Assignment 2 - Scanconversion of triangles

Mikkel Willén

December 13, 2022

Contents

1 Introduction

This assignment is about scanconversion of triangles. We are going to look at two algorithms and how they together can compute the points in the triangle to be drawn.

2 Theory

When a triangle needs to be scan converted, it is done from the bottom left corner, up towards the top right corner. To avoid overlapping of triangles, if we e.g. have to drawn a polygon, there have been made some rules. These rules regulated which pixels are supposed to be drawn and which is not. Pixels at the bottom edge and the left edge always have to be drawn. On the other hand, pixels on the top edge and the right edge is not supposed to be drawn. This means that to identical triangles kan be converted in different ways, depending on where it located.

A triangles edge is defined by its startpoint (x_i, y_i) og its endpoint (x_{i+k}, y_{i+k}). An edge can also be seen as a piece of a line, which we can show with the following formular

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

Since we scan convert edges along the y-axis, we can rewrite the formular like this

$$x = \frac{dx}{dy}(y - \beta) = \frac{dx}{dy}x - \frac{dx}{dy}\beta$$

This means that the difference in the x-direction, when we move from y_1 til y_{i+1} is

$$x_{i+1} - x_i = \frac{dx}{dy}(y_{i+1} - y_i)$$

which we can rewrite as

$$x_{i+1} = x_i\frac{dx}{dy}(y_{i+1} - y_i) = x_i + \frac{dx}{dy}$$

If we take a look at the calculation of

$$x_{i+3} = x_i + \sum_{j=1}^{3}\frac{dx}{dy}$$

we can call x_i our integer part. The rest of we will call frac. When we do scan conversion on an edge, there are two types of edges to look at. Edges with negativ slope and edges with positiv slope. Algorithms for edges with

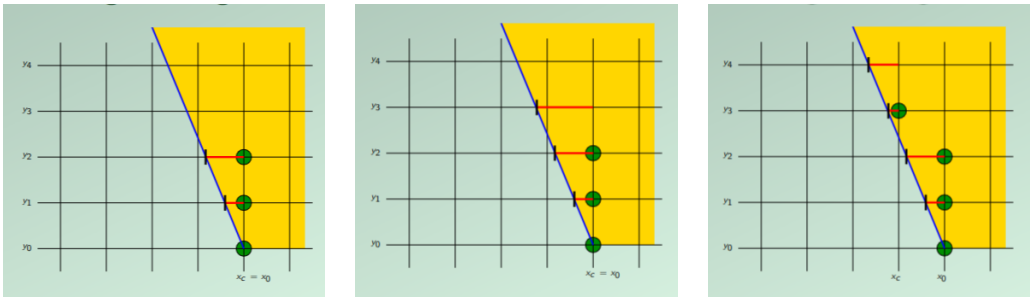

Figure 1: Example of edge with negativ slope

negativ slope starts of by setting $x_c = x_0$, and then draws a pixel there. We set our frac equal to 0, then move on and look at where the next pixel is supposed to go. The first thing we do is to set

$$\text{frac} = \text{frac} + \left|\frac{dx}{dy}\right|$$

If $frac < 1$ then $x_1 = x_c$, as seen on the first picture on figure 1. If $frac \geq 1$ then $x_1 = x_c = x_c - 1$ and $frac = frac - 1$, as seen on the second and third pictures in figure 1. Our decision therefor depends on $frac$ and $frac$ determines what x_c should be.

We can rewrite $frac$ as

$$\text{frac} = \frac{\sum |dx|}{|dy}$$

which we can rewrite as

$$\sum |dx| \leq |dy|$$

We can also write frac as

$$\text{frac} = \frac{\text{Accumulator}}{\text{Denominator}}$$

In this case frac $= 0$ would mean that Accumulator $= 0$. frac ≥ 1 would be the same as Accumulator $\geq$ Denominator. frac $=$ frac $- 1$ would be the same as Accumulator $=$ Accumulator $-$ Denominator. We will use this to help us

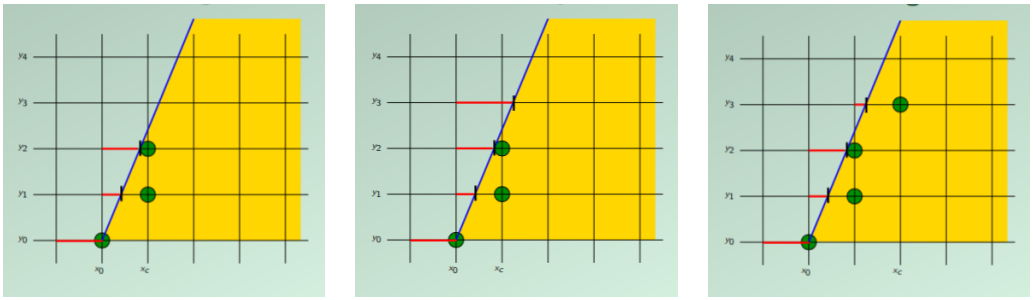

Figure 2: Example of edge with positiv slope

combine this with positiv slope later. The algorithm for edges with positiv slope starts by setting frac $= 1$ and $x_c = x_0$ as before. We will now set frac to be

$$\text{frac} = \text{frac} + \left|\frac{dx}{dy}\right|$$

If frac > 1 we set $x_1 = x_c + 1$ and frac $=$ frac $- 1$. If frac ≤ 1 then $x_1 = x_c$ as seen in figure 2. We set frac $= 1$ here to make sure our x_c will be incremented the first iteration.

Once again we have that frac $= \frac{\text{Accumulator}}{\text{Denominator}}$. So if frac $= 1$ it would mean that Accumulator $= |dy|$, frac > 1 would be the same as Accumulator $>$ Denomantor and frac $=$ frac $- 1$ would be the same as Accumulator $=$ Accumulator $-$ Denominator.

To combine these to algorithms, we set Accumulator $= 1$ and replace $\geq$ with $>$. This would make the algorithm work for both kinds of lines.

3 Implementation

Our implementation uses the to algorithms combined. Before we can use the algorithm however, we have to figure out how triangles look. There are four cases.
The first two cases are that we have a triangle with a left edge and a right edge. This triangle will then have either a top edge or a bottom edge. When we have either of these two cases, we do not have to think about top edge or bottom edge, since these are drawn with pixels from the two other edges.
The two last cases are that we have a triangle with two right edges or two left edges. If we have a triangle with two left edges, we have to run the algorithm

on both these edges.

In the file edge.cpp we do a check for whether we have a top edge or a bottom edge and assign this to a variable, then we initialize the edges and do a count along the scan line.

This is then used in the triangle.cpp file, which does the actual scanconversion. First it initializes the left edge/edges, then it initializes the right edge/edges. Then we get the coordinates for the left and right edges. The left points are then checked, if they are smaller than the right points, and thereby if they are inside the triangle or not. In the end this result in all the pixel being drawn between the left and right edge.

4 Testing

We have tested a couple of different triangles, to see if they are drawn correctly. After our tests, we kan conclude that the triangles are drawn as we wanted them two and within our rules. It does not drawn the right edge and the top edge. Some triangles are two small to be drawn due to their area. One of the pictures show one such triangle only containing one pixel. This is not a great representation of this triangle, though it cannot do better for these kinds of triangles.

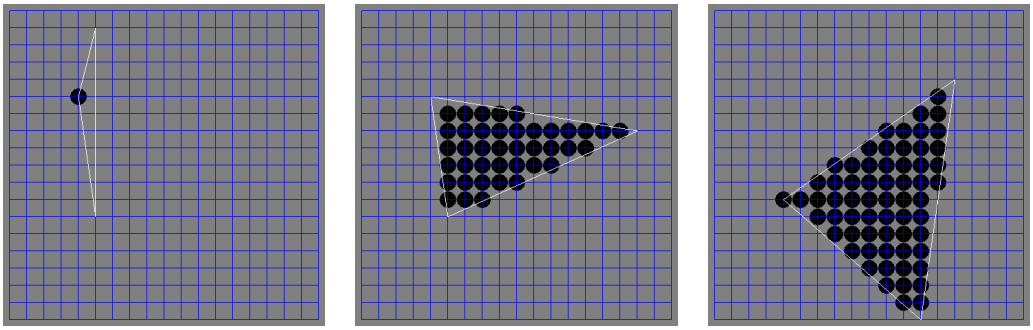

Figure 3: Some tests of different triangles

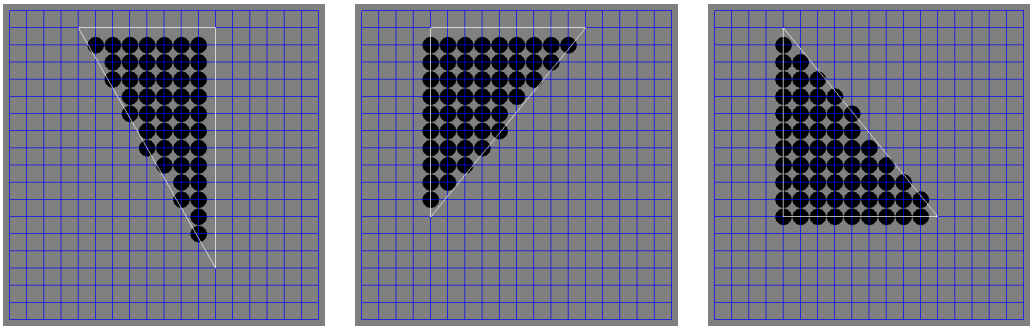

Figure 4: Some more tests of different triangles

5 Conclusion

We have made an implementation of scanconversion of triangles after going through and describing, how to handle different kinds og edges, and how it can be combined and used to make triangles. The implementation has been tested to see if it can handle different kinds og triangle and if it satisfies our demands for the function.

References