

Raster Algorithms

Overview

- Drawing Lines and Circles
- Filling Algorithms
 - ⇒ Boundary, Floodfill
 - ⇒ Scanline Algorithm
- Clipping Algorithms
 - ⇒ Lines - Cohen-Sutherland, Liang-Barsky Algorithm
 - ⇒ Polygon - Sutherland-Hodgman
- Anti-aliasing

Raster Algorithms

“The process of converting geometric primitives into their discrete approximations”

Scan Conversion:

⇒ Approximate geometric primitives (analytically defined) by a set of pixels, stored in frame-buffer or memory.



Clipping:

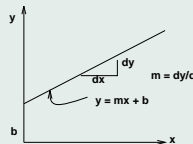
⇒ Only sections of primitives determined to be within a clipping region is drawn (scan-converted).

Speed:

⇒ Algorithms must be very efficient. Why?

Drawing Lines

Given the line end points (x_0, y_0) and (x_1, y_1)



Line Equation

$$y = mx + b$$

m = Slope, b = y intercept

To determine

the sequence of points between (x_0, y_0) and (x_1, y_1) on the raster grid (end points are in screen coordinates).

Brute Force Algorithm

```
for  $i = x_0$  to  $i = x_1$ 
{
     $y_i = mx_i + b$ 
    PLOT ( $x_i$ , ROUND ( $y_i$ ))
}
```

Inefficient: Involves multiply and rounding.

DDA Algorithm

Features

- Exploits the fact that the line equation is a linear function that needs to be evaluated over a regular lattice.
- Constant** increments in both dimensions to obtain successive points along the line eliminates multiplies in the inner loop.

$$\begin{aligned}
 y_{i+1} &= mx_{i+1} + b \\
 &= m(x_i + \Delta x) + b \\
 &= (mx_i + b) + m\Delta x \\
 &= y_i + m(1)
 \end{aligned}$$

DDA Algorithm

Algorithm:

```

m = (y1 - y0)/(x1 - x0)
for xi = x0 to xi = x1
{
    plot ( xi, Round(yi))
    yi = yi + m
    xi = xi + 1
}
    
```

Inefficient, involves division to compute slope, and rounding.

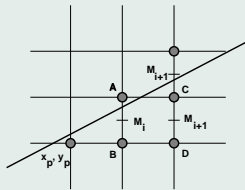
MidPoint Line Algorithm

For lines and circles, same as **Bresenham's** algorithm.

Features:

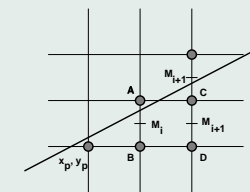
- Uses only integer operations.
- Uses incremental calculations to determine successive pixels.

Idea



- Determine location of mid point, M_i with respect to the line.
- Mid points M_i are computed using incremental calculations.

MidPoint Line Algorithm(contd)

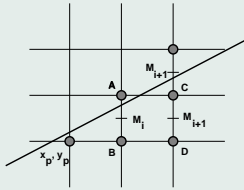


$$\begin{aligned}
 y &= (dy/dx).x + B \\
 dx.y &= dy.x + B.dx, \text{ or}
 \end{aligned}$$

$$\begin{aligned}
 F(x, y) &= dy.x - dx.y + B.dx = 0 \\
 &= a.x + b.y + c = 0, \quad (a = dy, b = -dx, c = B .dx)
 \end{aligned}$$

$$\begin{aligned}
 d = F(x, y) &= 0, & x, y &\text{ on the line} \\
 &> 0, & x, y &\text{ below the line} \\
 &< 0, & x, y &\text{ above the line}
 \end{aligned}$$

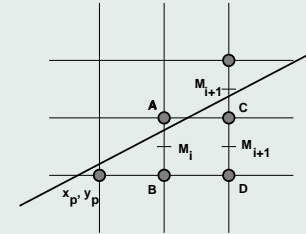
MidPoint Line Algorithm(contd)



Strategy:

- The sign of $d = F(x, y)$, the **decision variable**, will determine whether A or B is chosen as the next pixel on the line.
- “Insert M_i into $F(x, y)$ and check the sign of F .”

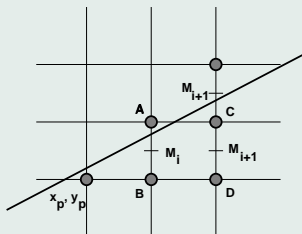
Efficient calculation of d



Case 1: A is chosen

$$\begin{aligned}
 M_i &= F(x_p + 1, y_p + 1/2) \\
 M_{i+1} &= F(x_p + 2, y_p + 3/2) \\
 M_{i+1} - M_i &= F(x_p + 2, y_p + 3/2) - F(x_p + 1, y_p + 1/2) \\
 &= \{a(x_p + 2) + b(y_p + 3/2) + c\} - \\
 &\quad \{a(x_p + 1) + b(y_p + 1/2) + c\} \\
 &= (a + b) \\
 M_{i+1} &= M_i + (a + b) = M_i + dy - dx
 \end{aligned}$$

Efficient calculation of d (contd)



Case 2: B is chosen

$$\begin{aligned}
 M_{i+1} &= F(x_p + 2, y_p + 1/2) \\
 M_{i+1} - M_i &= F(x_p + 2, y_p + 1/2) - F(x_p + 1, y_p + 1/2) \\
 &= a \\
 M_{i+1} &= M_i + (a) \\
 &= M_i + dy
 \end{aligned}$$

Initialization

$$\begin{aligned}
 d &= F(x_0 + 1, y_0 + 1/2) \\
 &= a(x_0 + 1) + b(y_0 + 1/2) + c \\
 &= (ax_0 + by_0 + c) + a + b/2 \\
 &= a + b/2
 \end{aligned}$$

or

$$d = 2a + b \text{ (only sign of } d \text{ is important)}$$

Hence

$$\begin{aligned}
 d_1 &= 2(a + b) = 2(dy - dx) \\
 d_2 &= 2a = 2(dy)
 \end{aligned}$$

Final Algorithm (Quadrant 1 only)

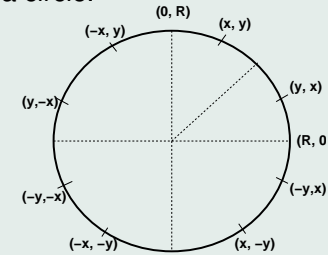
$d = 2 dy - dx$
 $d_1 = 2 dy - 2 dx$
 $d_2 = 2 dy$

```

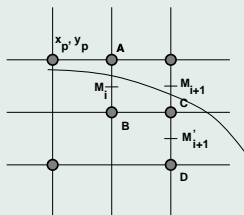
for  $x = x_0$  to  $x_1$  by 1
{
  if ( $d \leq 0$ )
     $d = d + d_1$ 
  else
    {
       $d = d + d_2$ 
       $y = y + 1$ 
    }
  plot ( $x, y$ )
}
    
```

Drawing Circles

Rotational symmetry makes it sufficient to scanconvert only 1 octant of a circle.



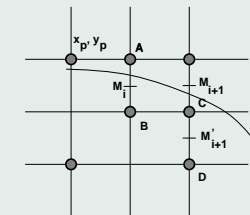
MidPoint Circle Algorithm



$$F(x, y) = x^2 + y^2 - R^2 = 0 \quad (\text{circle centered at the origin})$$

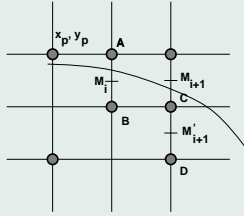
$d = F(x, y) = 0,$ x, y **on** the circle
 $> 0,$ x, y **outside** the circle
 $< 0,$ x, y **inside** the circle

MidPoint Circle Algorithm(contd)



$$\begin{aligned}
 F(M_i) &= F(x_p + 1, y_p - 1/2) \\
 &= (x_p + 1)^2 + (y_p - 1/2)^2 - R^2 \\
 F(M_{i+1}) &= F(x_p + 2, y_p - 1/2) \dots (\text{East Neighbor}) \\
 &= (x_p + 2)^2 + (y_p - 1/2)^2 - R^2 \\
 F(M'_{i+1}) &= F(x_p + 2, y_p - 3/2) \dots (\text{S. East Neighbor}) \\
 &= (x_p + 2)^2 + (y_p - 3/2)^2 - R^2
 \end{aligned}$$

MidPoint Circle Algorithm(contd)



$$\begin{aligned}
 d_{old} &= F(M_i) \\
 d_{new} &= F(M_{i+1}), \text{ A chosen} \\
 &= F(M'_{i+1}), \text{ B chosen} \\
 d_{incr} &= d_{new} - d_{old} \\
 &= 2x_p + 3, \text{ A chosen.} \\
 &= 2(x_p - y_p) + 5, \text{ B chosen.}
 \end{aligned}$$

Midpoint Circle Alg.:Initialization

$$\begin{aligned}
 (x_0, y_0) &= (0, R) \\
 F(1, R - 1/2) &= 1 + R^2 + 1/4 - R - R^2 \\
 &= 5/4 - R
 \end{aligned}$$

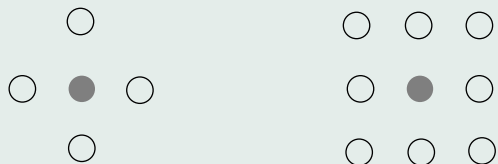
Efficiency

The algorithm can be made more efficient (Refer Foley/van Dam).

Fill Algorithms

4-Connected

8-Connected



4 - Connected

8 Connector

FloodFill Algorithm

Interior pixels are defined by a unique color.

FloodFill (x, y, InteriorColor, NewColor)

```

{
  if pixel_color (x, y) EQUALS InteriorColor
  {
    set_pixel (x, y, NewColor)
    FloodFill (x-1, y, InteriorColor, NewColor)
    FloodFill (x+1, y, InteriorColor, NewColor)
    FloodFill (x, y+1, InteriorColor, NewColor)
    FloodFill (x, y-1, InteriorColor, NewColor)
  }
}
    
```

Boundary Fill Algorithm

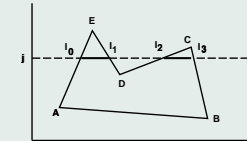
Boundary pixels are defined by a unique color.

```
BoundaryFill (x, y, BoundaryColor, NewColor)
{
  if (pixel_color (x, y) NOT_EQUAL_TO BoundaryColor) AND
    (pixel_color (x, y) NOT_EQUAL_TO NewColor)
  {
    set_pixel (x, y, NewColor)
    BoundaryFill (x-1, y, BoundaryColor, NewColor)
    BoundaryFill (x+1, y, BoundaryColor, NewColor)
    BoundaryFill (x, y+1, BoundaryColor, NewColor)
    BoundaryFill (x, y-1, BoundaryColor, NewColor)
  }
}
```

The Scanline Polygon Fill Algorithm

Approach

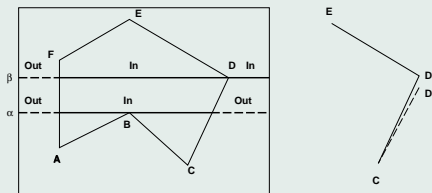
- Exploit the geometry of the polygon.
- Use scanline and edge coherence for efficient filling.



Algorithm

```
for each scanline
{
  Calculate Intersections between scanline and all polygon edges
  Sort Intersections on X.
  Fill pixels between pairs of intersection points.
}
```

The Scanline Polygon Fill Algorithm: Special Cases



1. *Horizontal Edges*: Discard all horizontal edges

2. *Scanline passing through a vertex*:

- scanline α : no problem, scanline β : wrong results
- *Solution*: check for local extrema (in Y)
- if local maxima/minima count **two** intersections
else count **one** intersection
- *Implementation*: if vertex does not conform to a local extrema,
shorten edge by 1 pixel in Y.

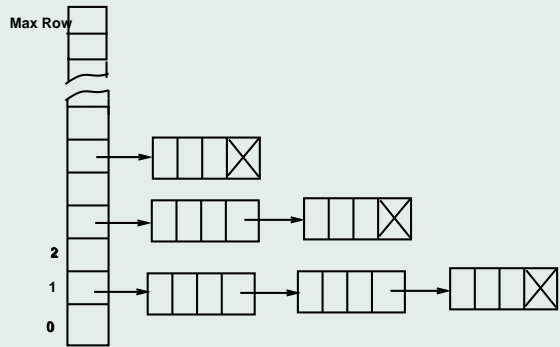
Improving Performance

- Use edge and scanline coherence.
- Implemented using an Edge Table (ET) and an Active Edge Table (AET)

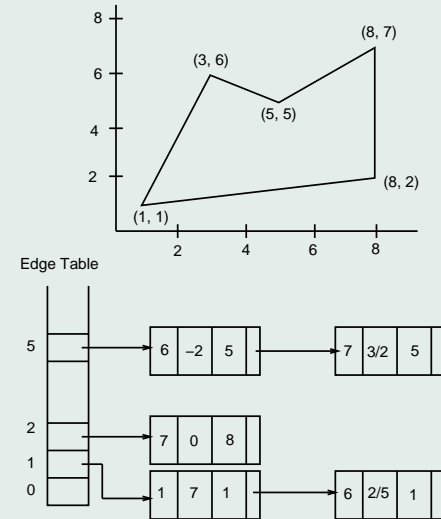
Edge Record

```
{
  Integer y_upper /* y coordinate at upper end point */
  Integer x_int /* x coordinate of intersection with edge */
  Float recip_slope /* 1/m */
  Pointer next; /* to next edge record */
}
```

Edge Table



Example



Algorithm

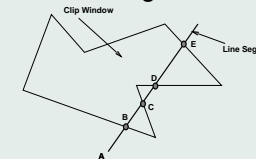
```

Initialize          /* take care of special cases */
AET = NULL
for y = min_row to max_row
{
  add all edge records in ET[y] to AET
  if ( AET NOT_EQUAL_TO NULL )
  {
    Sort AET on x /* x coordinates of intersection points */
    Fill pixel runs.
    Delete edge records for whom y = y_upper
    Update x_int values by adding 1/m
  }
}
  
```

Line Clipping

Problem Definition

Given a clip region and a line segment, determine the sections of the line interior to the region.



Simplification

Clip region (window) is an upright rectangle.

Line Clipping Alg: Strategy

```

if end points are inside clip region
    entire line segment is visible
else
    calculate intersections between line segment and clip region
    output segments interior to clip region
end
    
```

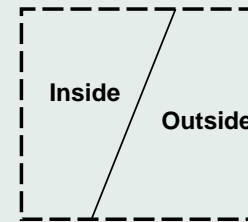
Note:

- we deal with **line segments**, not infinite lines.
- Slope-Intercept form ($y = mx + c$) of line equation is not convenient
- **Parametric** representation preferred.

Cohen-Sutherland Clipping Algorithm

Idea:

Consider clip region to be the **intersection** of half-spaces.



- Define **In** and **Out** half-spaces for each boundary segment of the clip window.
- clip region = **Intersection** of all “In” spaces.
- Each In/Out region is represented by a 1 bit code.

Cohen-Sutherland Clipping Algorithm (contd.)

Given line end points: $(x_0, y_0), (x_1, y_1)$, and clip Boundaries:
 $x_{min}, x_{max}, y_{min}, y_{max}$

1001	0001	0101
1000	0000	0100
1010	0010	0110

○	○	○	○
L	R	B	T

Encoding a point:

```

code0[0] =  $x_0 < x_{min}$ 
code0[1] =  $x_0 > x_{max}$ 
code0[2] =  $y_0 < y_{min}$ 
code0[3] =  $y_0 > y_{max}$ 
    
```

and similarly for (x_1, y_1)

Algorithm

Calculate codes of both end points.

done = FALSE

while not done

{

if (all 4 codes of both endpoints equal zero)

 “**Accept** line segment”, done = TRUE

else if (bit-wise logical AND of code0 and code1 is non-zero)

 “**Reject** line segment”, done = TRUE

else

 {

 choose end point P outside clip window

 Intersect line segment with clip window

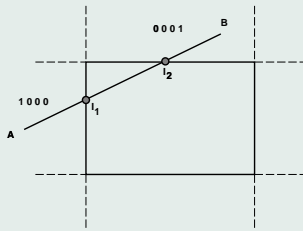
 Discard line segment from P to intersection point

 compute codes of the new (clipped) line segment

 }

}

Example



Trivial Accept: No!

Trivial Reject: $(1000 \& 0001) = 0000$. NO!

Pick end point with non-zero code - A (can also pick B)

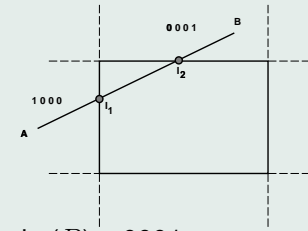
Intersect with left boundary (corresponding to the first non-zero bit)

$$\frac{t}{1.0} = \frac{x_{min} - x_0}{x_1 - x_0}$$

$$y = y_0 + t(y_1 - y_0) = y_0 + \frac{x_{min} - x_0}{x_1 - x_0}(y_1 - y_0)$$

$$\text{Intersection } I_1 = (x_{min}, y), \quad x_0 = x_{min}, y_0 = y$$

Example (contd.)



Code $(I_1) = 0000$, Code $(I_2) = 0001$

Cannot trivially accept or reject, choose end point B

Intersect with top boundary, $y = y_{max}$

$$\frac{t}{1.0} = \frac{y_{max} - y_1}{y_0 - y_1}$$

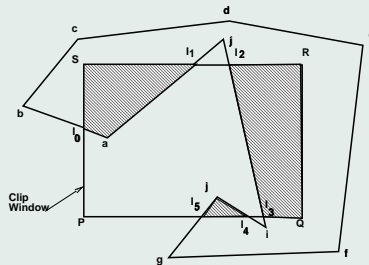
$$x = x_1 + t(x_0 - x_1) = x_1 + \frac{y_{max} - y_1}{y_0 - y_1}(x_0 - x_1)$$

$$\text{Intersection } I_2 = (x, y_{max})$$

Calculate codes of I_1 and I_2

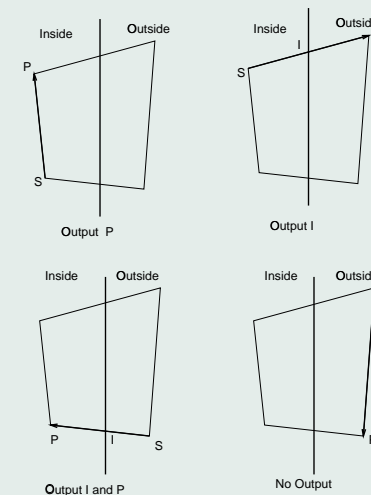
Trivially accept the segment (I_1, I_2)

Polygon Clipping



Using a Line Clipper

- Apply Cohen Sutherland algorithm to each segment of polygon
- Insufficient, as polygons can become fragmented after clipping
- Polygons must remain polygons after clipping



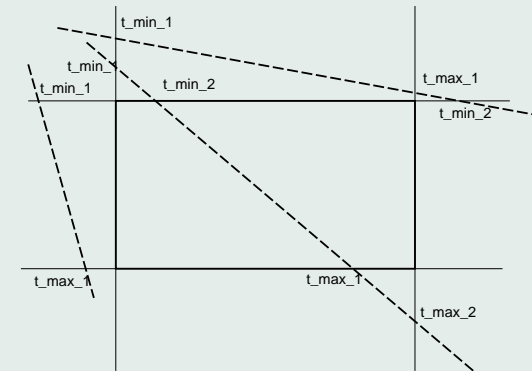
Liang-Barsky Clipping Algorithm

- A special case of the Cyrus-Beck clipping algorithm.

Features

- In general, more efficient than the Cohen-Sutherland algorithm.
- Most of the operations are performed in parametric space.

Liang-Barsky Clipping: Idea



- Intersections between a line and the clipping boundaries are distinguished as entering or exiting the boundary (t_{enter} and t_{exit}).
- If $Max(t_{enter_k}) < Min(t_{exit_k})$ and the interval is a subset of $(0, 1.0)$, then the line segment intersects the interior of the clipping region.

Liang-Barsky Clipping: Details

A point (x, y) is in the interior if

$$\begin{aligned} xw_{min} &\leq x_0 + t\Delta x \leq xw_{max} \\ yw_{min} &\leq y_0 + t\Delta y \leq yw_{max} \end{aligned}$$

which can be rewritten as

$$tp_k \leq q_k, \quad k = 1, 2, 3, 4$$

$$\begin{aligned} p_1 &= -\Delta x, & q_1 &= x_0 - xw_{min} \\ p_2 &= \Delta x, & q_2 &= xw_{max} - x_0 \\ p_3 &= -\Delta y, & q_3 &= y_0 - yw_{min} \\ p_4 &= \Delta y, & q_4 &= yw_{max} - y_0 \end{aligned}$$

Liang-Barsky Clipping: Details

Intersection

$$t = \frac{q_k}{p_k}$$

- $p_k = 0$: line parallel to the k th clipping boundary; if $q_k < 0.0$, line is trivially rejected.
- $p_k < 0$: line proceeds from outside to inside across clipping boundary.
- $p_k > 0$: line proceeds from inside to outside across clipping boundary.

Algorithm

```
 $t_{min} = 0.0, t_{max} = 1.0$   
if ClipBoundary ( $p_1, q_1, \&t_{min}, \&t_{max}$ )  
  if ClipBoundary ( $p_2, q_2, \&t_{min}, \&t_{max}$ )  
    if ClipBoundary ( $p_3, q_3, \&t_{min}, \&t_{max}$ )  
      if ClipBoundary ( $p_4, q_4, \&t_{min}, \&t_{max}$ )  
        {  
          if ( $t_{min} > 0.0$ ) {  
             $pt1.x = pt1.x + t_{min} * dx$   
             $pt1.y = pt1.y + t_{min} * dy$   
          }  
          if ( $t_{min} < 1.0$ ) {  
             $pt2.x = pt2.x + t_{max} * dx$   
             $pt2.y = pt2.y + t_{max} * dy$   
          }  
        }  
      }  
    }  
  }  
} "draw line from pt1 to p2"
```