Project 1: Line drawing algorithm

Abhishek Murugappan

February 10, 2022

Computer Graphics 4810

CRN: 52483

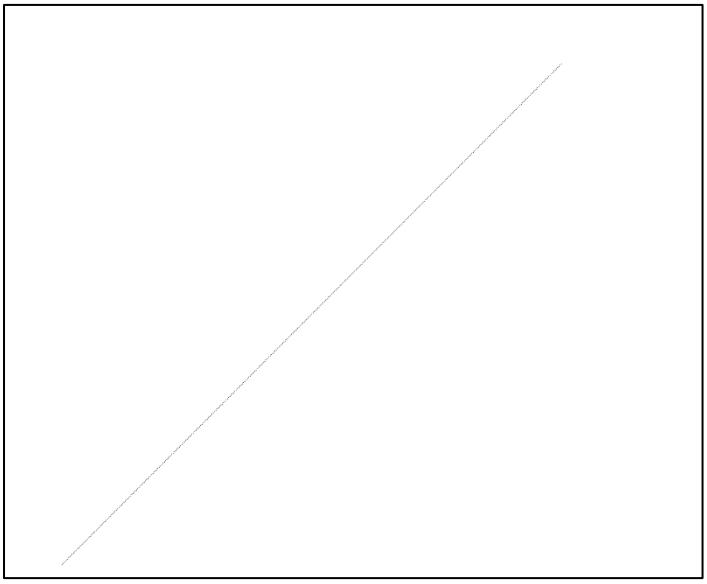
The goal of this project was to test the effectiveness of the Bresenham line drawing algorithms compared to the basic line drawing algorithm and to see whether one was faster than the other in real time. Both can draw straight lines in any orientation. However, the Bresenham line algorithm uses only whole numbers in its computation to decrease memory usage while the basic algorithm does not. The expectation is that Bresenham should perform faster than the basic algorithm in most scenarios.

TEST 1:

For the first test, I used a single line that had a slope of -8 and tested the average time for both algorithms to perform the operation. The average time for the basic algorithm was 1.621 seconds, and the average time for Bresenham was 1.832 seconds. Both algorithms were moderately close together timewise, with the basic algorithm running faster on average.

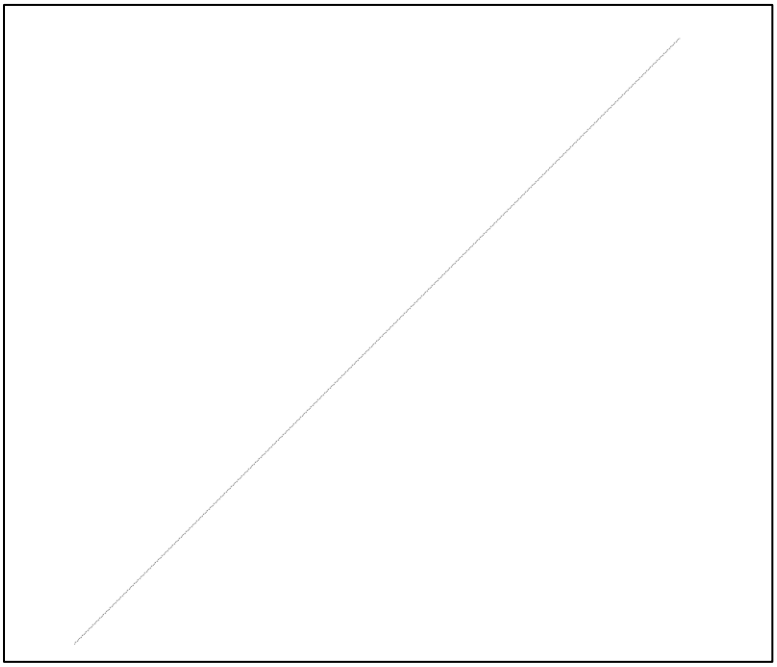
For a line from x1=800, y1=100, x2=100, y2= 800:

Bresenham Algorithm:



Trial #	Number of seconds taken
1	2.2662074 sec
2	0.6706616 sec
3	2.5576602 sec

Basic Line Algorithm:



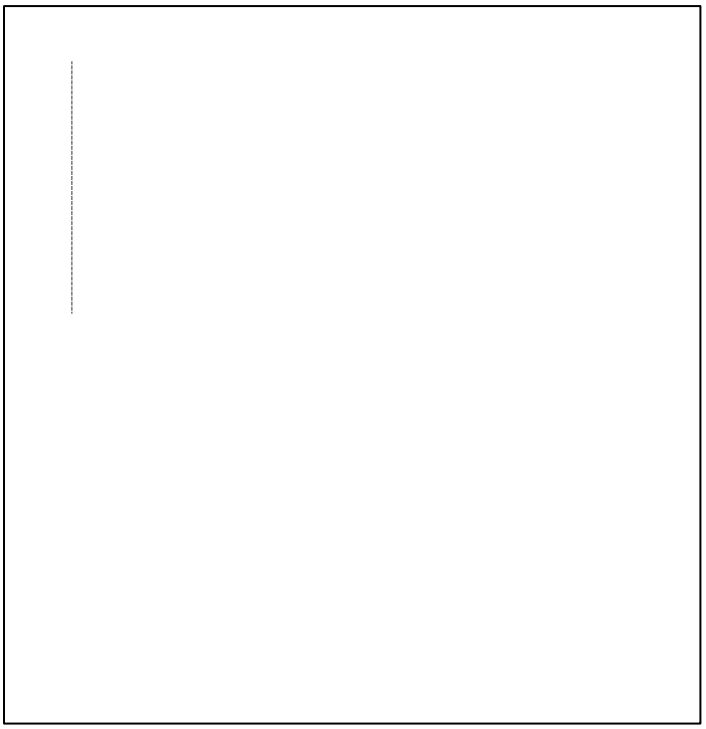
Trial	Number of seconds taken
1	2.1724827 sec
2	1.7759683 sec
3	0.9158291 sec

TEST 2:

For the second test, I used a vertical line that was also shorter to see whether both algorithms performed faster because of the length. Also, since there was an if statement in the basic algorithm it seemed likely that it would perform even faster since it would avoid the extra calculations that Bresenham needed. As expected, the average time for the basic algorithm was 0.841 seconds and the time for the Bresenham method was 1.627 seconds. The basic line conversion took half the time compared to the Bresenham method. Since the line was shorter compared to the first test, the times had reduced, and it further decreased for the basic algorithm because of an if case.

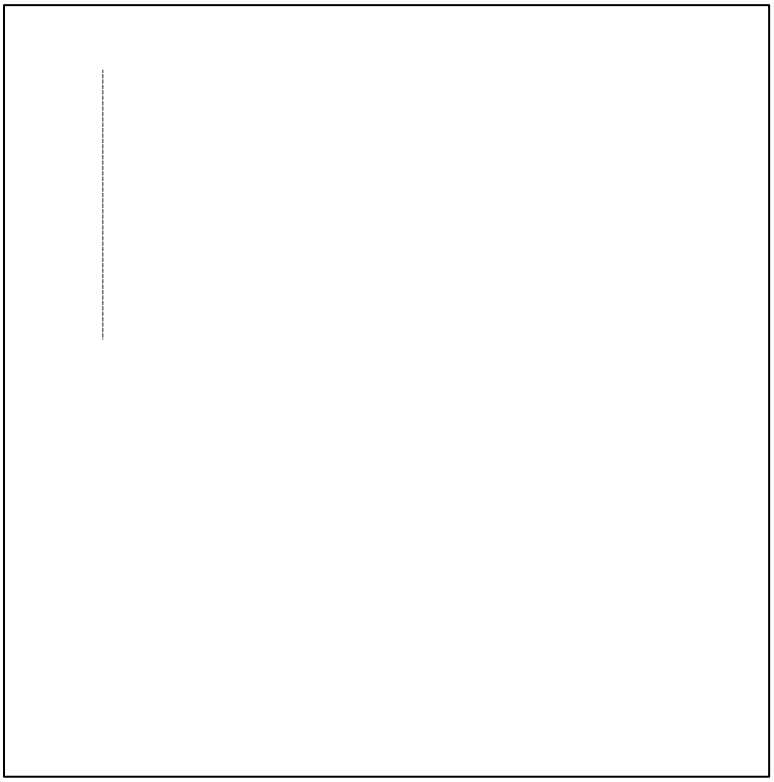
For a line from x1= 100, y1=100, x2=100, y2=300

Bresenham



Trial	Number of seconds taken
1	2.7526084 sec
2	0.8175001 sec
3	1.3115863 sec

Basic Line algorithm



Trial	Number of seconds taken
1	0.972317 sec
2	0.5180311 sec
3	1.031646 sec

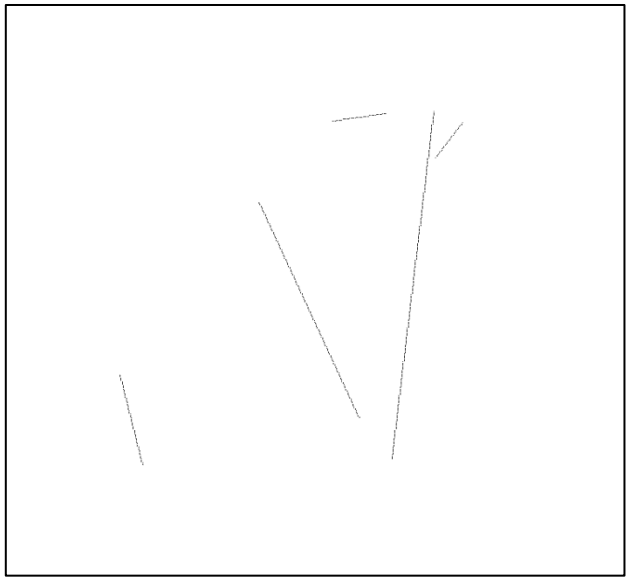
TEST 3:

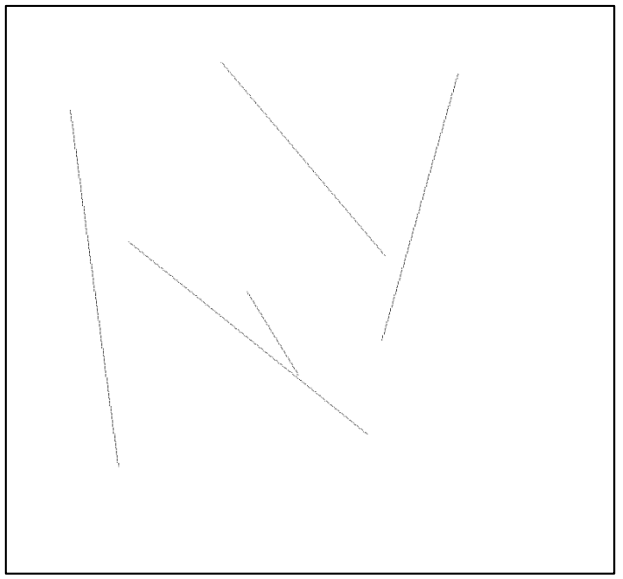
For the third test, I used 5 randomly coordinated lines to see its effects on time. My expectations were that the algorithms would be slightly slower since there would be extra lines for the program to make. As expected, With the increase of number lines both algorithms increase in time. Comparison between both lines seems unbeneficial since both algorithms generated random lines.

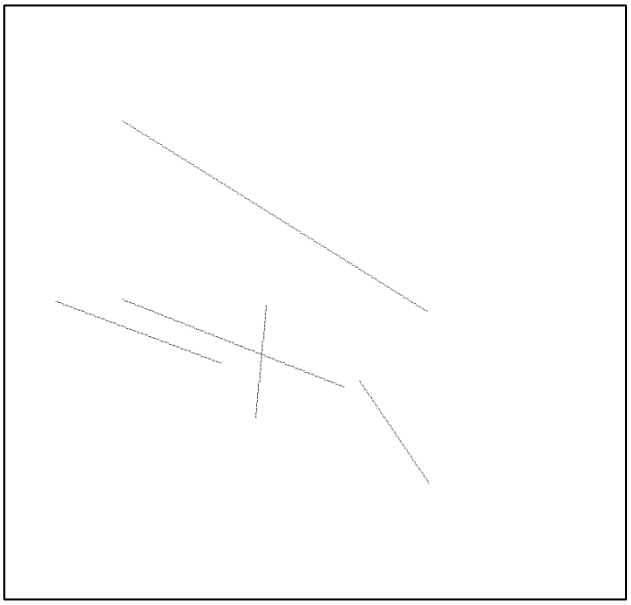
.

5 Random lines:

Bresenham:

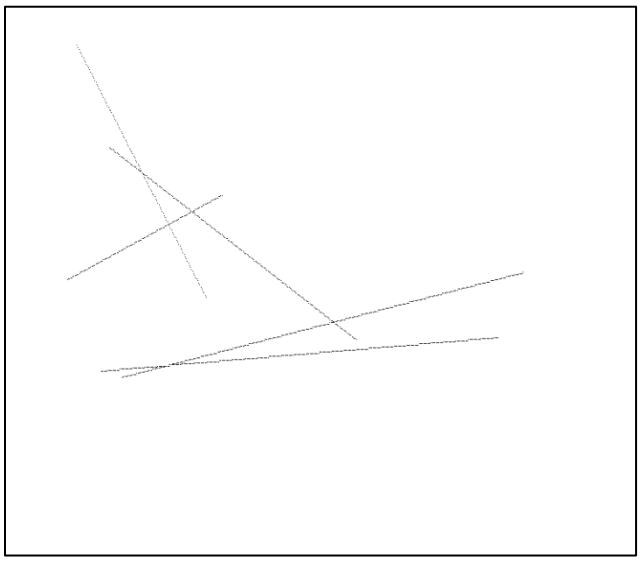


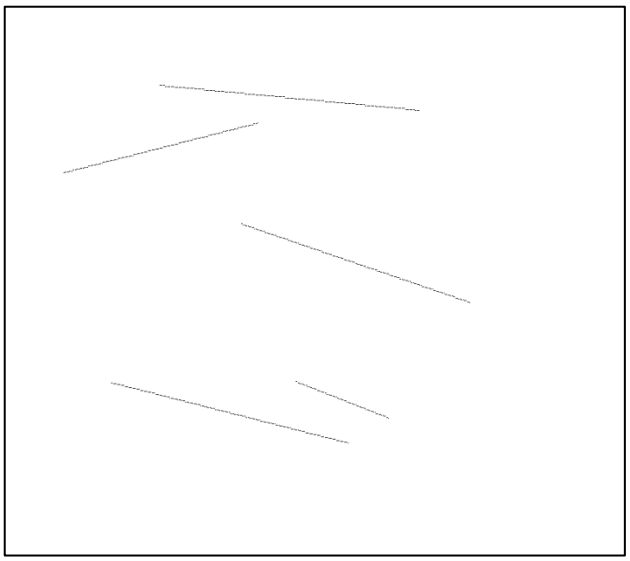


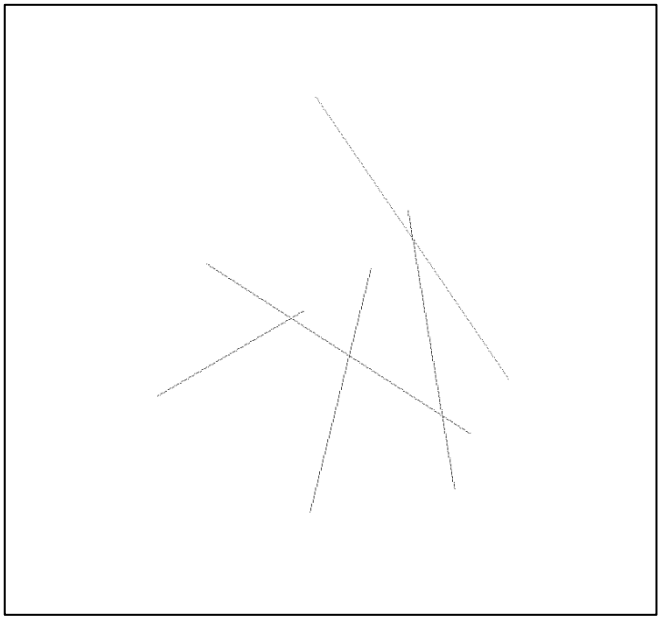


Trial	Number of seconds taken
1	**1.5533292 sec**
2	**1.930647401 sec**
3	**1.121967801 sec**

Basic Line Algorithm:







Trial	Number of seconds taken
1	**1.156104601 sec**
2	**1.038409601 sec**
3	**2.488427 sec**

Conclusion:

The goal of this experiment was to find out which line algorithm was faster, and the result was the basic line drawing algorithm. To go about testing the changes in times due to changes in lines and algorithms, I conducted a series of tests that compared size, number, and algorithms. The Basic algorithm is faster in execution of horizontal and vertical lines because of the bypass in computations and is slightly faster in other line orientations. Both algorithms increased their times whenever the length or number of lines increased. One explanation as to why the Bresenham algorithm was slower in general could be because of the implementation of the code being less optimal than it potentially could have been.