Aflevering 1 - Grafik

Mikkel Willén

December 6, 2022

Contents

1 Introduction

This is the first assignment in Introduction to computer graphics and is about scanconversion of straight lines. In this assignment we will try to understand, explain and implement an mid point algorithm.

2 Mid point algorithm

Before we look into the mid point algorithm, we first need to understand scan conversion of straight lines. Scan conversion of straight lines is to represent a straight line as accurately as possible using pixels.

2.1 Definition of a line

On the figure below we can see a line l and its normal vector (a, b).

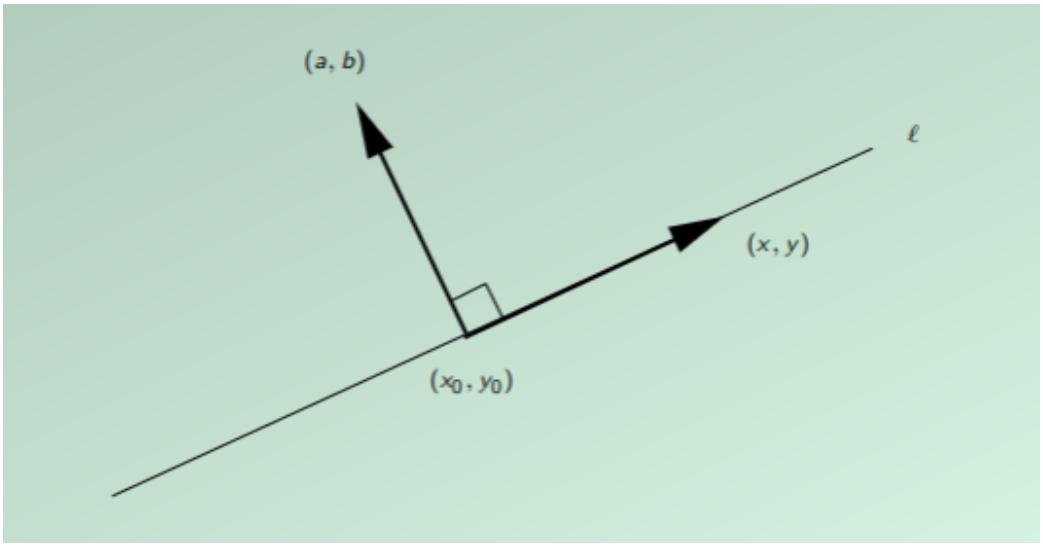

Figure 1: The line l and its normal vector (a, b).

l can be defined as

$$l \ : \{(x, y) \mid (a, b) \cdot (x - x_0, y - y_0) = 0\}$$

We can calculate the homogeneous equation of the straight line as follows

$$(a, b) \cdot (x - x_0, y - y_0) = 0$$

$$ax + by - (ax_0 + by_0) = 0$$

Then we set

$$c = -(ax_0 + by_0)$$

and we get the equation

$$ax + by + c = 0$$

which gives us the homogeneous equation of the line

$$l = \{ax + by + c = 0\}$$

Now we want to look at the distance $d(x, y)$ to a straight line. It is defined as

$$d(x, y) = \frac{(a, b)}{\sqrt{a^2 + b^2}} \cdot (x - x_0, y - y_0)$$

When we simplify this expression we get

$$d(x, y) = \frac{ax + by + c}{\sqrt{a^2 + b^2}}$$

where the numerator is the same as our l. The expression $d(x, y)$ doesn't give us an integer, but we can use the sign of $d(x, y)$ to know where the point (x, y) lies. It follows

$$d(x, y) \begin{cases} > 0 & \text{over the line} \\ = 0 & \text{on the line} \\ < 0 & \text{under the line} \end{cases}$$

Since the square root always gives a positiv number, we dont have to calculate that part, and we get the function

$$F(x, y) = ax + b + c$$

Now we don't have to compute a square root, which is a computationally expensive operation.

2.2 Theory

The mid point algorithm is an algorithm that choses between to points NE or E depending on the sign of the value d. After a point has been chosen, we update our values for the next calculation depending on wether we chose NE or E.

On the figure below we see a line and the two points the algorithm has to chose between.

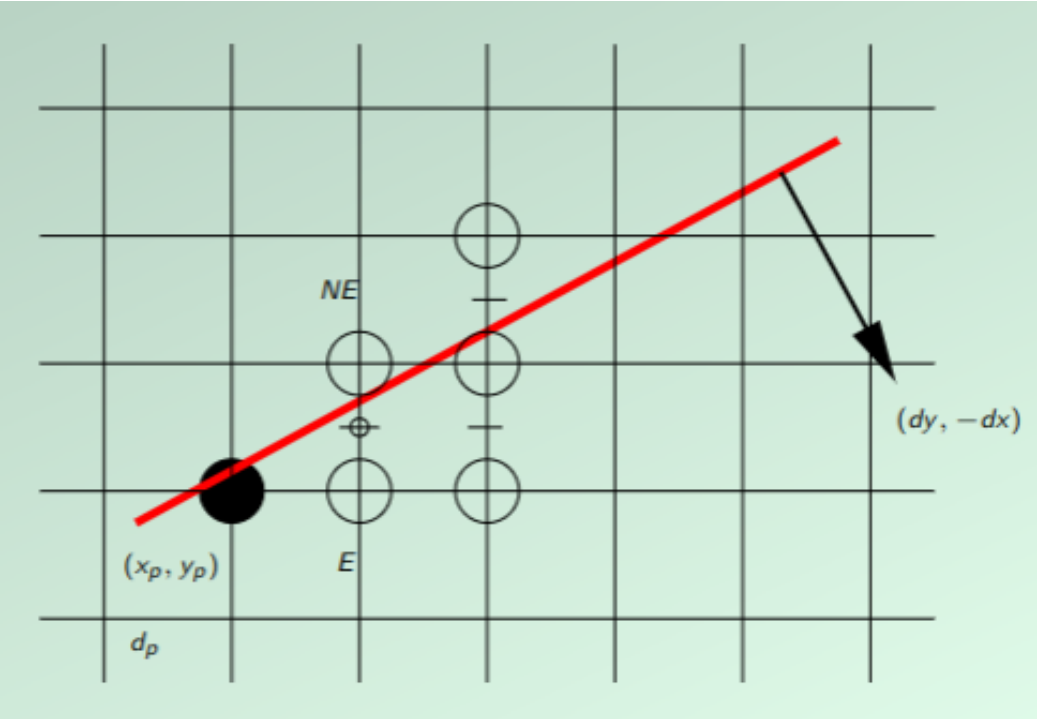

Figure 2: The pixel grid for the mid point algorithm

We start by looking at the black pixel. This is the chosen point which we call P and define as

$$P = (x_P, y_P)$$

The points NE and E are the points the algorithm has to chose between. The red line is the line we need to do scan conversion on. We have a point we call Q which lies where the red line crosses x_{p+1}. In the middle of NE and E we have the mid point M. If M is under the point Q we chose NE as

the pixel to be drawn and if M is above the point Q we chose E as the pixel to be drawn.

We start by representing the line l with the implicit function

$$F(x, y) = ax + by + c = 0$$

If we then define $dx = x_1 - x_0$ and $dy = y_1 - y_0$ we can write the slope-intercept as

$$y = \frac{dy}{dx}x + B$$

which we can rewrite as

$$F(x, y) = dy \cdot x - dx \cdot y + B \cdot dx = 0$$

if we have $a = dy$, $b = -dx$ and $c = B \cdot dx$ from our implicit form.
To use the midpoint criteria M we only need to compute

$$F(M) = F\left(x_p + 1, y_p + \frac{1}{2}\right)$$

which we can use to get the sign, which we define as

$$d = F\left(x_p + 1, y_p + \frac{1}{2}\right)$$

We can rewrite this as

$$d = a(x_p + 1) + b\left(y_p + \frac{1}{2}\right) + c$$

If $d > 0$ we chose NE and if $d < 0$ we chose E. In case $d = 0$ we can chose either.

The next step is to figure out what happens to M and d, which depends on wether we chose NE or E.
If we chose E we need to increment M with 1 in the x-direction. Since d depends on M, d is also changed, and the new value will be

$$d_{new} = F\left(x_p + 2, y_p + \frac{1}{2}\right) = a(x_p + 2) + b\left(y_p + \frac{1}{2}\right) + c$$

Looking at these two equations we can find the difference, which is

$$d_{new} = d_{old} + a$$

We see that the value we increment with after chosing E is $\Delta E = a = dy$. We can use this value to increment each time, instead of having to recompute $F(M)$.

If NE has been chosen, we need to increment M with 1 in both the x-direction and the y-direction and we have

$$d_{new} = F\left(x_p + 2, y_p + \frac{3}{2}\right) = a(x_p + 2) + b\left(y_p + \frac{3}{2}\right) + c$$

and we get the difference

$$d_{new} = d_{old} + a + b$$

and we see that we can increment with $\Delta NE = a + b = dy - dx$. We have now been through all the steps of the algorithm from [Foley, 1990].

2.3 Implementation

Implementation of the algorithm is much like it is explained i the theory section, though we there are a couple of changes, such as making d and interger, since only the sign is important.

In the implementation, we use and if-statement to check if the point is x-dominant or y-dominant. We do this by seeing, if the variation is bigger on the x-axis or y-axis. We start by initializing some variables x_step and y_step, which we can use to move up and down or left and right. Then we have a boolean value x_dominant which indicates whether the variation is bigger in the x or y direction.

In the x_dominant statement we have defined a decision variable and a variable checking if we go from left to right, or right to left. We need this, to make sure that the line is drawn the same way whether we go from left to right or vise versa. We do this both for the x_dominant statement and the y_dominant statement. The point is drawn with a glm vector. The linerasterizer functions pretty much in the same way, we have just setup up the

code in a slightly different way. The result of the two functions are the same.

The implementation takes all slopes into account and is the same no matter what direction is it drawn from. This makes it able to work on lines with slopes greater than 1 and less than 0.

2.4 Testing

To test if our implementation draws lines with all slopes, we check if it draws lines correctly in all octants. As we can see from the picture below, our implementation can draw lines in all octants. We have also checked if the lines are the same, not matter what direction they are drawn from.

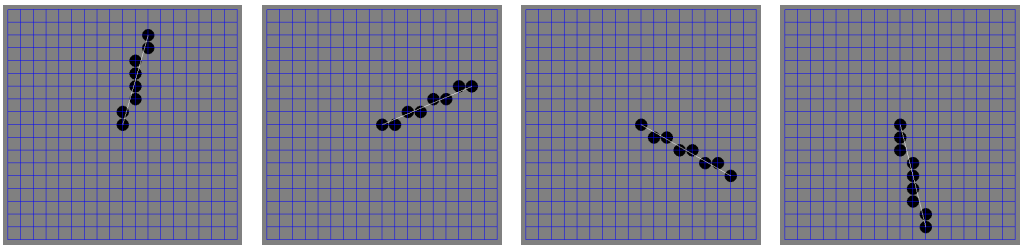

Figure 3: Lines in the first 4 octants

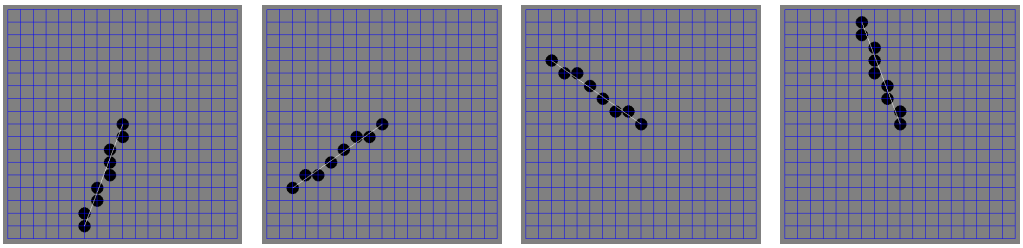

Figure 4: Lines in the last 4 octants

3 Conclusion

We have now implemented the mid point algorithm. We started by going through how the algorithm works and then wrote the code for it. We have

tested our implementation to see if it handles all slopes and if it draws the pixels the same whether we go from left to right or the other way around.

References

[Foley, 1990] Foley, J. (1990). *Computer graphics: principles and practice.* The Systems Programming Series. Addison-Wesley.