Computer Graphics

P. Healy

CS1-08
Computer Science Bldg.
tel: 202727
patrick.healy@ul.ie

Spring 2021–2022

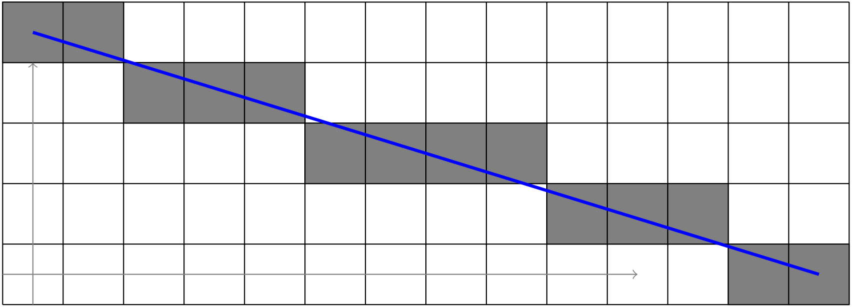
Outline

1. Drawing Algorithms
   - Line Drawing Algorithms: §6

Outline

Which pixels?

Which pixels do we "light up" when drawing the line **segment** $(0,4)$ to $(13,0)$? (Note pixel "surrounds" the (x,y) value.)



A line-drawing demo

Equation of a line

- The equation of a line is $y = mx + c$, where $m = \frac{\Delta y}{\Delta x}$ is its slope and c is its y-intercept (when x = 0)
- The line containing $p = (x_p, y_p)$ and $q = (x_q, y_q)$ has slope $m = \frac{y_q - y_p}{x_q - x_p}$
- For the same line we can figure out c since every (x, y) on must satisfy

$$\begin{aligned}\frac{(y - y_p)}{(x - x_p)} &= m\\(y - y_p) &= m(x - x_p)\\y &= mx + (y_p - mx_p)\end{aligned}$$

- Now use $y = mx + c$ to find appropriate y-value for each $x_p \leq x \leq x_q$

Equation of a line

- The equation of a line is $y = mx + c$, where $m = \frac{\Delta y}{\Delta x}$ is its slope and c is its y-intercept (when x = 0)
- The line containing $p = (x_p, y_p)$ and $q = (x_q, y_q)$ has slope $m = \frac{y_q - y_p}{x_q - x_p}$
- For the same line we can figure out c since every (x, y) on must satisfy

$$\frac{(y - y_p)}{(x - x_p)} = m$$
$$(y - y_p) = m(x - x_p)$$
$$y = mx + (y_p - mx_p)$$

- Now use $y = mx + c$ to find appropriate y-value for each $x_p \leq x \leq x_q$

Equation of a line

- The equation of a line is $y = mx + c$, where $m = \frac{\Delta y}{\Delta x}$ is its slope and c is its y-intercept (when $x = 0$)
- The line containing $p = (x_p, y_p)$ and $q = (x_q, y_q)$ has slope $m = \frac{y_q - y_p}{x_q - x_p}$
- For the same line we can figure out c since every (x, y) on must satisfy

$$\begin{aligned} \frac{(y - y_p)}{(x - x_p)} &= m \\ (y - y_p) &= m(x - x_p) \\ y &= mx + (y_p - mx_p) \end{aligned}$$

  Another way to look on this is

$$c = y - mx|_{(x_q, y_q)} = y_q - mx_q$$

- Now use $y = mx + c$ to find appropriate y-value for each $x_p \leq x \leq x_q$

Equation of a line

- The equation of a line is $y = mx + c$, where $m = \frac{\Delta y}{\Delta x}$ is its slope and c is its y-intercept (when x = 0)
- The line containing $p = (x_p, y_p)$ and $q = (x_q, y_q)$ has slope $m = \frac{y_q - y_p}{x_q - x_p}$
- For the same line we can figure out c since every (x, y) on must satisfy

$$\begin{aligned} \frac{(y - y_p)}{(x - x_p)} &= m \\ (y - y_p) &= m(x - x_p) \\ y &= mx + (y_p - mx_p) \end{aligned}$$

- Now use $y = mx + c$ to find appropriate y-value for each $x_p \leq x \leq x_q$

Digital Differential Analyzer

- If slope is $m = \frac{\Delta y}{\Delta x}$ and we're "standing" on line then advancing Δx units in x-direction and Δy units in y-direction brings us back on to line
- If we advance pixel in x-direction then we must advance pixels in y-direction to compensate
- To draw a line **segment** between two points we start at one end point and move towards the other as follows:
  - If line has then we "sample" at (step along) successive values of and

$$y_{k+1} = y_k + m$$

  - If the line has then $\frac{1}{m}$ is small and so we sample successively on using

$$x_{k+1} = x_k + \frac{1}{m}$$

- This is much faster than solving $y = mx + c$ repeatedly

Digital Differential Analyzer

- If slope is $m = \frac{\Delta y}{\Delta x}$ and we're "standing" on line then advancing Δx units in x-direction and Δy units in y-direction brings us back on to line
- If we advance pixel in x-direction then we must advance pixels in y-direction to compensate
- To draw a line **segment** between two points we start at one end point and move towards the other as follows:
  - If line has then we "sample" at (step along) successive values of and

$$y_{k+1} = y_k + m$$

  - If the line has then $\frac{1}{m}$ is small and so we sample successively on using

$$x_{k+1} = x_k + \frac{1}{m}$$

- This is much faster than solving $y = mx + c$ repeatedly

Digital Differential Analyzer

- If slope is $m = \frac{\Delta y}{\Delta x}$ and we're "standing" on line then advancing Δx units in x-direction and Δy units in y-direction brings us back on to line
- If we advance $\boxed{}$ pixel in x-direction then we must advance $\boxed{}$ pixels in y-direction to compensate
- To draw a line **segment** between two points we start at one end point and move towards the other as follows:
  - If line has $\boxed{\phantom{\text{small slope}}}$ then we "sample" at (step along) successive values of $\boxed{}$ and

$$y_{k+1} = y_k + m$$

  - If the line has $\boxed{\phantom{\text{large slope}}}$ then $\frac{1}{m}$ is small and so we sample successively on $\boxed{}$ using

$$x_{k+1} = x_k + \frac{1}{m}$$

- This is much faster than solving $y = mx + c$ repeatedly

Digital Differential Analyzer

- If slope is $m = \frac{\Delta y}{\Delta x}$ and we're "standing" on line then advancing Δx units in x-direction and Δy units in y-direction brings us back on to line
- If we advance $\boxed{\quad}$ pixel in x-direction then we must advance $\boxed{\quad}$ pixels in y-direction to compensate
- To draw a line **segment** between two points we start at one end point and move towards the other as follows:
  - If line has $\boxed{\quad\quad\quad\quad\quad\quad}$ then we "sample" at (step along) successive values of $\boxed{\quad}$ and

$$y_{k+1} = y_k + m$$

  - If the line has $\boxed{\quad\quad\quad\quad\quad\quad}$ then $\frac{1}{m}$ is small and so we sample successively on $\boxed{\quad}$ using

$$x_{k+1} = x_k + \frac{1}{m}$$

- This is much faster than solving $y = mx + c$ repeatedly

Digital Differential Analyzer

- If slope is $m = \frac{\Delta y}{\Delta x}$ and we're "standing" on line then advancing Δx units in x-direction and Δy units in y-direction brings us back on to line
- If we advance pixel in x-direction then we must advance pixels in y-direction to compensate
- To draw a line **segment** between two points we start at one end point and move towards the other as follows:
  - If line has then we "sample" at (step along) successive values of and

$$y_{k+1} = y_k + m$$

  - If the line has then $\frac{1}{m}$ is small and so we sample successively on using

$$x_{k+1} = x_k + \frac{1}{m}$$

- This is much faster than solving $y = mx + c$ repeatedly

Digital Differential Analyzer

- If slope is $m = \frac{\Delta y}{\Delta x}$ and we're "standing" on line then advancing Δx units in x-direction and Δy units in y-direction brings us back on to line
- If we advance pixel in x-direction then we must advance pixels in y-direction to compensate
- To draw a line **segment** between two points we start at one end point and move towards the other as follows:
  - If line has then we "sample" at (step along) successive values of and

$$y_{k+1} = y_k + m$$

  - If the line has then $\frac{1}{m}$ is small and so we sample successively on using

$$x_{k+1} = x_k + \frac{1}{m}$$

- This is much faster than solving $y = mx + c$ repeatedly

Digital Differential Analyzer

```
void lineDDA (int x0, int y0, int xEnd, int yEnd)
{
  int dx = xEnd - x0,  dy = yEnd - y0,  steps,  k;
  float xIncrement, yIncrement, x = x0, y = y0;

  if (fabs(dx) > fabs(dy)) steps = fabs (dx);
  else steps = fabs (dy);
  xIncrement = float (dx) / float (steps);
  yIncrement = float (dy) / float (steps);

  setPixel (round (x), round (y)); // round(x) = in
  for (k = 0; k < steps; k++) {
    x += xIncrement;
    y += yIncrement;
    setPixel (round (x), round (y));
  }
```

Digital Differential Analyzer

```
void lineDDA (int x0, int y0, int xEnd, int yEnd)
{
  int dx = xEnd - x0,  dy = yEnd - y0,  steps,  k;
  float xIncrement, yIncrement, x = x0, y = y0;

  if (fabs(dx) > fabs(dy)) steps = fabs (dx);
  else steps = fabs (dy);
  xIncrement = float (dx) / float (steps);
  yIncrement = float (dy) / float (steps);

  setPixel (round (x), round (y)); // round(x) = in
  for (k = 0; k < steps; k++) {
    x += xIncrement;
    y += yIncrement;
    setPixel (round (x), round (y));
  }
```

Digital Differential Analyzer

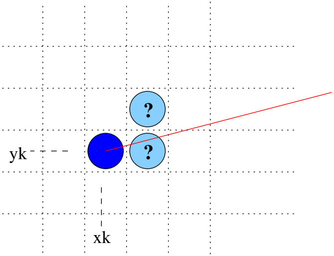
```
void lineDDA (int x0, int y0, int xEnd, int yEnd)
{
  int dx = xEnd - x0,  dy = yEnd - y0,  steps,  k;
  float xIncrement, yIncrement, x = x0, y = y0;

  if (fabs(dx) > fabs(dy)) steps = fabs (dx);
  else steps = fabs (dy);
  xIncrement = float (dx) / float (steps);
  yIncrement = float (dy) / float (steps);

  setPixel (round (x), round (y)); // round(x) = in
  for (k = 0; k < steps; k++) {
    x += xIncrement;
    y += yIncrement;
    setPixel (round (x), round (y));
  }
```

Bresenham's Line-Drawing Algorithm

- DDA requires floating point arithmetic
- Bresenham's algorithm requires only (much faster) integer calculations
- The general idea also works for circles and other curves
- Idea: as with prev. alg., from current pixel (x_k, y_k), we need to decide where to go to next
- Assuming (w.l.o.g.) that $|m| < 1$, and so we will **sample** at successive values of x again but increasing x
- That is with $x_{k+1} = x_k + 1$; find the best choice for y_{k+1}



Bresenham's Line-Drawing Algorithm (contd.)

- Given x co-ordinate of $x_k + 1$, what is **best** y value, $y_{k+1} = y_k$ or $y_{k+1} = y_k + 1$ (for the previous line)?
- The **exact** value is $y = m(x_k + 1) + c$
- So we should choose the value closest to this as best
- The two differences or *errors* are:

$$\begin{aligned} d_l &= y - y_k \\ &= m(x_k + 1) + c - y_k, \text{and} \\ d_u &= y_k + 1 - y \\ &= y_k + 1 - m(x_k + 1) - c \end{aligned}$$

- $d_l > d_u \to d_l - d_u > 0$ so sign tells us which is larger; then combining these

$$d_l - d_u = 2m(x_k + 1) - 2y_k + 2c - 1$$

Bresenham's Line-Drawing Algorithm (contd.)

- In $d_l - d_u = 2m(x_k + 1) - 2y_k + 2c - 1$ we still have $m = \frac{\Delta y}{\Delta x}$ which is a **real** no.; try fix this
- If we assume that $\Delta x > 0$ (always scan from left to right in x) then multiply through by Δx. Let $\boxed{}$ be the $\boxed{\phantom{\text{new}}}$ *predicate*:

$$\begin{aligned} p_k &= \Delta x(d_l - d_u) \\ &= 2\Delta y x_k - 2\Delta x y_k + C \end{aligned}$$

  has the same sign as before, where $C = 2\Delta y + 2c\Delta x - \Delta x$ is a **constant** throughout
- So $p_k > 0 \Rightarrow d_l > d_u$ and the upper y, y_{k+1}, is closest
- Having decided on y_k, compute p_k to find the better y_{k+1}
- Likewise, computing $p_{k+1} = 2\Delta y x_{k+1} - 2\Delta x y_{k+1} + C$ tells us what y_{k+2} to choose at $x = x_{k+2}$

Bresenham's Line-Drawing Algorithm (contd.)

- Can we squeeze any more juice from this?
- Subtracting successive values of p:

$$\begin{aligned} p_{k+1} - p_k &= 2\Delta y x_{k+1} - 2\Delta x y_{k+1} + C - (2\Delta y x_k - 2\Delta x y_k + C) \\ &= 2\Delta y (x_{k+1} - x_k) - 2\Delta x (y_{k+1} - y_k) \\ &= 2\Delta y - 2\Delta x (y_{k+1} - y_k) \end{aligned}$$

and

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x (y_{k+1} - y_k)$$

- The initial condition is (see p_k on prev. slide)

$$\begin{aligned} p_0 &= 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + 2c\Delta x - \Delta x \\ &= 2\Delta y - \Delta x \end{aligned}$$

since $y_0 = \frac{\Delta y}{\Delta x} x_0 + c$

Bresenham's Line-Drawing Algorithm (contd.)

- $p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k)$
- $p_0 = 2\Delta y - \Delta x$
- $p_k < 0 \Rightarrow$ stay with same y-value: $y_{k+1} = y_k$
- Note that $2\Delta y$ and $2\Delta x$ are **constants**
- Procedure:
  - Compute p_0 and use this to tell whether y_1 stays with same value as y_0 or to use $y_0 + 1$ at $x_1 = x_0 + 1$
  - We use this y-information in computing p_1:

$$p_1 = p_0 + 2\Delta y - 2\Delta x(y_1 - y_0)$$

  - Repeatedly use y-information from p_k to calc p_{k+1}
  - Note: each predicate p_k says what to at **next** sample, x_{k+1}

Bresenham's Line-Drawing Algorithm (concl.)

- The previous treatment was for "gently increasing lines" – lines of the form $0 \leq m \leq 1$
- What about "gently decreasing lines"?
- A similar derivation to the previous one can be done for when y *decreases* as x increases:
  - For consistency with previous case we base predicate p_k on "predicate less than 0 means we stay with current y"
  - To do this we must set

$$p_k = \Delta x(d_u - d_l)$$

  - The exact same formulae as before are yielded with the exception that $(y_{k+1} - y_k) = -1 \mathrm{or} 0$ – opposite to previous
  - To fix for this always – in either case – use absolute value of Δy
- As with DDA if $1 < |m|$ (slope is steep either positive or negative) we sample with increasing y to improve coverage