Bresenham Line Algorithm

Sami Blevens
CPSC 450
Spring 2021

Summary
The Bresenham Line Algorithm solves the problem of determining a straight-line point two locations. Bresenham's Algorithm is an incremental method, which utilizes error-tracking between the derived line and the ideal line. My implementation approach was to provide and animation of the algorithm drawing the lines it created. I used Turtle graphics with python to draw the lines.

1. ALGORITHM SELECTED
The problem I addressed was line generation. Bresenham algorithm addresses line generation through tracking the error between an ideal straight line and reused 'y' value (so that it wouldn't be recalculated at each iteration. The 'y' value is increased when the error is greater than a certain value. With the error tracking, the algorithm guarantees accuracy, though the lines generated are not always smooth. There were quite a few variations of the algorithm, especially dealing with floating points. Below is one example of the pseudocode I worked with. [1]

```
1.  dx ← x2 – x1
2.  dy ← y2 - y1
3.  y ← y1
4.  error ← 0
5.  for x from x1 to x2:
6.      plot x,y
7.      error ← error + dy
8.      if (error shifted left) >= dx
9.          y ← y + 1
10.         error ← error - dx
```

The Bresenham algorithm has the same worst case, average, and best-case time complexity. The Big-O time complexity is O(n) where n is the number of intervals in between the first x location and the second location. This is straightforward especially as much of the implementation occurs within the one for-loop. [4]

2. BASE IMPLEMENTATION
For my implementation I used python. With this, I did not have to use many data structures. I did use an array of tuples while storing the coordinates of each point generated by the algorithm. I maintained close to the pseudocode while implementing, however I did add checks for steepness of slope, the order of the x,y values to make sure the start and finish points moved left to right along the coordinated plane, and a check for negative slope to record whether to increment or decrement y. Along with this, I followed a second pseudocode in which floating points were calculated out of the algorithm by starting the error as a negative half of dx, which can be simulated by a right shift of 1. With that change, the error and y value will now be updated when the error reaches greater than 0 instead of dx. The checks for steepness of slope allowed the x and y values in to be switched so the algorithm would run against a non-steep slope, and then plotted y,x instead of x,y.

I had a series of test cases to ensure that each check in my algorithm was correctly implementing the changes in steepness and signs of slopes, as well as x and y values that were enter right to left. I also tested generating lines throughout every quadrant, as the initial pseudocode did not allow for lines not contained within the first quadrant. The checks ensure that a line was generated between the values and was the approximate shape and slope as expected.

3. EXTENDED IMPLEMENTATION
For the extended implementation, I chosen to produce an animation of the algorithm. I used Turtle in python to animate the drawing of lines in between each generated coordination point. When running the animation, it may be noted the lines seem "shaky." This is due to the integer incrementation, as it may bounce above and below the ideal slope while following it closely. [2] Figure 1 showcases three lines, purple and black finished generating, while the blue is continuing its path.

Figure 1. 3 positive slope lines being generated.

Figure 2 below showcases the lines above, with the blue line finished generating, and a negatively sloped line being generated.

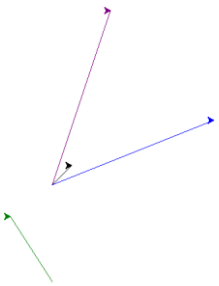

Figure 1. 3 positively sloped lines, and a negatively sloped line.

4. BUILD AND RUN INSTRUCTIONS

To run the unit tests for the algorithm, the command-line prompt is as follows

```
python3 -m unittest project_tests.py
```

To run the animation of the algorithm, the command-line prompt is as follows

```
python3 animation_line.py
```

These instructions are also available in the readme document of the github repository.

5. REFLECTION

Throughout this project, I researched the Bresenham algorithm, and built off of basic pseudocode to implement the algorithm to work for all line cases. I found it interesting the way in which the algorithm was able to make use of bit shifting in order to remove all floating-point math. In terms of what we've learned in class, this algorithm is on the of the basic algorithms. It has a complexity of O(n), and an average complexity of θ(n), as it loops through the same interval, there is not best or worst case. This problem being solved is not NP, but rather P as it is able to be solved in PTIME. I did face some original challenges with attempting to recreate the math from the original pseudocode and information into using bit shifting and addressing all possible slopes and inputs. However after looking at a few references, I was able to understand the math and why the improved version works. If I had more time, I would look into improving the way the way the error and slope was calculated as there were some implementations that utilized a multiplication of 2 that I didn't quite have time [2] [3]. Along with that I would add more robust testing and possibly improve the animation sequence.

6. RESOURCES

I utilized reference 1 as a basic implementation of the algorithm and to understand how the math of the algorithm worked out. It also explained all the ways in which the basic implementation was "broken" in that it would only work for one type of line. It also explained the associated error, and how that was utilized throughout the algorithm. [1]

I utilized reference 2, also as a source of information for how the algorithm worked but did not follow the pseudocode given. I also utilized it as a source of information for advantages and differences. [2]

Reference 3 was similar to reference 2 in the pseudocode in which it showcased, but it did not have a detailed explanation, so the reference was not incredibly helpful. [3]

I briefly looked to reference 4 in order to double check my calculation and belief for the complexity of the Bresenham algorithm. [4]

7. REFERENCES

[1] The Bresenham Line-Drawing Algorithm

https://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html

[2] Bresenham's Line Drawing Algorithm in Computer Graphics

https://www.includehelp.com/computer-graphics/bresenhams-line-drawing-algorithm.aspx

[3] Line Generation Algorithm

https://www.tutorialspoint.com/computer_graphics/line_generation_algorithm.htm

[4] Bresenham Line Drawing Algorithm

https://iq.opengenus.org/bresenham-line-drawining-algorithm/