

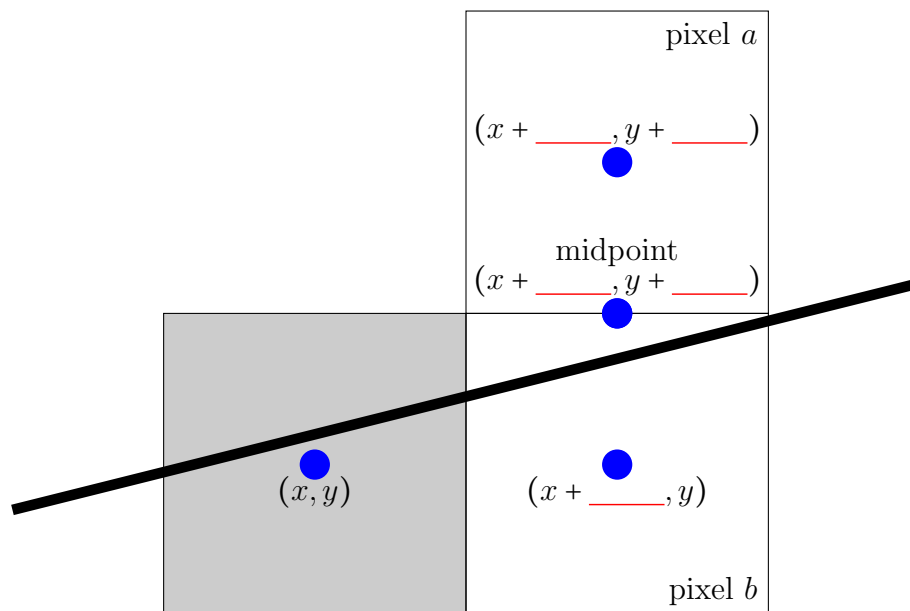
# CS130 - LAB - Bresenham's line algorithm / midpoint algorithm

Name: Jianeng Yang SID:

## Part 1: Bresenham's line algorithm / midpoint algorithm

References: [https://en.wikipedia.org/wiki/Bresenham%27s\\_line\\_algorithm](https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm)

This Lab consists of implementing the midpoint algorithm to draw continuous lines using only integer operations. Recall the line equation is  $y = mx + n$ , where  $m$  is the slope and  $n$  is the  $y$  intercept. Given two points  $p_0 = (x_0, y_0)$  and  $p_1 = (x_1, y_1)$ , the slope is calculated as  $m = \frac{y_1 - y_0}{x_1 - x_0} = \frac{dy}{dx}$ . Consider  $0 \leq m \leq 1$  (line in angle between 0 and 45 degrees). The idea is to determine which of the two pixels ( $a$  or  $b$ ) we should draw. Complete the missing coordinates of the points below.



In particular, we can evaluate the midpoint between  $a$  and  $b$  using a function  $f(x, y)$  that returns a positive number if the line is above the midpoint and negative if the line is below the midpoint. The function  $f(x, y)$  should be zero if the midpoint lies on the line

$$f(x, y) = 0 = mx + n - y \quad (1)$$

Rewrite the right-hand-side (RHS) of Equation (1) in the form  $Ax + By + C$ , where  $A$ ,  $B$ , and  $C$  depends on  $dx$ ,  $dy$  and  $n$  only. Remove any denominator by multiplying  $f(x, y)$  by  $dx$  to ensure we have an integer solution.

$$f(x, y) = (dy)x + (-dx)y + (dx)n \quad (2)$$

where  $A = dy$ ,  $B = -(dx)$ ,  $C = n(dx)$

We want to make a decision on whether to draw pixel  $a$  or  $b$  using  $f(x, y)$  on a sequence of points from  $p_0$  to  $p_1$ . Define variables  $da$  and  $db$  to hold the difference of  $f(x, y)$  from  $a$  and  $b$  to the previous midpoint  $(x + 1, y + 1/2)$ . Use Equation (2) to rewrite Equations (3) and (4) as function of  $A$  and  $B$  (you won't need  $C$ ).

$$da = f(x + 2, y + 3/2) - f(x + 1, y + 1/2) = dy - dx \quad (3)$$

$$db = f(x + 2, y + 1/2) - f(x + 1, y + 1/2) = dy \quad (4)$$

Final detail, we need to calculate the difference between the second and first point in order to use Equations (3) and (4) in a loop.

$$dinit = f(x_0 + 1, y_0 + 1/2) - f(x_0, y_0) = dy - \frac{1}{2}dx \quad (5)$$

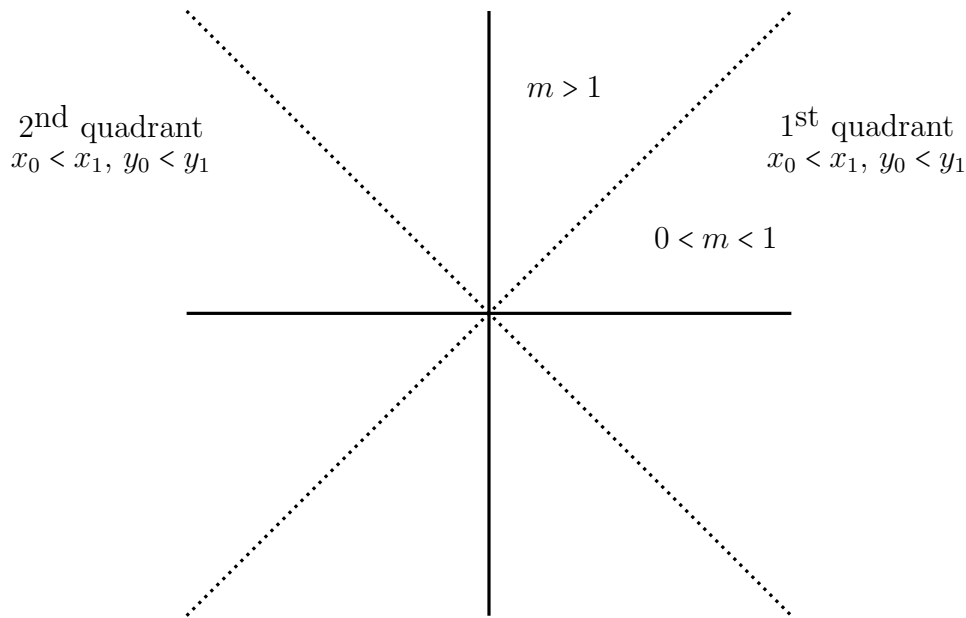
We only care whether  $d$  is positive or negative, hence, we can multiply  $dinit$ ,  $da$  and  $db$  by 2 to get equations containing only integers.

Using Equations (2) to (5), complete the midpoint algorithm below:

```
MPA(x0, y0, x1, y1):
    dx = x1 - x0
    dy = y1 - y0
    D = (2*dy - dx) // initialize D using dinit
    y = y0
    for x = x0 to x1:
        set_pixel(x, y)
        if D > 0:
            y = y + 1
            D = (D + 2*(dy - dx)) // D is positive, use da
        D = (D + 2*(dy)) // D is negative, use db
```

## Part 2: Implementing the midpoint algorithm

We considered  $0 \leq m \leq 1$  (line in angle between 0 and 45 degrees) to write the code in the previous part. How can we manage other angles? Feel free to think a little bit about it and to look at the Wikipedia page (link on Part 1). Here is a free blank space and a diagram.



Download the skeleton code on the website and write the midpoint algorithm in the function `draw_line_MPA` in `application.cpp`. To draw a pixel, you can use `set_pixel(x, y, linecolor)`, where `linecolor` is given to you as argument of `draw_line_MPA`. The following commands are available:

- Click and hold to draw lines.
- Type “c” to clear old lines.
- Type “a” to generate 1K random lines.
- Type “A” to generate 1M random lines.
- Type “m” to toggle between the MPA and the DDA algorithms.
- Type “[” or “]” to change point-size.

Make sure your code runs faster than the DDA algorithm. Why is the DDA algorithm slower? Feel free to take a look at `draw_line_DDA` code.gf

The DDA algorithm is slower than the Bresenham algorithm because DDA it uses real

arithmetic floating point operations but Bresenham use integer arithmetic fixed points. The floating point operations is very time consuming and rounding introduce errors that cause the calculated pixel positions to deviate from the true line path.

Run your MPA algorithm 3 times with 1K and 1M lines without increasing the point-size. Run the DDA algorithm 3 times with 1K and 1M lines without increasing the point-size. Fill the table below with the running time.

DDA (1K / 1M): Run 1 = 0.01/7.52; Run 2 = 0.01/7.61; Run 3 = 0.2/7.55

MPA (1K / 1M): Run 1 = 0.01/6.6; Run 2 = 0.01/6.56; Run 3 = 0.01/6.56