# Introduction

An accurate and efficient raster line-generating algorithm, developed by Bresenham, scan converts lines using only incremental integer calculations that can be adapted to display circles and other curves.

# Bresenham's Line Algorithm

- An **accurate**, **efficient** raster line drawing algorithm developed by Bresenham, scan converts lines using only *incremental integer* calculations that can be adapted to display circles and other curves.

- Keeping in mind the symmetry property of lines, lets derive a more efficient way of drawing a line.

  Starting from the left end point $(x_0,y_0)$ of a given line , we step to each successive column (x position) and plot the pixel whose scan-line y value closest to the line path

  Assuming we have determined that the pixel at $(x_k,y_k)$ is to be displayed, we next need to decide which pixel to plot in column $x_{k+1}$.

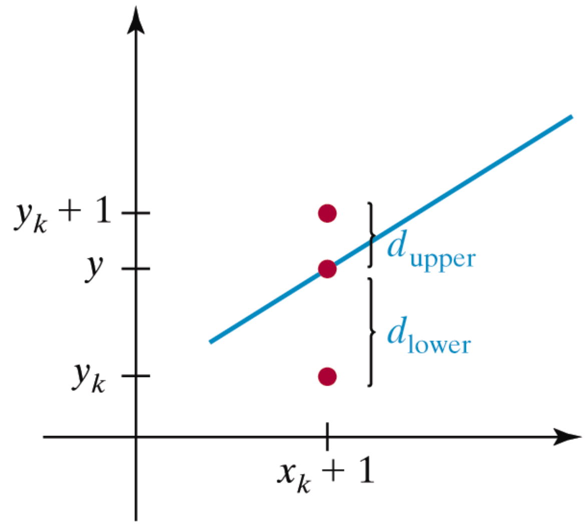


Figure 3-11

Vertical distances between pixel positions and the line $y$ coordinate at sampling position $x_k + 1$.

*Computer Graphics with Open GL*, Third Edition, by Donald Hearn and M.Pauline Baker.
ISBN 0-13-0-15390-7 

# Bresenham Line Algorithm (cont)

Choices are $(x_k + 1, y_k)$ and $(x_k+1, y_K+1)$

$$d_1 = y - y_k = m(x_k + 1) + b - y_k$$

$$d_2 = (y_k + 1) - y = y_k + 1 - m(x_k + 1) - b$$

- The difference between these 2 separations is

$$d1-d2 = 2m(xk + 1) - 2\ yk + 2b - 1$$

- A decision parameter $p_k$ for the $k^{th}$ step in the line algorithm can be obtained by rearranging above equation so that it involves only *integer calculations*

# Bresenham's Line Algorithm

- Define

$$P_k = \Delta x\ ( d_1 - d_2) = 2\Delta y x_k - 2\ \Delta x y_k + c$$

- The sign of $P_k$ is the same as the sign of $d_1$-$d_2$, since $\Delta x > 0$.
  *Parameter c* is a constant and has the value $2\Delta y + \Delta x(2b-1)$ (independent of pixel position)

- If *pixel at* $y_k$ is closer to line-path than pixel at $y_k + 1$ *(i.e, if* $d_1 < d_2$*)* then $p_k$ is negative. We plot lower pixel in such a case. Otherwise , upper pixel will be plotted.

# Bresenham's algorithm (cont)

- At step $k + 1$, the decision parameter can be evaluated as,

  $$p_k+1 = 2\Delta y x_k+1 - 2\Delta x y_k+1 + c$$

- Taking the difference of $p_{k+1}$ and $p_k$ we get the following.

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

- But, $x_{k+1} = x_k + 1$, so that

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

- Where the term $y_{k+1} - y_k$ is either 0 or 1, depending on the sign of *parameter* $p_k$

# Bresenham's Line Algorithm

- The first parameter $p_0$ is directly computed

  $p_0 = 2\,\Delta y x_k - 2\,\Delta x y_k + c = 2\,\Delta y x_k - 2\,\Delta y + \Delta x\,(2b-1)$

- Since $(x_0,y_0)$ satisfies the line equation , we also have

  $y_0 = \Delta y / \Delta x * x_0 + b$

- Combining the above 2 equations , we will have

  $p_0 = 2\Delta y - \Delta x$

  The constants $2\Delta y$ and $2\Delta y-2\Delta x$ are calculated once for each time to be scan converted

# Bresenham's Line Algorithm

- So, the arithmetic involves only integer addition and subtraction of 2 constants

*Input the two end points and store the left end point in $(x_0, y_0)$*

*Load $(x_0, y_0)$ into the frame buffer* ***(plot the first point)***

*Calculate the constants $\Delta x$, $\Delta y$, $2\Delta y$ and $2\Delta y - 2\Delta x$ and obtain the starting value for the decision parameter as*

$$p_0 = 2\Delta y - \Delta x$$

# Bresenham's Line Algorithm

*At each $x_k$ along the line, starting at $k=0$, perform the following test:*

*If $p_k < 0$ , the next point is $(x_k+1, y_k)$ and*

$$p_{k+1} = p_k + 2\Delta y$$

*Otherwise*

*Point to plot is $(x_k+1, y_k+1)$*

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$

*Repeat step 4 (above step) $\Delta x$ times*

```
#include "device.h"

void lineBres (int xa, int ya, int xb, int yb)
{
  int dx = abs (xa - xb), dy = abs (ya - yb);
  int p = 2 * dy - dx;
  int twoDy = 2 * dy, twoDyDx = 2 * (dy - dx);
  int x, y, xEnd;

  /* Determine which point to use as start, which as end */
  if (xa > xb) {
    x = xb;
    y = yb;
    xEnd = xa;
  }
  else {
```

```
    x = xa;
    y = ya;
    xEnd = xb;
  }
  setPixel (x, y);

  while (x < xEnd) {
    x++;
    if (p < 0)
      p += twoDy;
    else {
      y++;
      p += twoDyDx;
    }
    setPixel (x, y);
  }
}
```

# Application

One good use for the Bresenham line algorithm is for quickly drawing filled concave polygons (eg triangles). You can set up an array of minimum and maximum x values for every horizontal line on the screen. You then use Bresenham's algorithm to loop along each of the polygon's sides, find where it's x value is on every line and adjust the min and max values accordingly. When you've done it for every line you simply loop down the screen drawing horizontal lines between the min and max values for each line.

Another area is in linear texture mapping . This method involves taking a string of bitmap pixels and stretching them out (or squashing them in) to a line of pixels on the screen. Typically you would draw a vertical line down the screen and use Bresenham's to calculate which bitmap pixel should be drawn at each screen pixel.

# Scope of Research

- Bresenham's algorithm draws lines extremely quickly, but it does not perform anti-aliasing. In addition, it cannot handle any cases where the line endpoints do not lie exactly on integer points of the pixel grid.