An Analysis of Scan Converting a Line with Multi Symmetry

Md. Khairullah
Shahjalal University of Science and Technology
Sylhet-3114, Bangladesh

ABSTRACT

Line is a very important primitive in computer graphics. In this paper we analyze and discussan algorithm that exploits the multi symmetry present in certain line segments during scan conversion.This feature is implemented with the simple technique of direct line equation; digital differentiation analyzer (DDA) algorithm and the floating-point operation free Bresenham’s Algorithm. The benefit of exploiting this feature is clearly seen in the test results. Test results also show that by exploiting this feature, execution times of all these algorithms are very close, as the variations in these algorithms work for very small fraction of the line and the rest of the line is simply replicated from this pre-computation.

General Terms

Computer graphics, scan conversion of line

Keywords

Scan conversion, greatest common divisor, relative primality, symmetry, identical division

1. INTRODUCTION

The process of *scan conversion* or *rasterization*is a fundamental operation that is used extensively in computer graphics and visualization. The problem statement of scan conversion is “Given two points *P* and *Q* in *XY* plane, both with integer coordinates, determine which pixels on raster screen should be on in order to draw a unit-width line segment starting at *P* and ending at *Q*”. Sothe process of scan conversion algorithm for lines is to compute the coordinates of pixels that lie on or near an actual straight line imposed on a 2D-raster grid.The simplest way for scan conversion of lines is the sequence of operations: (i) to compute the slope *m* as *dy/dx*where *dx* and *dy* are the differences between the end points of the line along the *x-axis* and the *y-axis* respectively;(ii) to compute the y-intercept *c* as y_1-m*x_1 where (x_1,y_1) is the coordinate of the left end point; (iii) to increment *x* by *1* starting with the leftmost point to calculate $y_i = mx_i+c$ for each x_i and to intensify the pixel at (x_i, $round(y_i)$or$floor(0.5+y_i)$). This calculation finds the closest pixel. However, this strategy needs each point to be calculated requiring floating-point multiplications, addition and invoking the floor operation [1].Sometimes designs require hundreds of lines to be refreshed or redrawn in a fraction of a second. This clarifies the importance of efficientline drawing techniques.There are many elegant algorithms to handle this issue. However, by a special trick, the computation time for scan conversion can be significantly reduced for certain type of lines. Naturally every straight line has a two-way symmetry around the mid point of the linealong the both ends. That is, the change in both the coordinates for the points has the same characteristics in terms of change of coordinates, when approached from both ends of a straight line. However, beside this symmetry some line segments possess more symmetry. We say multi symmetry in a line exists when the distance along the *x-axis*or *dx* and the distance along the *y-axis*or *dy*are such that they are not co-prime or relatively prime [2]. When a *greatest common divisor(gcd)* exists other than 1, then we have the line property that we can divide the line in some equal length *divisions* which are replicas of each other in terms of changing of the coordinates of the contained points’ coordinates. In these situations we need to scan convert the first division only. Changes in the points’coordinates of other divisions only need to be replicated from the computations of the first division.This phenomenon can significantly reduce the computation time of scan conversion.

2. PREVIOUS WORKS

Bresenham developed an elegant line drawing algorithm, which uses only integer arithmetic and allows incremental calculations for finding next pixel’s coordinate [3, 4]. This incremental technique can be applied to integer computation of circles as well.In [5]a slightly different formulation of Bresenham’s algorithm known as the midpoint technique was proposed, which was later adapted by Van Aken [6] and other researchers. Wu and Rokne [7] modified Bresenham’s algorithm so that a single decision variable works for 2 pixels and consequently the line drawing time becomes half. In [8] another midpoint algorithm for line drawing is introduced. For lines and integer circles, the midpoint formulation reduces to Bresenham’s formulation and therefore generates the same pixels [9].Parallel methods for generating lines have been discussed in [10] and [11] to exploit multiple processors.

In [12] a new line drawing algorithm is developed which groups the points according to the less changing axis. It computes the first point of each group according to the algebraic equation and subsequent points of the group are obtained by incremental process. In [13] a novel approach for line drawing is shown based on the grouping idea of [12]. Here instead of explicit computation of the line equation decisions are taken by some integer operation involving the *group length* and the value of the unused amount in the current group or *carry*, and to be considered in the next group.In [14] a new approach is introduced to line drawing with logarithmic time complexity that attempts to maintain a uniform packing density of horizontal segments to diagonal segments throughout the line.

In [15], a new method is developed for drawing straight lines based on signal processing concepts related to *resampling*, *multirateprocessing* and *sample rate conversion* generally, and *decimation* in particular. This method eliminates test, compare and branch operations within the inner plotting loop. Furthermore, all multiplication, division, shifts and other complex CPU operations are eliminated from the inner loop, as well.

Initial idea of an improved algorithm for scan converting a line with multi symmetry is presented in [16]. However, we present a rigorous analysis of the test results of the implementation. Also we discover the hidden limitation of this approach.

3. EXPLOITING MULTI SYMMETRY

If two numbers are not co-prime or relatively prime we have the feature that we can divide both the numbers with their greatest common divisor and the results will be integers with the same ratio. For a line if the distances along the y-axis and the x-axis are not co-prime, we can divide both the lengths with their greatest common divisor and have some divisions, which ends on some integer coordinatesandhave theidenticalchanging behavior. This implies that we need to compute the pixels' coordinates only for the first division and pixels' coordinates for other divisions can be simply replicatedfrom the first division. For example, in figure-1 we have a line with end points A(0,0) and D(15,9). The distances along the x-axis and y-axis between the end points are 15 and 9 units respectively, having a greatest common divisor 3 and resulting in 3 equal length divisions of the main line. Each division has the lengths 5 and 3 units along the x-axis and the y-axis respectively. We observe the line divisions A(0,0) and B(5,3); B(5,3) and C(10,6); and C(10,6) and D(15,9) have the same changing behaviors and hence we need to compute the changes of the pixels' coordinates only for the first division A(0,0) and B(5,3).

However this benefit is not universal as all combinations of line's end points will not give us this feature. For the example in figure-2 a line having end points (0,0) and (15,8) will have no multi symmetry, as the distances along the x-axis (i.e. 15) and y-axis (i.e. 8) are co-prime. Probability of two numbers to be co-prime is 61% [17]. So for any raster graphics device we will have only 39% of lines having this special feature of multi symmetry.

As in other approaches of line drawing we divide atwo-dimensional plane into four quads, which ranges from 0° to 360°. Most of the line drawing algorithms limit the computation of the lines lying in the range 0° to 45° as the lines in other ranges can be easily drawn with slight modification of the same algorithm [18]. Below we list the algorithm, which exploits potential benefit of multi symmetry.

```
Algorithm Exploit-Multi-Symmetry
Input: End points of a line A(x1,y1) and B(x2,y2).
   1. divisor=gcd(y2-y1,x2-x1);
   2. yy=y1+(y2-y1)/divisor;
   3. xx=x1+(x2-x1)/divisor;
   4. length=-1;
   5. oldY=y1;
   6. Repeat steps 6a-6f for x=x1+1 to xx
        6a. Compute y using any line drawing
            algorithm; //ex. DDA or Bresenham
        6b. length=length+1;
        6c. store[length]=y-oldY;
        6d. oldY=y;
        6e. intensify pixel at (x,y)
        6f. x = x+1
   7. Repeat steps 7a and 7b for count=1 to divisor-1
        7a. Repeat steps i-iv for kount=0 to length-1
               i. y=y+store[kount];
              ii. intensify pixel at (x,y)
             iii. x=x+1;
              iv. kount=kount+1
        7b. count=count+1;
```

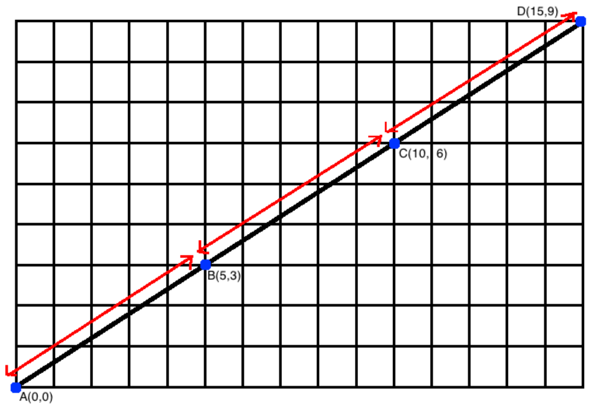

Figure 1: a line having 3 identical divisions

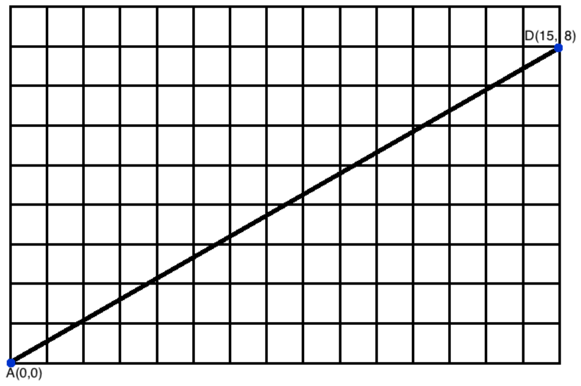

Figure 2: a line having no symmetry

4. RESULTS AND DISCUSSIONS

We already discussed the fact that the probability of a line having multi symmetry is only about 39%. Hence we confine all following discussion regarding test cases, results and performance analysis only to lines having multi symmetry and avoid other lines. We used the OpenGL routine *glVertex2f*to draw or intensify a pixel described in the algorithm. As the time to draw a single line is very little we drew 10,000 lines to have a significant time measures for comparison purposes. We worked with 3 different lines having various degree of multi symmetry. Line 1 has the end points (0,100; 1200,700) having greatest common divisor 600 and the highest amount of potential symmetry (600 identical divisions each having 2 pixels). On the contrary, line 2 has end points (0,100; 1200,102), the greatest common divisor 2 and the lowest amount of symmetry (2 identical divisions each having 600 pixels). Line 3 is in between in terms of multi symmetry, compared to line 1 and line 2. It has the end points (0,100; 1200,125), the greatest common divisor 25 and 25 identical divisions (each with 48 pixels) in it.

Table 1 lists the execution time of the simple algorithm (direct line equation), the DDA algorithm and the Bresenham's algorithm on these lines. It shows the difference between the classical version of these algorithms and the new version exploiting multi symmetry and also shows the *acceleration*, which we define as the ratio of the scan conversion times by the classical versions to the new versions.

Figure-3 shows the execution time of different new algorithms exploiting multi symmetry for the mentioned 3 lines. The interesting thing is that for the line 1 having the highest amount of multi symmetry, all the algorithms performs quite same. For the line 2, the differences among the drawing time are significant and the performance graph follows the case in the classical version of the algorithms. For line 1 we have to compute the y-coordinates only for 2 pixels by using the algorithms. Remaining pixels in rest of the divisions are replicated based on the computations for the first division. Computations for the first division having only two pixels do not have enough impact on the *total execution time* and hence we have the quite the same execution time for all the algorithms. On the other hand, line 2 has only two identical divisions and we need to go up to the middle of the line (600 pixels) using the classical algorithms for computations of the pixels and hence we have a significant difference among the execution time of the 3 algorithms. In line 3, 48 pixels need to be computed using the classical algorithms and 24 divisions can reuse these results and hence has less impact on the total execution time.

Figure-4, figure-5 and figure-6 depict the reduction in execution time by exploiting multi symmetry compared to the classical versions of the algorithms for line 1, line 2 and line 3 respectively. We can also perceive the acceleration in performance by these figures. In table 1 we observe that the acceleration in the simple algorithm for line 1 and line 3 is roughly 160% where in the worst case (line 2) it falls to 133%. For the DDA algorithm we again observe that the acceleration for line 1 and line 3 is quite same (137% and 134% respectively) which is 125% for the worst case (for line 2). As the classical version of DDA algorithm is better than the classical version of the simple method, we had less scope to enhance by exploiting multi symmetry.

For the same reasons Bresenham's algorithm achieves the lowest acceleration by exploiting multi symmetry, as it is the best algorithm among the three in their classical versions. And also an interesting thing regarding exploiting multi symmetry by Bresenham's algorithm is that accelerations for all the three lines are quite same (roughly 123%). This is a contradiction to the other two algorithms where a clear fall in the acceleration for the line 2 is observed. This may be due to the fact that the Bresenham's algorithm in its classical version is very good and has not enough potentiality to improve significantly by exploiting multi symmetry.

By these results, we can derive the *totalacceleration* by introducing this new feature of multi symmetry. We consider the probability of a line having multi symmetry to derive the totalacceleration by the formula:

*totalacceleration = acceleration using multi symmetry*probability of a line having multi symmetry+100* probability of a line lacking multi symmetry*

Then, for the simple algorithm the acceleration is (161*0.39+100*0.61)% or 124% in the best case and (133*0.39+100*0.61)% or 113% in the worst case. For the DDA algorithm the acceleration is (137*0.39+100*0.61)% or 115% in the best case and (125*0.39+100*0.61)% or 110% in the worst case. For Bresenham's algorithm the acceleration is (123*0.39+100*0.61)% or 109% roughly for all the cases. So we can say by exploiting multi symmetry feature we can speed up scan conversion of line by about 10% to 25%.

Table 1: Execution time (in seconds) of the classical and new version of various algorithms

	Line 1 (0,100; 1200,700)			Line 2 (0,100; 1200,102)			Line 3 (0,100; 1200,125)		
Algorithm	Classical	New	Acceleration	Classical	New	Acceleration	Classical	New	Acceleration
Simple	0.147399	0.091555	1.610	0.146449	0.110364	1.327	0.147063	0.09271	1.586
DDA	0.127206	0.092833	1.370	0.124050	0.099488	1.247	0.125672	0.093756	1.340
Bresenham	0.112921	0.091798	1.230	0.114078	0.093119	1.225	0.112003	0.090649	1.236

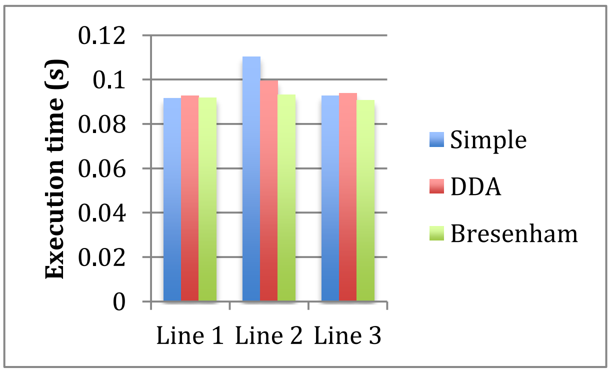

Figure 3: Comparison of the execution time exploiting multi symmetry

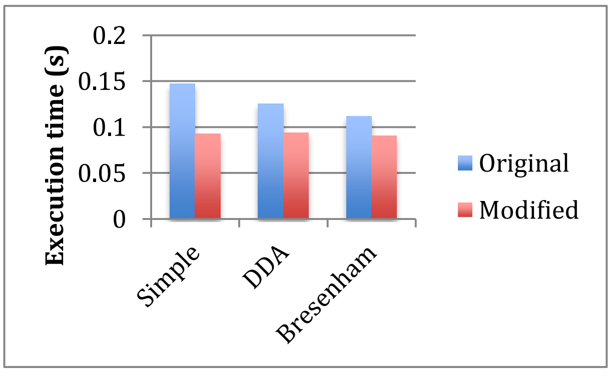

Figure 4: Comparison of the execution time of the classical and the new algorithm for line 1

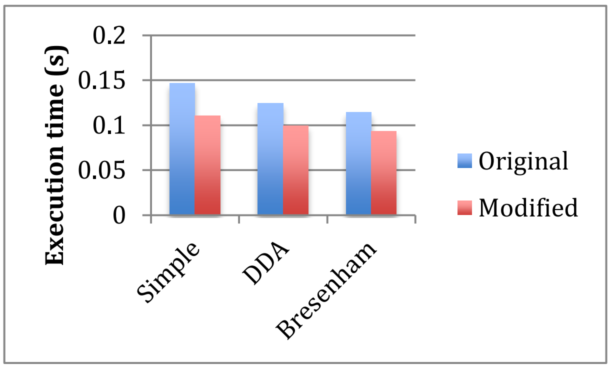

Figure 5: Comparison of the execution time of the classical and the new algorithm for line 2

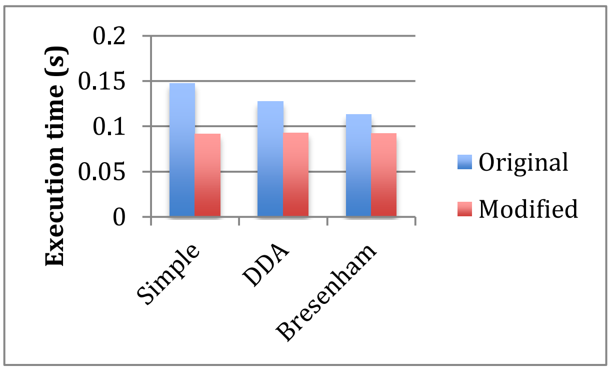

Figure 6: Comparison of the execution of the classical and the new algorithm for line 3

5. CONCLUSION

Other than devising new strategies for scan conversion of line, simply by exploiting the inherent multi symmetry property, we can enhance the process of line drawing to a certain extent.This approach can be applied to any newly discovered algorithm and the level of acceleration can be investigated. While a line has natural 2-way symmetry with respect to both the ends, the multi symmetry feature is affectively exploited for certain line orientations (end points). On the other hand a circle has a natural 8-way symmetry and an ellipse has a natural 4-way symmetry. It can be investigated whether there exists more symmetry for certain circle and ellipse orientation (center and radius for circle; center and axis's for ellipse) and also the corresponding probability. Then performance acceleration and also overall performance acceleration can by measuredfor these primitives when exploiting multi symmetry.

6. REFERENCES

[1] Roy, P. A., and Kaley, G., Fundamentals of Computer Graphics, Schaum's Outline Series, International Edition.

[2] Graham,R. L., Knuth, D. E., Patashnik, O., Concrete Mathematics, Addison-Wesley Publishing Company

[3] Bresenham, J. E., Algorithm for computer control of a digital plotter. IBM Systems Journal 1965; 4(1): 25–30.

[4] Bresenham, J. E., A Linear algorithm for incremental digital display of circular arcs. CACM 1977; 20(2): 100–6.

[5] Pitteway, M. L. V., Algorithm for drawing ellipses or hyperbolae with a digital plotter. Computer Journal 1967; 10(3): 282–9.

[6] Van Aken Jr., An efficient ellipse-drawing algorithm. CG&A 1984; 4(9): 24–35.

[7] Wu, X. and Rokne,J. G. , Double-Step Incremental Generation of Lines and Circles, Computer Vision, Graphics and Image Processing, 37: 331-334.

[8] Kappel, M.R., An ellipse-drawing algorithm for faster displays. Fundamental algorithms for computer graphics, Springer, Berlin, 1985, pp. 257–280,.

[9] Van Aken Jr, Novak, M., Curve-drawing algorithms for raster displays. ACM TOG 1985; 4(2): 147–69.

[10] Pang, A. T., Line-drawing algorithms for parallel machines. IEEE Computer Graphics and Applications 1990; 10(5): 54–9.

[11] Wright, W. E., Parallelization of Bresenham's line and circle algorithms. IEEE Computer Graphics and Applications 1990; 10(5): 60–7.

[12] Hasan, M. and Kashem, M. A., An Efficient Line Drawing Algorithm, Proceedings of ICCIT'99, pp. 204-207.

[13] Karmakar, C. K., Shams, S. M. S., and Rahman, M. S., Line Drawing Algorithm: A New Approach, SUST Studies, 2002, 4 (1): 65-69.

[14] Haque, A., Rahman, M. S., Bakht, M., Kaykobad, M., Drawing lines by uniform packing, *International journal of Computers and Graphics*, Elsevier, 30(1): 207-212, 2006

[15] Bond,C.,A New Line Drawing Algorithm Based on Sample Rate Conversion, http://www.crbond.com/papers/newline.pdf, last visited on 07-12-2012.

[16] Kabir, M. H., Hasan, I., and Azfar, A., An Improved Algorithm for Scan Converting a Line, Asian Journal of Information Technology 2005; 4 (9): 835-839

[17] Co-prime Integers, from Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Coprime_integers, last visited on 09-12-2012.

[18] Foley, J. D., Andries van F, Steven, K., Hughes J. F., Computer graphics principles and practice, 2nd ed. in C, Fourth Indian Reprint, 2000, Addison Wesley Longman, Singapore