CSCI 4810 Project 1

Analyzing the Computational Complexity of Common Line Scan-Conversion Algorithms

Ellemieke Van Kints, ejv88036@uga.edu
Hamid Arabnia, hra@uga.edu

September 11, 2022

1 Introduction

In computer graphics, scan-conversion algorithms are used to translate analog signals into digital space. *Line Scan-Conversion Algorithms*, in particular, calculate the coordinates of pixels that would need to be activated on a screen in order to draw a desired line. In this project, I implemented two variations of this algorithm: *Basic Scan-Conversion* and *Bresenham's Scan-Conversion.* Both algorithms draw N lines, where N is specified by user input, and the endpoints of each line are chosen at random. My programs are written in `Python3` and use the frameworks `PIL` and `random`. The framework `time` was used during experimentation to time the critical parts of each program. The code to create a window and draw a single pixel was provided by Rosetta Code [1].

This report begins with the tests to demonstrate the capabilities of my programs. The output for each test is provided. Then, I analyze the efficiency of each algorithm through various experiments. I record their execution times with increasing values of N and confirm that *Bresenham's Scan-Conversion Algorithm* performs best out of the two. My experiments and findings are discussed in more detail below.

2 Basic Line Scan-Conversion Algorithm

To demonstrate that my *Basic Line Scan-Conversion* program works, I tested all possible input combinations. The endpoint coordinates are denoted by (x_0, y_0) and (x_1, y_1). When endpoint coordinates are listed, it can be presumed that the first coordinate refers to (x_0, y_0) and the second coordinate refers to (x_1, y_1), unless explicitly stated otherwise. I first ran tests for the special cases, which include singular points, horizontal lines, vertical lines, and perfectly diagonal lines. These tests can be seen in *Section 2.1.* I then ran tests for all other cases. These tests can be seen in *Section 2.2.* Tests were ran with endpoint coordinates that encompass all input possibilities.

2.1 Special Cases

A line is represented as a singular point when $x_0 = x_1$ and $y_0 = y_1$. To show that my program can handle this special case, the endpoint coordinates $(50, 50)$ and $(50, 50)$ were tested. The output from this test can be seen in *Figure 1.*

A line is considered horizontal when $y_0 = y_1$. To show that my program can handle this special case, two different sets of endpoints were tested. The coordinates $(20, 10)$ and $(90, 10)$ were chosen because they encompass instances when $x_1 > x_0$, and the coordinates $(60, 10)$ and $(50, 10)$ were chosen because they encompass instances when $x_1 < x_0$. The output from these tests can be seen in *Figures 2* and *3.*

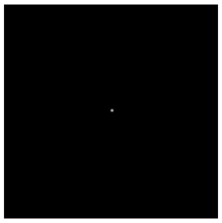

Figure 1: Output from the Basic Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(50, 50)$ and $(50, 50)$.

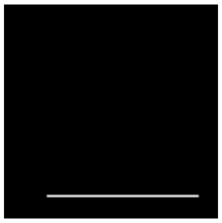

Figure 2: Output from the Basic Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(20, 10)$ and $(90, 10)$.

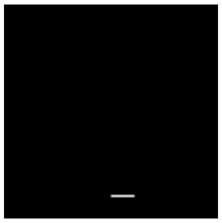

Figure 3: Output from the Basic Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(60, 10)$ and $(50, 10)$.

A line is considered vertical when $x_0 = x_1$. To show that my program can handle this special case, two different sets of endpoints were tested. The coordinates $(50, 10)$ and $(50, 30)$ were chosen because they encompass instances when $y_1 > y_0$, and the coordinates $(80, 90)$ and $(80, 10)$ were chosen because they encompass instances when $y_1 < y_0$. The output from these tests can be seen in *Figures 4* and *5*.

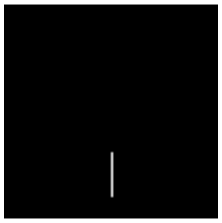

Figure 4: Output from the Basic Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(50, 10)$ and $(50, 30)$.

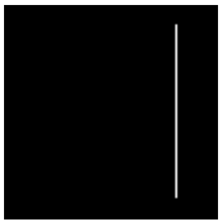

Figure 5: Output from the Basic Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(80, 90)$ and $(80, 10)$.

A line is considered perfectly diagonal when $\Delta x = \Delta y$. To show that my program can handle this special case, four different sets of endpoints were tested. The coordinates $(0, 0)$ and $(99, 99)$ were chosen because they encompass cases with a positive slope, where $x_1 > x0$. The coordinates $(0, 99)$ and $(99, 0)$ were chosen because they encompass cases with a negative slope, where $x_1 > x_0$. The coordinates $(99, 0)$ and $(0, 99)$ were chosen because they encompass cases with a negative slope, where $x_0 > x_1$. Lastly, the coordinates $(99, 99)$ and $(0, 0)$ were chosen because they encompass cases with a positive slope, where $x_0 > x_1$. The output from these tests can be seen in *Figures 6-9.*

2.2 Regular Cases

For all other cases, tests were done to encompass the remaining possible input combinations. These input combinations can be mapped to relevant inequality statements, as before, and used in conjunction to ensure comprehensive testing. These tests are described in detail below.

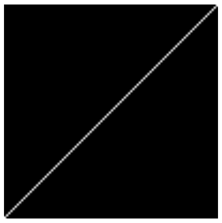

Figure 6: Output from the Basic Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(0, 0)$ and $(99, 99)$.

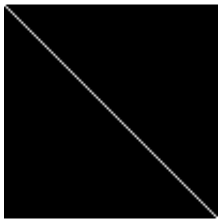

Figure 7: Output from the Basic Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(0, 99)$ and $(99, 0)$.

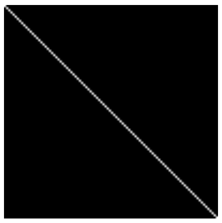

Figure 8: Output from the Basic Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(99, 0)$ and $(0, 99)$.

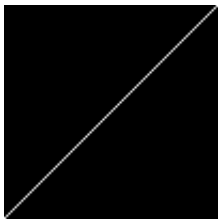

Figure 9: Output from the Basic Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(99, 99)$ and $(0, 0)$.

First, the endpoints $(0, 0)$ and $(80, 40)$ were chosen because they encompass instances when $\Delta x > \Delta y$ and $x_1 > x0$ and $y_1 > y_0$. The output from this test can be seen in *Figure 10.*

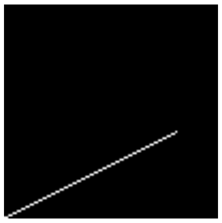

Figure 10: Output from the Basic Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(0, 0)$ and $(80, 40)$.

The endpoints $(0, 40)$ and $(80, 0)$ were chosen because they encompass instances when $\Delta x > \Delta y$ and $x_1 > x0$ and $y_0 > y_1$. The output from this test can be seen in *Figure 11.*

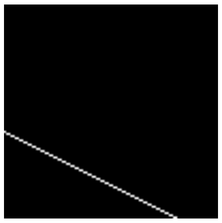

Figure 11: Output from the Basic Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(0, 40)$ and $(80, 0)$.

The endpoints $(70, 10)$ and $(30, 20)$ were chosen because they encompass instances when $\Delta x > \Delta y$ and $x_0 > x1$ and $y_1 > y_0$. The output from this test can be seen in *Figure 12.*

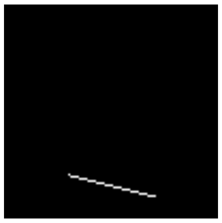

Figure 12: Output from the Basic Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(70, 10)$ and $(30, 20)$.

The endpoints $(70, 20)$ and $(30, 10)$ were chosen because they encompass instances when $\Delta x > \Delta y$ and $x_0 > x1$ and $y_0 > y_1$. The output from this test can be seen in *Figure 13.*

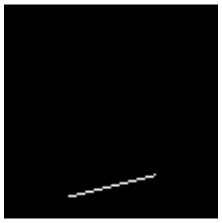

Figure 13: Output from the Basic Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(70, 20)$ and $(30, 10)$.

The endpoints $(40, 40)$ and $(50, 80)$ were chosen because they encompass instances when $\Delta y > \Delta x$ and $x_1 > x_0$ and $y_1 > y_0$. The output from this test can be seen in *Figure 14.*

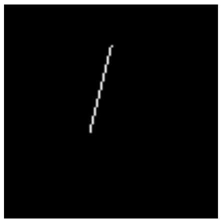

Figure 14: Output from the Basic Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(40, 40)$ and $(50, 80)$.

The endpoints $(40, 80)$ and $(50, 40)$ were chosen because they encompass instances when $\Delta y > \Delta x$ and $x_1 > x_0$ and $y_0 > y_1$. The output from this test can be seen in *Figure 15.*

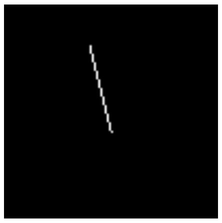

Figure 15: Output from the Basic Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(40, 80)$ and $(50, 40)$.

The endpoints $(20, 20)$ and $(10, 90)$ were chosen because they encompass instances when $\Delta y > \Delta x$ and $x_0 > x_1$ and $y_1 > y_0$. The output from this test can be seen in *Figure 16.*

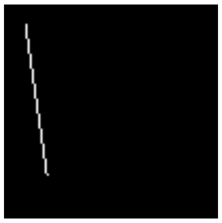

Figure 16: Output from the Basic Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(20, 20)$ and $(10, 90)$.

The endpoints $(20, 90)$ and $(10, 20)$ were chosen because they encompass instances when $\Delta y > \Delta x$ and $x_0 > x_1$ and $y_0 > y_1$. The output from this test can be seen in *Figure 17.*

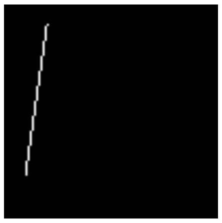

Figure 17: Output from the Basic Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(20, 90)$ and $(10, 20)$.

3 Bresenham's Line Scan-Conversion Algorithm

To demonstrate that my *Bresenham's Line Scan-Conversion* program works, I tested all possible input combinations. The endpoint coordinates are denoted by (x_0, y_0) and (x_1, y_1). When endpoint coordinates are listed, it can be presumed that the first coordinate refers to (x_0, y_0) and the second coordinate refers to (x_1, y_1), unless explicitly stated otherwise. I first ran tests for the special cases, which include singular points, horizontal lines, vertical lines, and perfectly diagonal lines. These tests can be seen in *Section 3.1.* I then ran tests for all other cases. These tests can be seen in *Section 3.2.* Tests were ran with endpoint coordinates that encompass all input possibilities.

3.1 Special Cases

A line is represented as a singular point when $x_0 = x_1$ and $y_0 = y_1$. To show that my program can handle this special case, the endpoint coordinates $(70, 70)$ and $(70, 70)$ were tested. The output from this test can be seen in *Figure 18*.

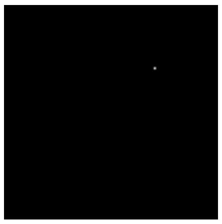

Figure 18: Output from the Bresenham's Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(70, 70)$ and $(70, 70)$.

A line is considered horizontal when $y_0 = y_1$. To show that my program can handle this special case, two different sets of endpoints were tested. The coordinates $(0, 50)$ and $(50, 50)$ were chosen because they encompass situations when $x_1 > x_0$, and the coordinates $(99, 50)$ and $(50, 50)$ were chosen because they encompass situations when $x_1 < x_0$. The output from these tests can be seen in *Figures 19* and *20*.

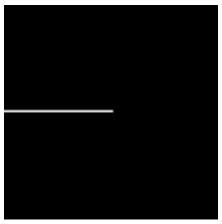

Figure 19: Output from the Bresenham's Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(0, 50)$ and $(50, 50)$.

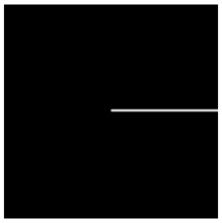

Figure 20: Output from the Bresenham's Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(99, 50)$ and $(50, 50)$.

A line is considered vertical when $x_0 = x_1$. To show that my program can handle this special case, two different sets of endpoints were tested. The coordinates $(50, 0)$ and $(50, 50)$ were chosen because they encompass situations when

$y_1 > y_0$, and the coordinates $(50, 99)$ and $(50, 50)$ were chosen because they encompass situations when $y_1 < y_0$. The output from these tests can be seen in *Figures 21* and *22*.

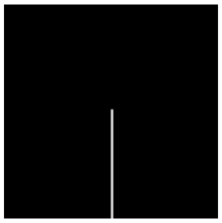

Figure 21: Output from the Bresenham's Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(50, 0)$ and $(50, 50)$.

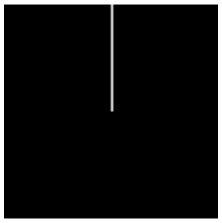

Figure 22: Output from the Bresenham's Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(50, 99)$ and $(50, 50)$.

A line is considered perfectly diagonal when $\Delta x = \Delta y$. To show that my program can handle this special case, four different sets of endpoints were tested. The coordinates $(0, 0)$ and $(99, 99)$ were chosen because they encompass cases with a positive slope, where $x_1 > x0$. The coordinates $(0, 99)$ and $(99, 0)$ were chosen because they encompass cases with a negative slope, where $x_1 > x_0$. The coordinates $(99, 0)$ and $(0, 99)$ were chosen because they encompass cases with a negative slope, where $x_0 > x_1$. Lastly, the coordinates $(99, 99)$ and $(0, 0)$ were chosen because they encompass cases with a positive slope, where $x_0 > x_1$. The output from these tests can be seen in *Figures 23-26.*

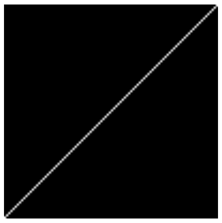

Figure 23: Output from the Bresenham's Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(0, 0)$ and $(99, 99)$.

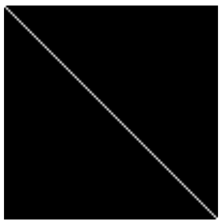

Figure 24: Output from the Bresenham's Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(0, 99)$ and $(99, 0)$.

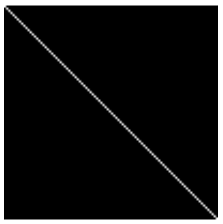

Figure 25: Output from the Bresenham's Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(99, 0)$ and $(0, 99)$.

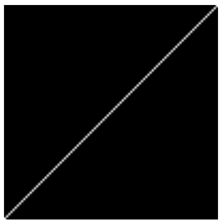

Figure 26: Output from the Bresenham's Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(99, 99)$ and $(0, 0)$.

3.2 Regular Cases

For all other cases, tests were done to encompass the remaining possible input combinations. These input combinations can be mapped to relevant inequality statements, as before, and used in conjunction to ensure comprehensive testing. These tests are described in detail below.

First, I chose to test endpoints $(10, 10)$ and $(90, 30)$ because they encompass instances when $\Delta x > \Delta y$ and $x_1 > x0$ and $y_1 > y_0$. The output from this test is shown in *Figure 27.*

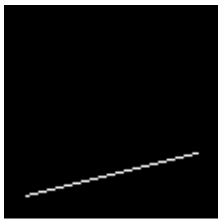

Figure 27: Output from the Bresenham's Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(10, 10)$ and $(90, 30)$.

I chose to test the endpoints $(10, 30)$ and $(90, 10)$ because they encompass instances when $\Delta x > \Delta y$ and $x_1 > x0$ and $y_0 > y_1$. The output from this test is shown in *Figure 28.*

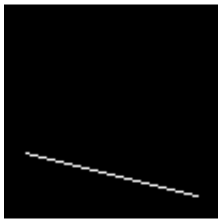

Figure 28: Output from the Bresenham's Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(10, 30)$ and $(90, 10)$.

I chose to test the endpoints $(60, 20)$ and $(20, 50)$ because they encompass instances when $\Delta x > \Delta y$ and $x_0 > x1$ and $y_1 > y_0$. The output from this test is shown in *Figure 29.*

I chose to test the endpoints $(60, 50)$ and $(20, 20)$ because they encompass instances when $\Delta x > \Delta y$ and $x_0 > x1$ and $y_0 > y_1$. The output from this test is shown in *Figure 30.*

The endpoints $(20, 40)$ and $(50, 90)$ were chosen because they encompass instances when $\Delta y > \Delta x$ and $x_1 > x_0$ and $y_1 > y_0$. The output from this test can is shown in *Figure 31.*

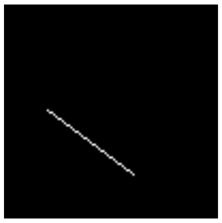

Figure 29: Output from the Bresenham's Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(60, 20)$ and $(20, 50)$.

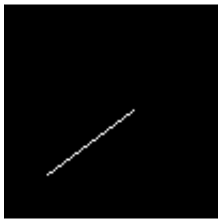

Figure 30: Output from the Bresenham's Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(60, 50)$ and $(20, 20)$.

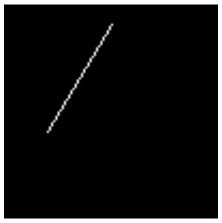

Figure 31: Output from the Bresenham's Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(20, 40)$ and $(50, 90)$.

The endpoints $(20, 90)$ and $(50, 40)$ were chosen because they encompass instances when $\Delta y > \Delta x$ and $x_1 > x_0$ and $y_0 > y_1$. The output from this test can is shown in *Figure 32.*

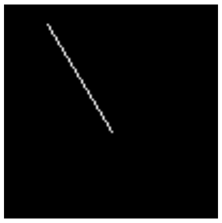

Figure 32: Output from the Bresenham's Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(20, 90)$ and $(50, 40)$.

The endpoints $(50, 0)$ and $(0, 99)$ were chosen because they encompass instances when $\Delta y > \Delta x$ and $x_0 > x_1$ and $y_1 > y_0$. The output from this test can is shown in *Figure 33.*

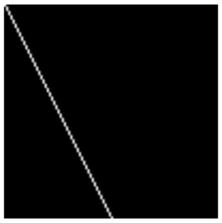

Figure 33: Output from the Bresenham's Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(50, 0)$ and $(0, 99)$.

The endpoints $(50, 99)$ and $(0, 0)$ were chosen because they encompass instances when $\Delta y > \Delta x$ and $x_0 > x_1$ and $y_0 > y_1$. The output from this test is shown in *Figure 34.*

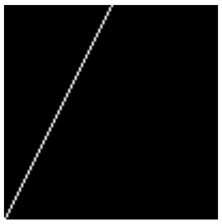

Figure 34: Output from the Bresenham's Line-Scan Conversion Algorithm, where $N = 1$ and endpoints are $(50, 99)$ and $(0, 0)$.

4 Experiments and Results

I timed both the *Basic Line Scan-Conversion Algorithm* and *Bresenham's Line Scan-Conversion Algorithm* with increasing values of N, where N represents the number of lines to be drawn. It should be noted that only the critical parts of each program were timed. The common overhead involved in random number generation and I/O operations was purposefully filtered out. The results from this experiment can be seen in *Table 1*. T_{Basic} and $T_{Bresenham}$ denote the execution times in seconds for the *Basic Scan-Conversion Algorithm* and *Bresenham's Scan-Conversion Algorithm*, respectively. The output for tests $N = 10$, $N = 100$, and $N = 1,000$ are shown in *Figures 35-40*.

N	T_{Basic}	$T_{Bresenham}$
10	0.0016	0.0012
100	0.0129	0.0118
1,000	0.0994	0.0975
10,000	0.9072	0.7201
100,000	8.7872	7.7011

Table 1: Execution times in seconds for each algorithm.

5 Conclusion

Based on my results, I can conclude that *Bresenham's Line Scan-Conversion Algorithm* performs better than the *Basic Line Scan-Conversion Algorithm*. In each test I conducted, the execution time for *Bresenham's Algorithm* remained below that of the *Basic Algorithm*. Overall, my results show that *Bresenham's Algorithm* performed 25% faster, on average. This aligns with our expectations, as *Bresenham's Algorithm* is able to map floating-point arithmetic to integer space, thus making it more compatible with analog signals. It removes the `truncate()` operation entirely, and all multiplication is done with a constant multiplier, outside of the critical loop. This makes the algorithm much faster and more efficient than the *Basic Algorithm*.

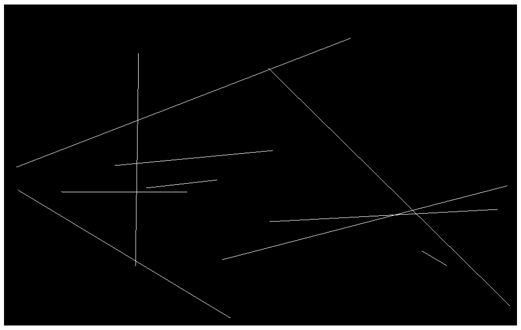

Figure 35: Output from the Basic Line Scan-Conversion Algorithm, where $N = 10$.

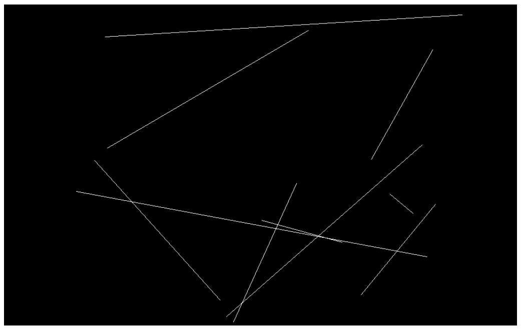

Figure 36: Output from the Bresenham's Line Scan-Conversion Algorithm, where $N = 10$.

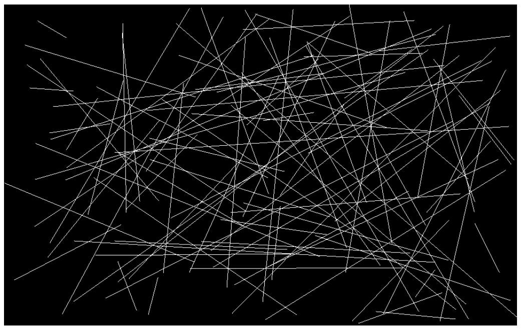

Figure 37: Output from the Basic Line Scan-Conversion Algorithm, where $N = 100$.

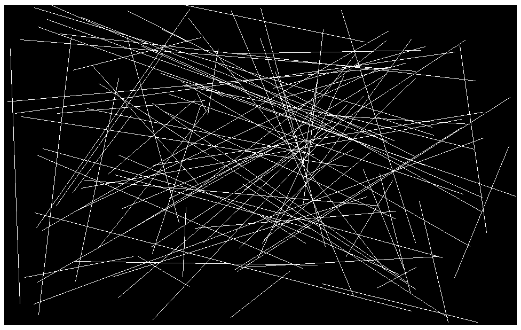

Figure 38: Output from the Bresenham's Line Scan-Conversion Algorithm, where $N = 100$.

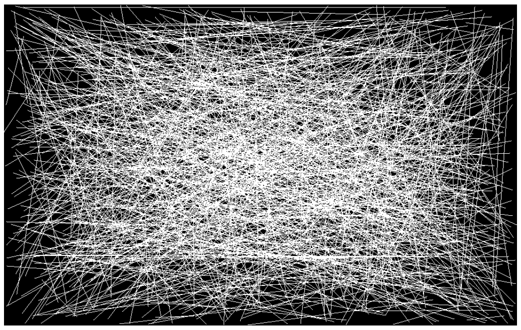

Figure 39: Output from the Basic Line Scan-Conversion Algorithm, where $N = 1,000$.

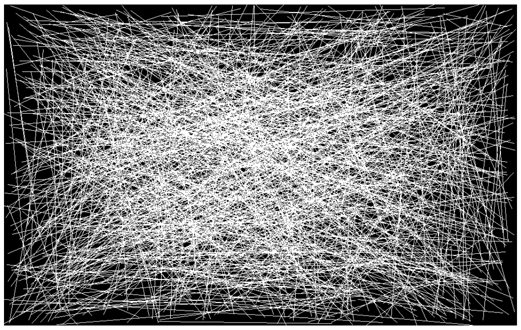

Figure 40: Output from the Bresenham's Line Scan-Conversion Algorithm, where $N = 1,000$.