Florida Tech

CSE 5255

INTRODUCTION TO COMPUTER GRAPHICS CLASS NOTES

Dr. William D. Shoaff

Spring 1996

Table of Contents

- Scan Conversion — the process of converting a vertex representation of data into a pixel representation
- Digital Differential Analyzer (DDA) for lines
- Bresenham's Algorithms for lines and circles
- Flood fill for polygons
- Boundary fill for polygons
- Scan line fill for polygons
- Many of the scan conversion algorithms use *incremental* methods that exploit *coherence*
  - An incremental method computes a new value from an old value (this is usually much faster than computing the new value from scratch)
  - Coherence (in space and/or time) implies nearby objects (e.g. pixels) have qualities similar to the current object

The DDA Algorithm

- Let $p = (p_1, p_2)$ and $q = (q_1, q_2)$ be two endpoints of a line (in device coordinate, so p_1, p_2, q_1, q_2 are integers)
- The point-intercept form of the equation of the line from p to q is

  $$y = mx + b$$

  where

  $$m = \frac{q_2 - p_2}{q_1 - p_1}$$

  and (although it is not really used here)

  $$b = p_2 - mp_1$$

- Notice that if x is changed to $x + 1$ then y changes to $y + m$
- Notice that if y is changed to $y + 1$ then x changes to $x + \frac{1}{m}$
- The slope (and its reciprocal) often won't be integers

- Case 1: $0 \leq \mid m \mid \leq 1$
  - With little lost of generality, assume $p_1 < q_1$
  - If $(x,\ y)$ is on the line, then so is $(x+1, y+m)$
  - The pseudocode for the algorithm is:

$x = p_1$;

$y = p_2$;

$plot\ (x,\ y)$;

while $(x < q_1)$ $\{$

  $x = x + 1$;

  $y = y + m$;

  $plot\ (x,\ round(y))$;

$\}$

- Case 2: $\mid m \mid > 1$
  - interchange roles of x and y
  - starting from the lowest y value, increment y by 1 and increment x by $1/m$

DDA Example

- Given end-points $p = (1, 3)$ and $q = (8, 9)$, find the pixels illuminated by the DDA algorithm
- The slope is $m = 6/7$
- x is incremented by 1 and y by m at each step
- The first point is $(1, 3)$
- The next point on the line is $(2, 27/7)$
- The rounded pixel value is $(2, 4)$
- The plotted pixels are:

Step	y	pixel
1	$21/7$	$(1, 3)$
2	$27/7$	$(2, 4)$
3	$33/7$	$(3, 5)$
4	$39/7$	$(4, 6)$
5	$45/7$	$(5, 6)$
6	$51/7$	$(6, 7)$
7	$57/7$	$(7, 8)$
8	$63/7$	$(8, 9)$

- What pixels illuminated by the DDA algorithm given end-points $p = (1, 3)$ and $q = (8, 11)$?

- DDA algorithm requires floating point arithmetic and rounding inside the main loop
- Bresenham's algorithm works solely with integers
- Let $p = (p_1, p_2)$ and $q = (q_1, q_2)$ be two endpoints of a line (in device coordinate, so p_1, p_2, q_1, q_2 are integers)
- The implicit equation of the line from p to q is

$$Ax + By + C = 0$$

where

$$A = (q_2 - p_2) = \triangle y,$$
$$B = -(q_1 - p_1) = -\triangle x,$$

and

$$C = -(Ap_1 + Bp_2)$$

- Assume that $p_1 < q_1$ and the slope is between 0 and 1, so that

$$0 \leq A \leq -B$$

- Define the *error* $e = Ax + By + C$
- Since slope is between 0 and 1, and since $p_1 < q_1$, the only two valid moves are

  $$\text{Right (R): } x \to x + 1$$

  or

  $$\text{Diagonally up (D): } x \to x + 1,\ y \to y + 1$$

- Consider an R move, the error becomes

  $$e_R = A(x + 1) + By + C = e + A$$

- Consider a D move, the error becomes

  $$e_D = A(x + 1) + B(y + 1) + C = e + A + B$$

- Move in whichever direction has smaller error
- If $\mid e_R \mid < \mid e_D \mid$ move to right
- If $\mid e_D \mid \leq \mid e_D \mid$ move to diagonally up
- The sign of the difference

  $$\mid e_R \mid - \mid e_D \mid$$

  can be used to judge which is smaller
- The errors e_R, e_D can be either non-negative $(+)$ or negative $(-)$. There are four possibilities

e_R	e_D	$\mid e_R \mid - \mid e_D \mid$	next pixel
$+$	$+$	$-B > 0$	D
$+$	$-$	$e_r + e_D = A + B$	don't know
$-$	$+$	can't happen	
$-$	$-$	$B < 0$	R

- Use the indeterminate case to define a *biased error*

$$g = e_R + e_D = 2e + 2A + B$$

- If e_R and e_D are both non-negative, so is g (move D)
- If e_R and e_D are both negative, so is g (move R)
- Indeterminate case, if g is non-negative, move D; if g is negative, move R
- Biased error update for move D:

$$g_D = g + 2A + 2B$$

- Biased error update for move R:

$$g_R = g + 2A$$

- Initial value of $g = 2A + B$

Pseudocode for Bresenham's Algorithm

- Let $p = (p_1, p_2)$ and $q = (q_1, q_2)$ be line end-points
- Assume slope of line is between 0 and 1 and $p_1 < q_1$

$A = q_2 - p_2$;
$B = p_1 - q_1$;
$g = 2 * A + B$;
$diag\_inc = 2 * (A + B)$;
$right\_inc = 2 * A$;
$x = p_1$;
$y = p_2$;
while $(x \leq q_1)$ {
  $plot\ (x, y)$;
  if $(g \geq 0)$ {
    $y = y + 1$;
    $g = g + diag\_inc$;
  }
  else $g = g + right\_inc$;
  $x = x + 1$;
}

Bresenham Example

- Given end-points $p = (1, 3)$ and $q = (8, 9)$, find the pixels illuminated by Bresenham's algorithm
- $A = 9 - 3 = 6$, $B = -(8 - 1) = -7$
- Initial biased error $g = 2A + B = 5$
- Biased error update for diagonal move

$$diag\_inc = 2(A + B) = -2$$

- Biased error update for right move

$$right\_inc = 2A = 12$$

Step	old g	pixel	new g
1	5	$(1, 3)$	3
2	3	$(2, 4)$	1
3	1	$(3, 5)$	-1
4	-1	$(4, 6)$	11
5	11	$(5, 6)$	9
6	9	$(6, 7)$	7
7	7	$(7, 8)$	5
8	5	$(8, 9)$	3

Bresenham's Circle Algorithm

- Let $x^2 + y^2 - r_2 = 0$ be the equation of a circle centered at the origin with integer radius r
- Starting at the top of the circle $(0, r)$, we want to determine which pixels to illuminate in the clockwise direction until we hit the line $x = y$
- The remainder of the circle can be determined by symmetry
- At each step we choose to move either to the right (R), $x = x + 1$, or diagonally down (D), $x = x + 1$, $y = y - 1$
- Define the error to be

$$e = x^2 + y^2 - r^2$$

- If we move right, then the error will become

$$e_R = (x+1)^2 + y^2 - r^2 = e + (2x + 1)$$

- If we move diagonally down, then the error will become

$$e_D = (x+1)^2 + (y-1)^2 - r^2 = e + (2x + 1) - (2y - 1)$$

- The errors e_R, e_D can be either non-negative $(+)$ or negative $(-)$ (four possibilities)

e_R	e_D	$\mid e_R \mid - \mid e_D \mid$	next pixel
$+$	$+$	$2y - 1 > 0$	D
$+$	$-$	$e_r + e_D = A + B$	don't know
$-$	$+$	can't happen	
$-$	$-$	$-(2y - 1) < 0$	R

- The "biased" error
  $g = e_R + e_D = 2e + (4x + 2) - (2y - 1)$ is used in all three cases
- If e_R and e_D are both non-negative, so is g (move D)
- If e_R and e_D are both negative, so is g (move R)
- In indeterminate case, if g is non-negative, move D; if g is negative, move R
- Initial value for $g = 3 - 2r$
- Biased error update for move D:

$$g_D = g + 4x - 4y + 10$$

- Biased error update for move R:

$$g_R = g + 4x + 6$$

Pseudocode for Bresenham's Circle Algorithm

```
x = 0;
y = r;
g = 3 − 2 * r;
diag_inc = 10 − 4 * r;
right_inc = 6;
while (x ≤ y) {
   sym_plot (x, y);
   if (g ≥ 0) {
      g = g + diag_inc;
      diag_inc = diag_inc + 8;
      y = y − 1;
   }
   else {
      g = g + right_inc;
      diag_inc = diag_inc + 4;
   }
   right_inc = right_inc + 4;
   x = x + 1;
}
```

- Note $sym\_plot(x, y)$ calls $plot$ with all 8 values $(\pm x, \pm y), (\pm y, \pm x)$

Polygon Filling Algorithms

- Scan conversion algorithms that generate all pixels in the interior of a polygon
- The polygon can be filled with a color or pattern
- There are two general classes of polygon filling algorithms:
  - Fill algorithms that spread out from a *seed*
  - Scan-line algorithms which fill a polygon one scan line at time
- Flood fill algorithm color all pixels not equal to a given boundary color
- Boundary fill algorithm color all pixels interior to a region
- Fill algorithms are common in "paint" programs
- Scan line algorithms can be used for hidden surface removal
- Scan line algorithms typically work only with polygons but fill algorithms work for more general objects

A Boundary Fill Algorithm

- Given a *seed* point (x_s, y_s) known to be in the region we want to fill
- Push the seed onto a stack
- While the stack is not empty
  - Pop a point from the stack
  - Color the point
  - For each neighbor of the point, not already colored and not on the region boundary, push the neighbor on the stack
- We need to be able to read the frame buffer to see if the pixel is colored
- We need to be able to test whether a point is inside/outside or on the boundary

Boundary Fill Algorithm, continued

- Two common topologies for neighbors
  - 4-neighbor topology (North, South, East, West)
  - 8-neighbor topology (N, NE, E, SE, S, SW, W, NW)
- Advantages
  - Easy to code (could put it in hardware)
  - Works for any shape we can test for "being inside"
- Disadvantages
  - Stack overflow

Pseudocode for Boundary Fill Algorithm

- Assume a seed point inside the region to be filled and empty stack

```
push (seed, stack);
while (not_empty(stack)) {
   point = pop(stack);
   plot(point, color);
   nghbrs = neighbors(point);
   while (nghbrs) {
      neighbor = nghbrs→point;
      if ((interior(neighbor)) and
         (not_colored(neighbor, color))) {
      push(neighbor, stack);
      nghbrs = nghbrs→next;
      }
   }
```

Pseudocode for Flood Fill Algorithm

- Assume a seed point inside the region to be filled
- Algorithm uses a 4-point topology

```
FloolFill(x, y, interior_color, new_color)
{
  if GetPixel(x, y) = interior_color {
    SetPixel(x, y, new_color);
    FloodFill(x − 1, y, interior_color, new_color)
    FloodFill(x + 1, y, interior_color, new_color)
    FloodFill(x, y − 1, interior_color, new_color)
    FloodFill(x, y + 1, interior_color, new_color)
  }
}
```

Filling Using Coherence

- Need to avoid stack overflow
- Need to improve performance
- A *span* or *run* of pixels is a horizontally adjacent group of pixels on a scan line
- Given a seed, fill the span it is on
- Fill any span above current span that is reachable
- Fill any span below current span that is reachable

- An infinite line intersects a closed bounded region R an *even* number of times (this is a little white lie)
- Fill a polygon one scan line at a time, by filling between successive intersections
- **Definition**: Scan-line order

  $(x_1, y_1) \leq (x_2, y_2)$ in scan-line order if

  $$y_1 > y_2$$

  or

  $$y_1 = y_2 \quad \text{and} \quad x_1 \leq x_2$$

- Given a polygon P, with edges $e_1, e_2, \ldots, e_n$
- Use DDA or Bresenham's algorithm to find each pixel on P's boundary
- Sort the pixels in scan-line order, storing them in a list L
- Pull off pairs of points (x_1, y_1), (x_2, y_2) from L and color each pixel (x, y) such that

  $$x_1 \leq x \leq x_2$$

- Note it must be the case that $y_1 = y = y_2$

Scan-line Fill, continued

- You've been lied to, scan-lines that hit polygon vertices intersect the region an odd number of times and horizontal edges have infinitely many intersections with a scan-line
- Corrections are:
  - Don't process horizontal edges
  - When a scan-line hits a vertex
    - record two intersection when the vertex is a local maxima or local minima
    - record one intersection when the vertex is between edges that continue increasing or decreasing

Using Buckets for Sorting

- For a high resolution device there may be thousands of intersections, so even an $n \log n$ sort routine may take time
- Bucket sorting (a form of hashing) can save time
  - For each scan-line, create a list (bucket) of pixels
  - When an intersection point (x, y) is found to be on an edge, place it in the bucket corresponding to y
  - For each scan-line only the points in its bucket need to be sorted

- Not all intersection points need to be save, we can compute them on the fly
- For each non-horizontal edge, let

  $$(x_1,\, y_1) \quad \text{and} \quad (x_2,\, y_2)$$

  denote end points
- With little loss of generality, assume $y_1 < y_2$ so that $y_2 - y_1 + 1$ scan-lines are intersected by the edge.
- Let $\delta y = y_2 - y_1 + 1$ be the number of intersected scan-lines
- Starting at the top of the edge, if y is decremented by 1, x will change by $\delta x = -1/m$ where m is the slope of the edge

- For each non-horizontal edge,
  - if high point on edge is a local maximum, store

    δy, (number of scan lines intersected)
    δx, (increment in x, for decrement in y)
    x, (value at maximum y for the edge)

    in the y-bucket of the maximum y for the edge
  - else, store

    $\delta y - 1$, (number of scan lines intersected-1)
    δx, (increment in x)
    $x + \delta x$, (value at $y - 1$ where y is maximum)

    in the y-bucket at $y - 1$ where y is the maximum for the edge
- Starting from top scan-line, pull off pairs of stored data
- Plot all points on the scan-line between the x values
- Update $x \rightarrow x + \delta x$, $\delta y \rightarrow \delta y - 1$.
- When δy becomes zero, drop corresponding triple from active-edge list
- Go to next (lower) scan line, merge any new data into active-edge list, sorted on x values

Active-Edge List Example

- Consider the polygon defined by vertices

$$P_0 = (0,0),\ P_1 = (8,1),\ P_2 = (10,5),\ P_3 = (6,4)$$

- Edge 1: $P_0 = (0,0),\ P_1 = (8,1)$

$$\delta y = 1 - 0 + 1 = 2$$

$$\delta x = -1/m = -8$$

store $x + \delta x = 0,\ \delta x = -8,\ \delta y - 1 = 1$ in bucket 0

- Edge 2: $P_1 = (8,\ 1),\ P_2 = (10,\ 5)$

$$\delta y = 5 - 1 + 1 = 5$$

$$\delta x = -1/m = -1/2$$

store $x = 10,\ \delta x = -1/2,\ \delta y = 5$ in bucket $y = 5$

- Edge 3: $P_2 = (10,\ 5),\ P_3 = (6,\ 4)$

$$\delta y = 5 - 4 + 1 = 2$$

$$\delta x = -1/m = -4$$

store $x = 10,\ \delta x = -4,\ \delta y = 2$ in bucket $y = 5$

- Edge 4: $P_3 = (6,\ 4),\ P_0 = (0,\ 0)$

$$\delta y = 4 - 0 + 1 = 5$$

$$\delta x = -1/m = -3/2$$

store $x + \delta x = 9/2,\ \delta x = -3/2,\ \delta y - 1 = 4$ in bucket 3

Bucket-List

bucket	x, δx, δy	$x, \delta x$, δy
6		
5	10, -4, 2	10, -1/2, 5
4		
3	9/2, -3/2, 4	
2		
1		
0	0, -8, 1	

Active-Edge List

5	10, -4, 2	10, -1/2, 5

Active-Edge List Example, continued

- Plot $(10, 5)$
- Update Active-Edge List

4	6, -4, 1	19/2, -1/2, 4

- Plot $(6, 4), (7, 4), (8, 4), (9, 4), (10, 4)$
- Update Active-Edge List

3	9/2, -3/2, 4	18/2, -1/2, 3

- Plot $(4, 3), (5, 3), (6, 3), (7, 3), (8, 3), (9, 3)$
- Update Active-Edge List

2	6/2, -3/2, 3	17/2, -1/2, 2

- Plot $(3, 2), (4, 2), (5, 2), (6, 2), (7, 2), (8, 2)$
- Update Active-Edge List

1	3/2, -3/2, 2	16/2, -1/2, 1

- Plot $(2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1)$
- Update Active-Edge List

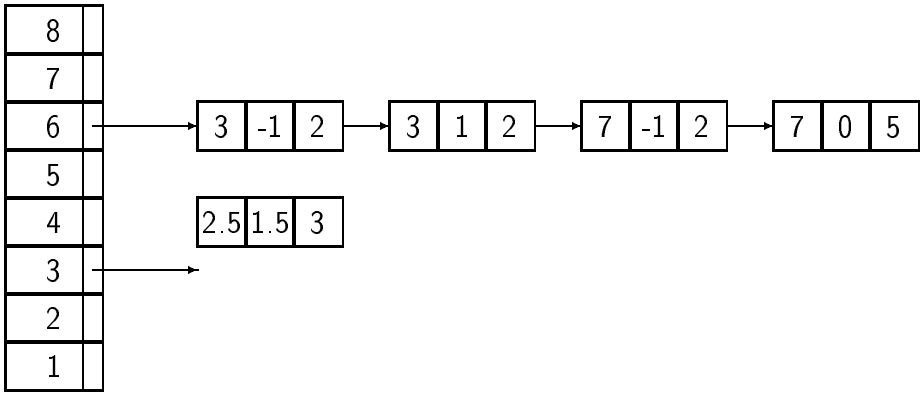
0	0, -8, 1	0, -3/2, 1

- Plot $(0, 0)$

Problems

1. Given the following end-points for line segments, determine which pixels would be illuminated using the DDA and Bresenham's algorithm
   - $(1, 1), (5, 4)$
   - $(1, 1), (5, 7)$
   - $(4, 9), (11, 3)$
   - $(4, 9), (10, 2)$
   - $(12, 3), (5, 7)$
2. Use the DDA algorithm to determine which pixels would be illuminated in drawing the line from $(2, 4)$ to $(11, 10)$
3. Use the DDA algorithm to determine which pixels would be illuminated in drawing the line from $(2, 4)$ to $(9, 12)$
4. Use Bresenham's algorithm to determine which pixels would be illuminated in drawing the line from $(2, 4)$ to $(11, 10)$
5. Use Bresenham's algorithm to determine which pixels would be illuminated in drawing the line from $(2, 4)$ to $(9, 12)$
6. Given the points $(3, 1)$ and $(10, 6)$, use Bresenham's algorithm to draw a line between them.
7. Given the equation of a circle $(x-2)^2 + (y+3)^2 - 12^2$, What pixels would Bresenham's circle algorithm illuminate, from $(2, 9)$ (the top of the circle) to the 45° line where $x = y$?
8. Use Bresenham's circle algorithm to plot the points (in the upper octant) for the circles with radii $4, 5, 6$.
9. Show how to modify the pseudocode for Bresenham's circle algorithm to draw the circle $(x-a)^2 + (y-b)^2 - r^2 = 0$.
10. Apply Bresenham's circle algorithm to the circle $(x-2)^2 + (y+4)^2 - 100 = 0$.
11. Derive a Bresenham-like algorithm for the ellipse $(\frac{x}{a})^2 + (\frac{y}{b})^2 - 1 = 0$.
12. Given a polygon specified by vertices $(0, 0)$, $(4, 4)$, $(10, 8)$, $(12, 2)$ show how to "fill" the polygon using the active edge list scan line algorithm.

13. Suppose you had the following information stored in a y-scan-line-bucket for an ordered edge list scan line fill algorithm. Which pixels would be illuminated to fill the polygon? (The data is stored in the order x, δx, δy



14. Given the quadrilateral with edges e_0, e_1, e_2, e_3 defined below, construct the triple $(x, \delta x, \delta y)$ for each edge and attach each triple to the appropriate scan line in the scan line list below (the "/" symbol indicates the pointer field is initially null). From this initial configuration, determine which pixels would be illuminated by using the active-edge list algorithm presented in class.

Edge	From	To
e_0	(1,4)	(4,6)
e_1	(4,6)	(8,2)
e_2	(8,2)	(3,0)
e_3	(3,0)	(1,4)

7	/
6	/
5	/
4	/
3	/
2	/
1	/
0	/