin
      PlotPixel ( x, Round ( y ) ) ;
      y <= y + slope ;
    end ;
  end ;
end ;
```

Questions:
Can this algorithm use integer arithmetic?
Midpoint (Bresenham) Algorithm

**Assumptions:**

\[ x_2 > x_1, y_2 > y_1 \text{ and } \frac{dy}{dx} = \frac{y_2 - y_1}{x_2 - x_1} < 1 \]

**Idea:**

- Proceed along the line incrementally
- Have ONLY 2 choices
- Select one that minimizes error (distance to line)
Bresenham Algorithm

Distance (error):

\[ \tau = \{(x, y) \mid ax + by + c = xdy - ydx + c = 0\} \]

\[ d(x, y) = 2(xdy - ydx + c) \]

- Given point \( P = (x, y), d(x, y) \) is signed distance of \( P \) to \( \tau \) (up to normalization factor)
- \( d \) is zero for \( P \in \tau \)
Midpoint Line Drawing (cont’d)

- Starting point satisfies
  \[ d(x_1, y_1) = 0 \]
- Each step moves right (east) or upper right (northeast)
- Sign of \( d(x + 1; y + \frac{1}{2}) \) indicates if to move east or northeast
Midpoint Line Algorithm (version 1)

Line \((x_1, y_1, x_2, y_2)\)
begin
int \(x, y, dx, dy, d\);
\(x \leftarrow x_1;\)
\(y \leftarrow y_1;\)
\(dx \leftarrow x_2 - x_1;\)
\(dy \leftarrow y_2 - y_1;\)
PlotPixel \((x, y)\);
while \((x < x_2)\) do
\[d = (2x + 2)dy - (2y + 1)dx + 2c;\]
if \((d < 0)\) then
begin
\(x \leftarrow x + 1;\)
end;
else begin
\(x \leftarrow x + 1;\)
\(y \leftarrow y + 1;\)
end;
PlotPixel \((x, y)\);
end;
end;
Midpoint Line Drawing (cont’d)

- Insanely efficient version (less computations inside the loop)
  - compute \( d \) incrementally

At \((x_1, y_1)\)
\[ d_{\text{start}} = d(x_1 + 1, y_1 + \frac{1}{2}) = 2dy - dx \]

Increment in \( d \) (after each step)

- If move east \[ \Delta_e = d(x + 2, y + \frac{1}{2}) - d(x + 1, y + \frac{1}{2}) = 2((x+2)dy - (y+\frac{1}{2})dx + c)) - 2((x+1)dy - (y+\frac{1}{2})dx + c) = 2dy \]

- If move northeast \[ \Delta_{ne} = d(x_1 + 2, y_1 + \frac{3}{2}) - d(x_1 + 1, y_1 + \frac{1}{2}) = 2((x + 2)dy - (y + \frac{3}{2})dx + c)) - 2((x + 1)dy - (y + \frac{1}{2})dx + c) = 2(dy - dx) \]
**Midpoint Line Algorithm**

**Line** \(( x_1, y_1, x_2, y_2 )\)

begin

int \( x, y, dx, dy, d, \Delta_e, \Delta_{ne} \);

\( x \leftarrow x_1 ; \quad y \leftarrow y_1 ; \)

\( dx \leftarrow x_2 - x_1 ; \quad dy \leftarrow y_2 - y_1 ; \)

\( d \leftarrow 2 \times dy - dx ; \)

\( \Delta_e \leftarrow 2 \times dy ; \quad \Delta_{ne} \leftarrow 2 \times (dy - dx) ; \)

**PlotPixel** \(( x, y )\);

while \(( x < x_2 )\) do

if \(( d < 0 )\) then

begin

\( d \leftarrow d + \Delta_e ; \)

\( x \leftarrow x + 1 ; \)

end ;

else begin

\( d \leftarrow d + \Delta_{ne} ; \)

\( x \leftarrow x + 1 ; \)

\( y \leftarrow y + 1 ; \)

end ;

**PlotPixel** \(( x, y )\);

end ;

end ;
Midpoint Examples

- Question: Is there a problem with this algorithm (horizontal vs. diagonal lines)?

- Comment: extends to higher order curves – e.g. circles
Error Function Intuition

- Error function $d$ can be viewed as explicit surface:

$$d(x,y) = 2(xdy - ydx + c)$$