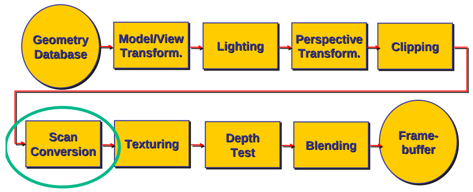
University of British Columbia

Chapter 7

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

The Rendering Pipeline

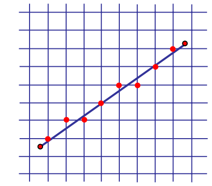


Scan Conversion - Rasterization

- 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

line

$dy = y_2 - y_1$

$dx = x_2 - x_1$

$F(x,y) = (x - x_1)dy - (y - y_1)dx$

$F(x,y) = 0$ **(x,y) is on line**

$F(x,y) > 0$ **(x,y) is below line**

$F(x,y) < 0$ **(x,y) is above line**

$F(x,y) = xdy - ydx + (y_1 dx - x_1 dy)$

circle

$F(x,y) = x^2 + y^2 - r^2$

$F(x,y) = 0$ **(x,y) is on circle**

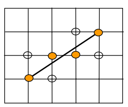
$F(x,y) > 0$ **(x,y) is outside**

$F(x,y) < 0$ **(x,y) is inside**

Basic Line Drawing

Assume $x_1 < x_2$ & line slope absolute value is ≤ 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 ;
```

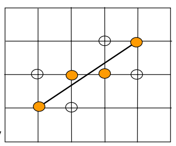


Questions:
Can this algorithm use integer arithmetic ?

Midpoint (Bresenham) Algorithm

- **Assumptions:**

$$x_2 > x_1, y_2 > y_1 \; and \; \frac{dy}{dx} = \frac{y_2 - y_1}{x_2 - x_1} < 1$$

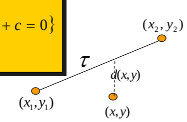


- **Idea:**
  - Proceed along the line incrementally
  - Have ONLY 2 choices
  - Select one that minimizes error (distance to line)

Bresenham Algorithm

Distance (error):

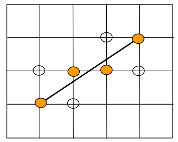
$$\tau = \{(x,y) | ax + by + c = xdy - ydx + c = 0\}$$
$$d(x,y) = 2(xdy - ydx + c)$$

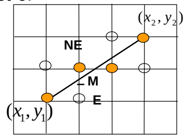


- Given point $P = (x,y)$, $d(x,y)$ is signed distance of p to τ (up to normalization factor)
- d is zero for $P \in \tau$

Midpoint Line Drawing (cont'd)

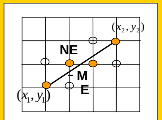
- Starting point satisfies $d(x_1, y_1) = 0$
- Each step moves right (east) or upper right (northeast)
- Sign of $d(x+1; y+\frac{1}{2})$ indicates if to move east or northeast





Midpoint Line Algorithm (version 1)

```
Line (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 ;
```

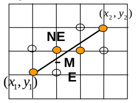


bresenham

Midpoint Line Drawing (cont'd)

- Insanely efficient version (less computations inside the loop)
  – compute d incrementally
- At (x_1, y_1)

$$d_{start} = d(x_1 + 1, y_1 + \tfrac{1}{2}) = 2dy - dx$$



- Increment in *d* (after each step)
  - If move east $\Delta_e = d(x+2, y+\frac{1}{2}) - d(x+1, y+\frac{1}{2}) =$

$$2((x+2)dy - (y+\tfrac{1}{2})dx + c)) - 2((x+1)dy - (y+\tfrac{1}{2})dx + c) = 2dy$$

  - If move northeast $\Delta_{ne} = d(x_1+2, y_1+\frac{3}{2}) - d(x_1+1, y_1+\frac{1}{2}) =$

$$2((x+2)dy - (y+\tfrac{3}{2})dx + c)) - 2((x+1)dy - (y+\tfrac{1}{2})dx + c) = 2(dy - dx)$$

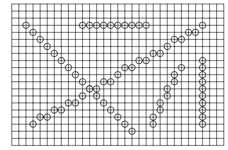
Midpoint Line Algorithm

```
Line (x1, y1, x2, y2)
begin
int x, y, dx, dy, d, Δe, Δne ;
x ⇐ x1 ;          y ⇐ y1 ;
dx ⇐ x2 − x1 ;    dy ⇐ y2 − y1 ;
d ⇐ 2 * dy − dx ;
Δe ⇐ 2 * dy ;     Δne ⇐ 2 * (dy − dx) ;
PlotPixel ( x, y ) ;
while ( x < x2 ) do
    if ( d < 0 ) then
    begin
        d ⇐ d + Δe ;
        x ⇐ x + 1 ;
    end ;
    else begin
        d ⇐ d + Δne ;
        x ⇐ x + 1 ;
        y ⇐ y + 1 ;
    end ;
    PlotPixel ( x, y ) ;
end ;
end ;
```

bresenham

Midpoint Examples

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



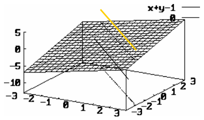
- Comment: extends to higher order curves – e.g. circles

fineline

Error Function Intuition

- Error function d can be viewed as explicit surface:

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



University of British Columbia