Chapter 7

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

1. Geometry Database
3. Lighting
4. Perspective Transform.
5. Clipping

- Scan Conversion
- Texturing
- Depth Test
- Blending
- Framebuffer
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**
\[ x(t) = x_1 + t(x_2 - x_1) \]
\[ y(t) = y_1 + t(y_2 - y_1) \]
\[ t \in [0, 1] \]

**circle**
\[ x(\theta) = r \cos(\theta) \]
\[ y(\theta) = r \sin(\theta) \]
\[ \theta \in [0, 2\pi] \]
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

<table>
<thead>
<tr>
<th>line</th>
<th>circle</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ dy = y_2 - y_1 ]</td>
<td>[ F(x, y) = (x - x_1) dy - (y - y_1) dx ]</td>
</tr>
<tr>
<td>[ dx = x_2 - x_1 ]</td>
<td>[ F(x, y) = x^2 + y^2 - r^2 ]</td>
</tr>
<tr>
<td>[ F(x, y) = 0 ] (x,y) is on line</td>
<td>[ F(x, y) = 0 ] (x,y) is on circle</td>
</tr>
<tr>
<td>[ F(x, y) &gt; 0 ] (x,y) is below line</td>
<td>[ F(x, y) &gt; 0 ] (x,y) is outside</td>
</tr>
<tr>
<td>[ F(x, y) &lt; 0 ] (x,y) is above line</td>
<td>[ F(x, y) &lt; 0 ] (x,y) is inside</td>
</tr>
<tr>
<td>[ F(x, y) = xdy - ydx + (y_1dx - x_1dy) ]</td>
<td></td>
</tr>
</tbody>
</table>
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;
```

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)
Bresenhan Algorithm

**Distance (error):**

\[
\tau = \{ (x, y) | ax + by + c = xdy - ydx + c = 0 \} \\
d(x, y) = 2(xdy - ydx + c)
\]

- 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)

```c
Line (x₁, y₁, x₂, y₂)
begin
  int x, y, dx, dy, d;
  x ← x₁;
  y ← y₁;
  dx ← x₂ - x₁;
  dy ← y₂ - y₁;
PlotPixel (x, y);
while (x < x₂) do
  d = (2x + 2)dy - (2y + 1)dx + 2c; // 2((x + 1)dy - (y + .5)dx + c)
  if (d < 0) then
    begin
      x ← x + 1;
    end;
  else begin
    x ← x + 1;
    y ← y + 1;
  end;
PlotPixel (x, y);
end;
end;
```

bresenham
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

```c
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 * dy - dx ;
\Delta_e \Leftarrow 2 * dy ; \quad \Delta_{ne} \Leftarrow 2 * (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:

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