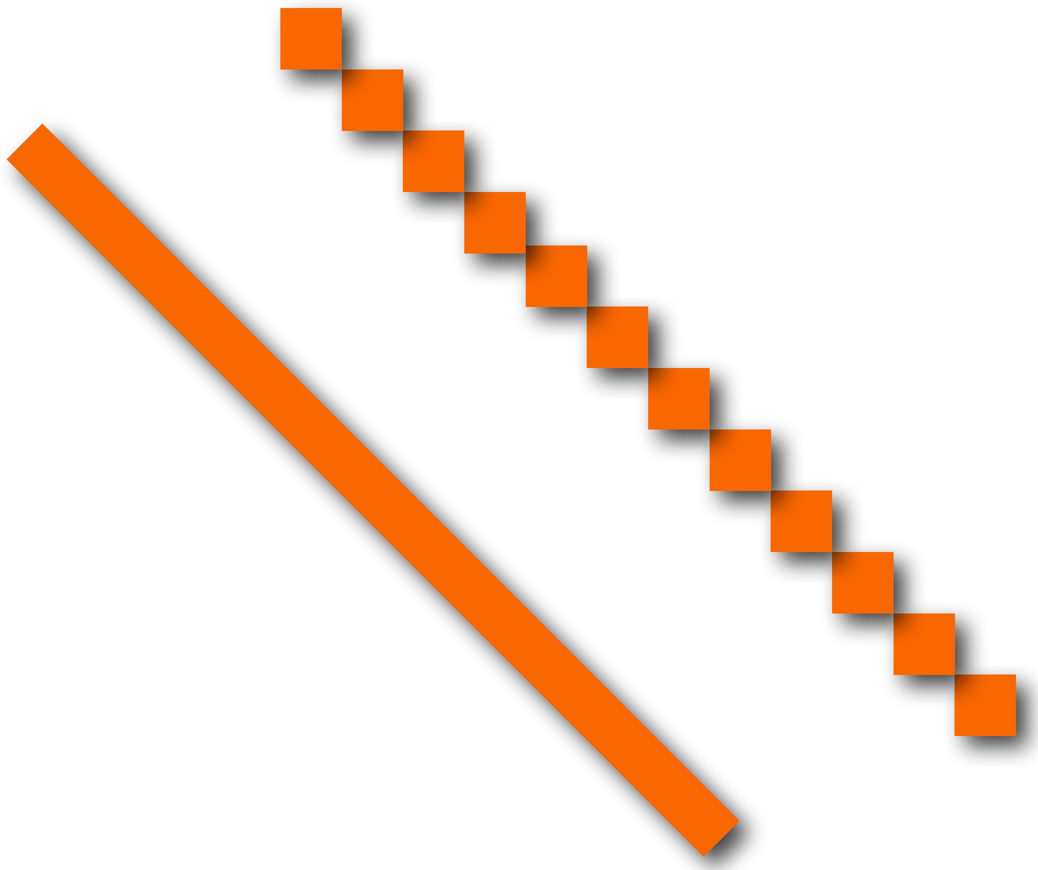


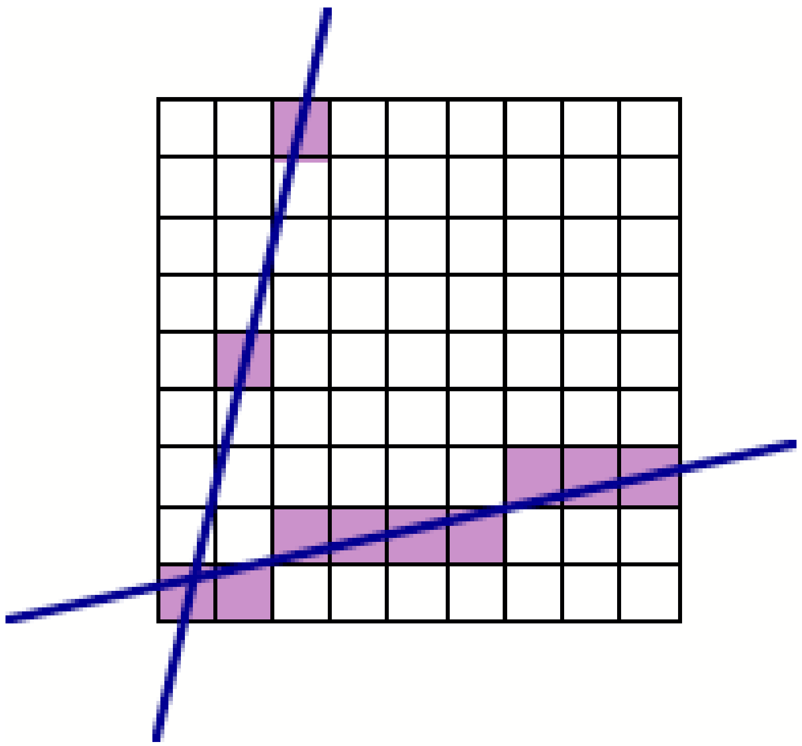
Monash University • Clayton's School of Information Technology

CSE3313 Computer Graphics

Lecture 11: Scan Conversion of Lines

Scan Conversion

- A line is defined by its two endpoints.
- To draw a line all pixels between the two endpoint pixels must be illuminated. On vector devices we can vary the deflection inputs linearly to draw a line.
- On raster devices we need to calculate exactly which pixels in the frame buffer are to be set. This conversion is usually done in hardware.



Scan Conversion (cont.)

- Criteria for well generated lines:
  - lines must appear straight or gently curving. Points must be set so that they are ≤ 1 pixel away from the true position of the line;
  - the line must start and end accurately, so lines can be joined;
  - the line should have a constant brightness (thickness) along its length (number of pixels per unit distance constant);
  - all lines should have the same density (brightness, thickness) irrespective of their length or orientation;
  - all lines should be drawn rapidly.

Incremental Line Drawing Methods

- The equation for a line is
  $$y = m x + c$$
  where m and c are constants. m gives the *gradient* or slope of the line.

- For two points (x_1, y_1) and (x_2, y_2) we have
  $$y_1 = m x_1 + c \text{ and } y_2 = m x_2 + c \qquad (1)$$
  Thus
  $$y_2 - y_1 = m(x_2 - x_1) \text{ and } y_2 = m(x_2 - x_1) + y_1 \qquad (2)$$

- To compute y_2 from (1) requires 1 multiplication and 1 addition.

- To compute y_2 from (2) requires 1 addition if $m(x_2 - x_1)$ is held constant.

- Incremental methods save on computational effort in computing a new point by computing a difference from the previous point.

The DDA algorithm (Digital Differential Analyser)

- Using a parametric representation for a line

$$x = x_1 + \alpha (x_2 - x_1)$$

$$y = y_1 + \alpha (y_2 - y_1)$$

where $0 \leq \alpha \leq 1$.

- With an incremental method we have a choice of incrementing either x or y to produce a pixel in the next column or row.

$$x = x_1 + \alpha \Delta x,\ \Delta x = (x_2 - x_1)$$

$$y = y_1 + \alpha \Delta y,\ \Delta y = (y_2 - y_1)$$

- We could increment α so that either x or y in increased by exactly one each time (e.g. $\alpha \Delta x = 1$ or $\alpha \Delta y = 1$).

- if ($|\Delta x| > |\Delta y|$) then increment x, since x is changing quicker else increment y since y is changing quicker.

DDA Algorithm

```
dx ← x2 - x1
dy ← y2 - y1
if ( abs(dx) > abs(dy) ) {
    increment ← dy / dx
    for i ← x1 to x2 {
        setPixel( i, round(y) )
        y ← y + increment
    }
} else {
    increment ← dx / dy
    for i ← y1 to y2 {
        setPixel( round(x), i )
        x ←x + increment
    }
}
```

Bresenham's Algorithm

- The main work is done in the body of the `for` loops where there is a floating point addition and at each stage we need to convert a floating point number to an integer device coordinate.

- **Bresenham's Algorithm** is more efficient than the DDA since it avoids the use of any floating point arithmetic. It still sets the same pixels that the DDA would set.

- It first checks the slope to see whether x or y should be incremented. Without loss of generality we consider the case where Δx is positive and $0 \leq m \leq 1$.

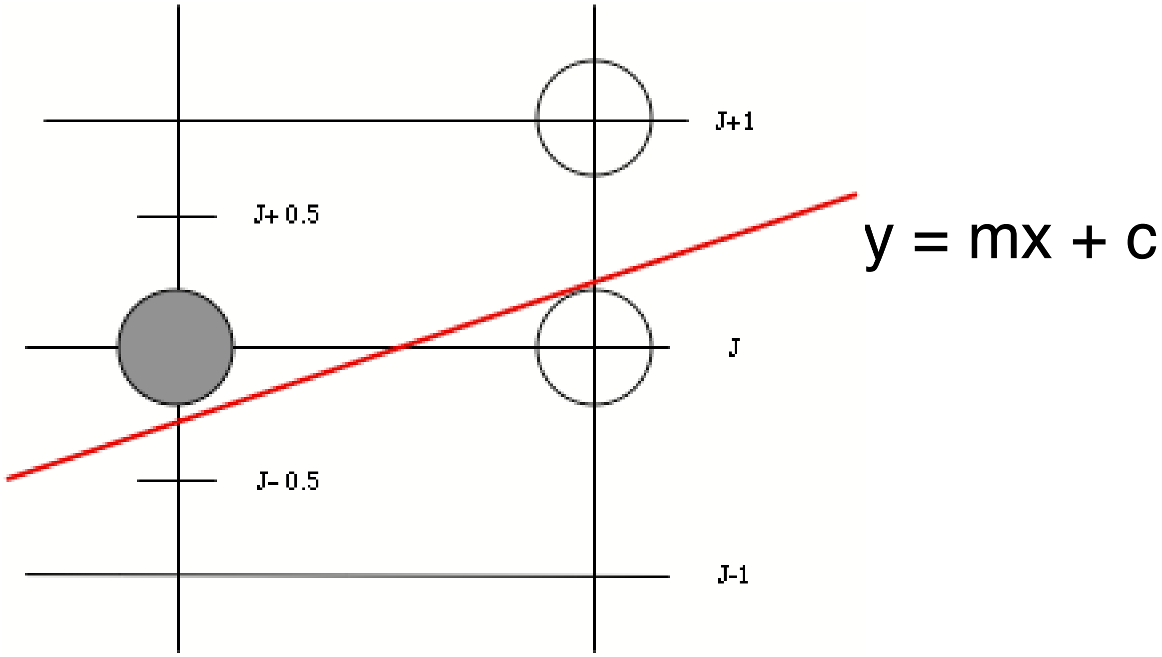
- Suppose we have just set a pixel at (I, J). (I,J) might not be on the actual line but it is within 1 pixel of the line

  $$J - 0.5 \leq y \leq J + 0.5$$

  where (I, y) is on the actual line.

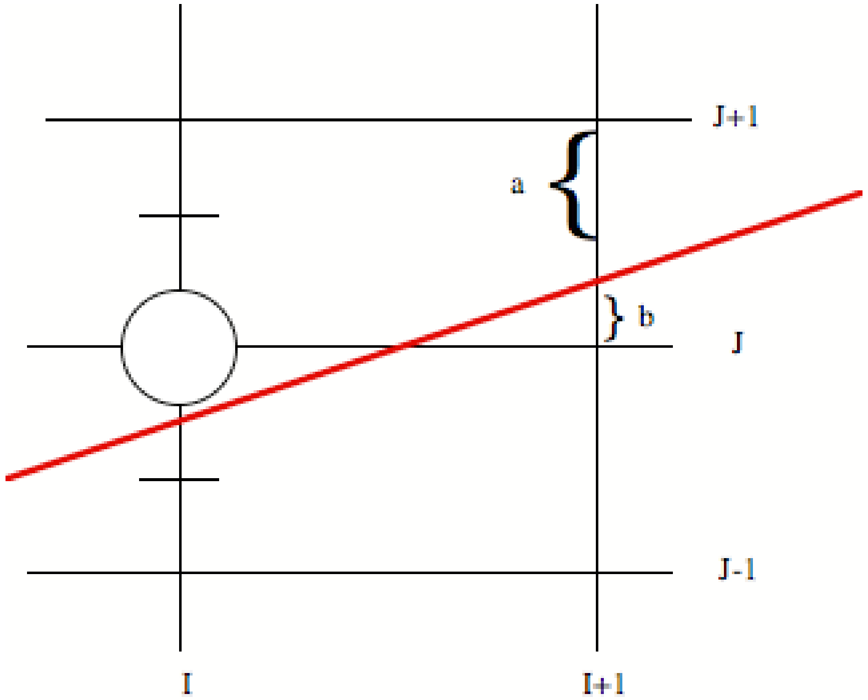
Bresenham's Algorithm (cont.)

- The constraints on m imply the next point must be either $(I+1, J)$ or $(I+1, J+1)$



(1) if $a > b$, i.e. $a - b > 0$ then the y coordinate is J.

(2) if $a < b$, i.e. $a - b < 0$ then the y coordinate is $J+1$.



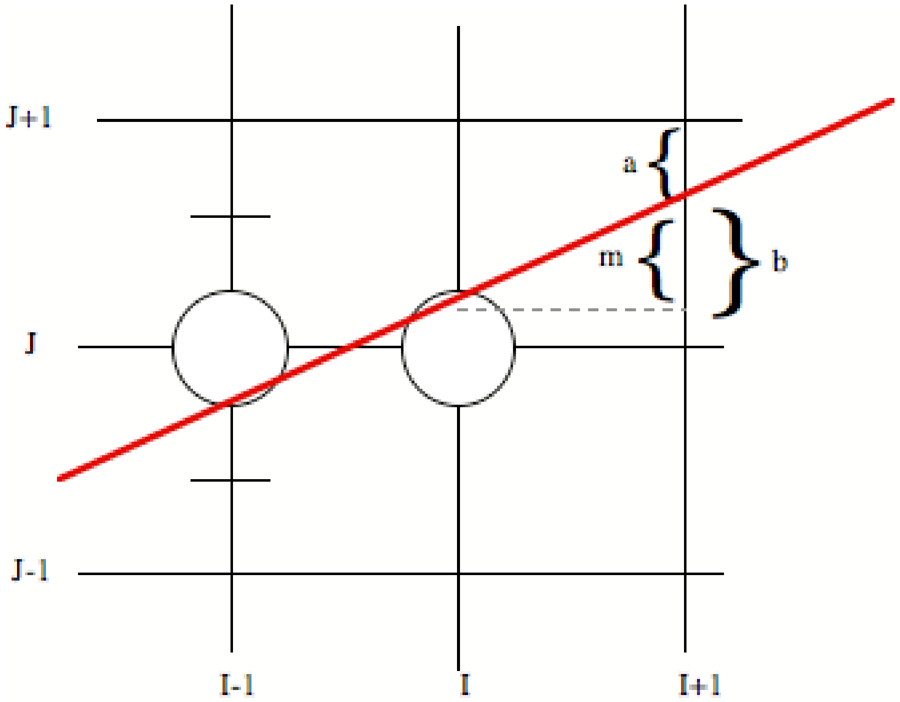
Bresenham's Algorithm (cont.)

- If J was not incremented at the last point:

  $a' \leftarrow a - m$

  $b' \leftarrow b + m$

  $a' - b' \leftarrow (a - b) - 2m$

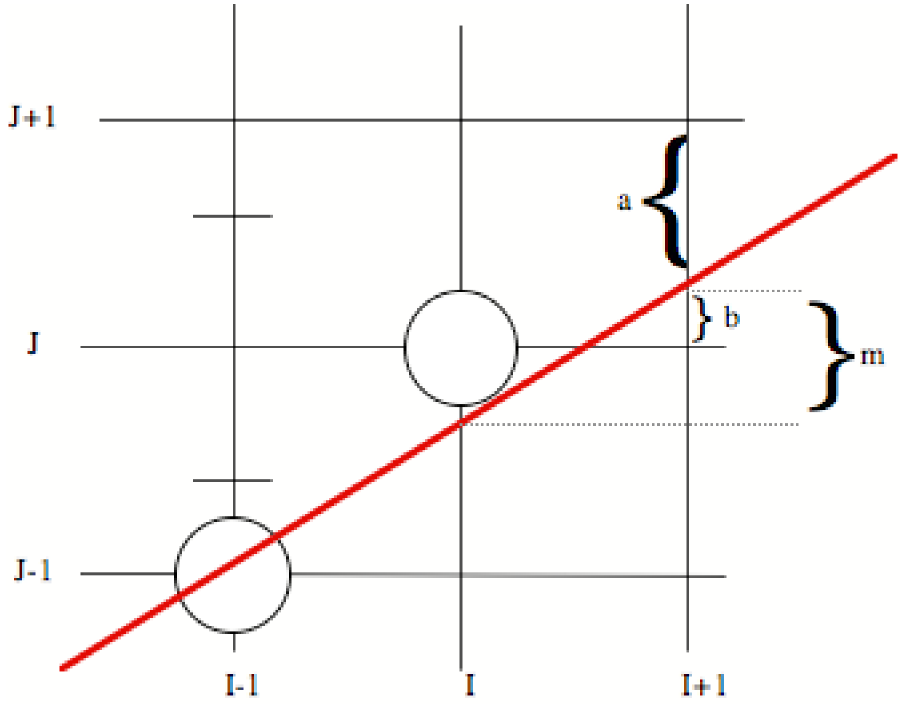


- If J was incremented at the last point:

  $a' \leftarrow 1 + a - m$

  $b' \leftarrow b + m - 1$

  $a' - b' \leftarrow (a - b) - 2m + 2$



- $(a - b)$ is the important quantity. It is a floating point number, but we are only interested in its sign.

Bresenham's Algorithm (cont.)

Bresenham's algorithm takes advantage of the fact that x_1, y_1, x_2, y_2 are all integers, so that the slope m is a rational number.

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x}$$

- In Bresenham's algorithm we scale $(a - b)$ so that we can deal with quantities that are purely integer.
- Thus $e = \Delta x(a - b)$ has the same sign as $(a - b)$ and will tell us when it is time to increment the y coordinate.
- When we decrement $(a - b)$ by either $2m$ or $2m - 2$ we need to decrement $\Delta\mathrm{x}(a - b)$ $(= e)$ by either:

  $2m\ \Delta x\ (= 2\ \Delta y)$

  or $2m\ \Delta x - 2\ \Delta x\ \ (= 2\ \Delta y - 2\ \Delta x)$

Bresenham's Line-drawing Algorithm for |m| < 1

1. Input the two line endpoints and store left endpoint in (x_0, y_0)

2. Plot the first point at (x_0, y_0)

3. Calculate the constants Δx, Δy, $2\Delta y$ and $2\Delta y - 2\Delta x$ and starting value for the decision parameter as:

   $e_0 = \Delta x - 2\Delta y$

4. At each x_k along the line, starting at $k = 0$ perform the following test: if $e_k > 0$ the next point is (x_k+1, y_k) and $e_{k+1} = e_k - 2\Delta y$

   Otherwise, the next point is (x_{k+1}, y_{k+1}) and $e_{k+1} = e_k - 2\Delta y + 2\Delta x$

5. Repeat step 4 $\Delta x - 1$ times.

Bresenham's Algorithm (cont.)

- ```
  Δx = |x2 - x1|, Δy = |y2 - y1|
  twoDy  = 2 * Δy
  twoDyMinusDx  = 2 * (Δy - Δx)
  e = Δx - 2 * Δy
  ```

- Determine endpoint to start:

  ```
  if (x1 > x2) { x = x2, y = y2, x2 = x1; }
  else { x = x1, y = y1; }
  setPixel(x,y);
  ```

- Body of main loop:

  ```
  while (x < x2) {
      x = x + 1
      if (e > 0) {
      /* a - b > 0,  a > b, don't increment J */
          e = e - twoDy;
      } else {
          y = y + 1
          e = e - twoDyMinusDx;
      }
      setPixel(x, y)
  }
  ```
  ```

k	p_k	(x_{k+1}, y_{k+1})
0	6	(21, 11)
1	2	(22, 12)
2	−2	(23, 12)
3	14	(24, 13)
4	10	(25, 14)

k	p_k	(x_{k+1}, y_{k+1})
5	6	(26, 15)
6	2	(27, 16)
7	−2	(28, 16)
8	14	(29, 17)
9	10	(30, 18)

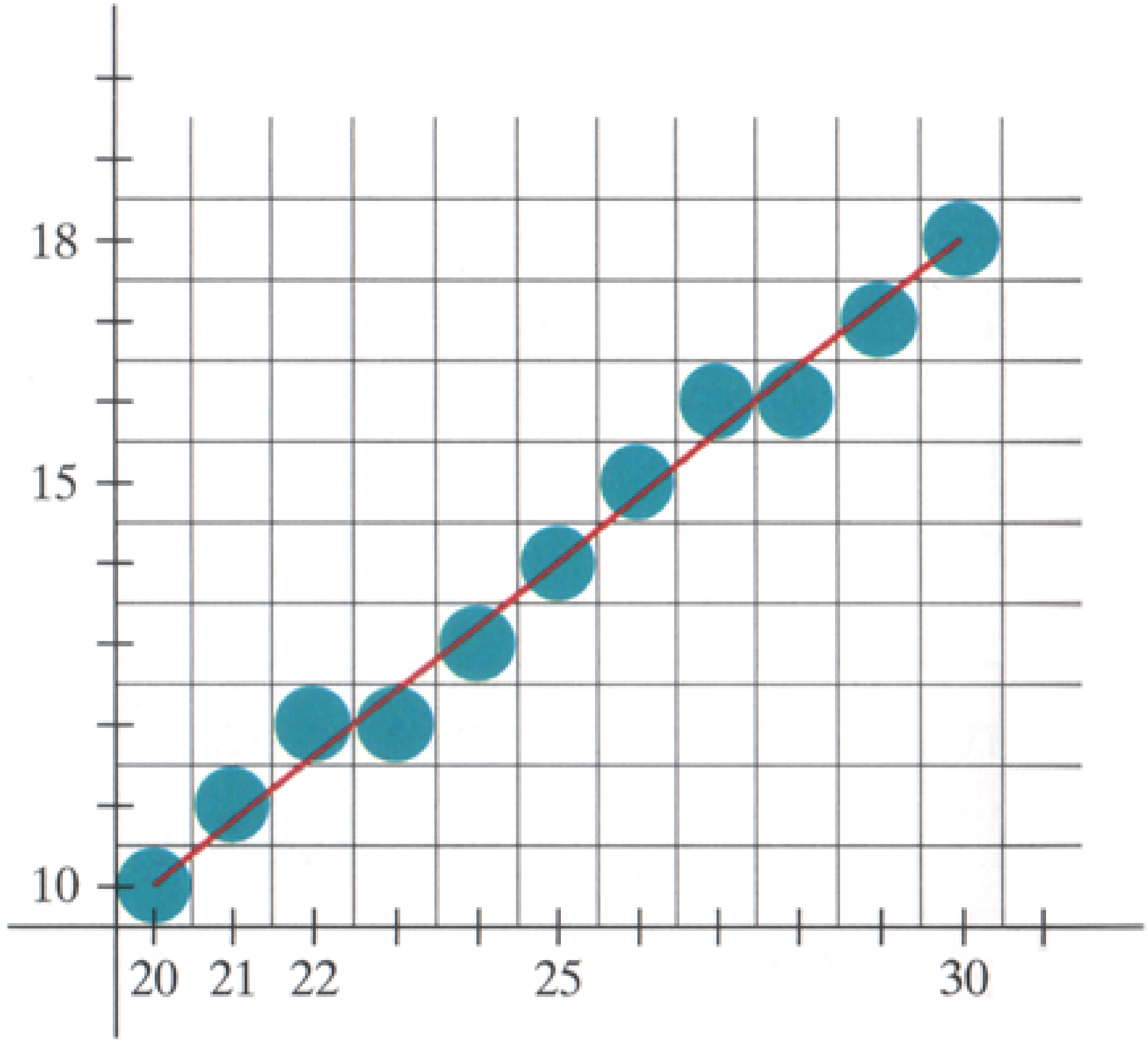


FIGURE 3–12 Pixel positions along the line path between endpoints (20, 10) and (30, 18), plotted with Bresenham's line algorithm.

Bresenham's Algorithm (cont.)

- All those operations can proceed using only integer arithmetic.
- For each output pixel we have one integer comparison, and an integer addition and possibly an increment by 1.
- Looking at the conditions for the initial point, (x_1, y_1), the initial value for $\Delta x(a - b)$ is $\Delta x - 2\Delta y$.
- The initial value assumes the first point of the line coincides exactly with the pixel location for the start of the line.
- For lines with gradient greater than 1 the roles of x and y are swapped.
- The case where lines have negative slopes is handled by symmetry.