

CRN 148 MECH 576
**Computer Graphics &
Geometric Modeling**

November 17, 2006

Still an Important Message

Evolution of Software & Bresenham's Algorithm

Paul Zsombor-Murray
Department of Mechanical Engineering
McGill University, 817 r. Sherbrooke o., Montréal, Qué., H3A 2K6
-&-
TU-Graz Inst. f. Mech.
A-8010 Graz, Kopernikusgasse 24, Austria
<paul@cim.mcgill.ca>

November 17, 2006

1 Introduction

Fourteen years ago I was invited to address a general meeting of *l'Ordre des Ingénieurs du Québec* and my lecture seems as relevant today, in connection with our “Mechatronics Project **II**”, as it was in 1985. It is nice to see that some things do not change. In this course, these *invariant principles* will be emphasized in the lectures and we will undergo training together, doing projects in the lab, ably assisted, in the details of applying the peculiar hardware and software at hand, by some younger and more up-to-date teachers.

2 The Lecture

It gives me great pleasure to have been chosen to speak on a subject on which I am somewhat ignorant. Nevertheless let us examine some facets of this huge, diverse and costly field.

Because in many ways the demarcation between micro- and large computers has become quite blurred, software and hardware issues that we face pertain to both. To see the problems that confront both the user-engineer and the software engineer, consider the themes of the major articles which appeared in a special issue of “Scientific American”, 84-09.

1. Computer software
2. Algorithms and data structures
3. Programming languages
4. Applications
5. Word processors and related software
6. Graphics software
7. Information management software
8. Process control software
9. Mathematical and scientific software
10. Software for intelligent systems

Other articles treated visual perception, finite state automata, error detection and correction codes, security, communication protocols and signal processing. Note that “Scientific American” is a popular interest magazine and in no way dedicated to informatics. In the same way, as engineers, we must stay current to exploit software advances peculiar to our field. Only in this way may we avoid the half-life degradation to which our training is inevitably exposed. In this way we may be equipped to do what engineers are supposed to do: Reduce the cost and enhance the value and reliability of goods and service that we design. Software is labour intensive and accounts for 80% of computational costs. It is here that the principal opportunities for economization are to be found. I feel a bit like Tom Leher who covered chemistry by singing a song composed on the recitation of Medele’ev’s periodic table. That is what happens if one tries to tell the whole story of software evolution in fifteen minutes.

How can software evolution make me a better engineer? Certainly not by replacing me with a button-pusher who has learned the liturgy of the latest CAD package. How can I find inspiration in this melée of progress before becoming entirely confused and frustrated? Take, for example, the subject of the sixth and eighth articles in the list above, “Process Control Software”. Say that I am a machine designer with little experience in microprocessors. What can I do?

1. Machines which have more than a few moving parts use some of these to transmit information, not power.
2. Fewer moving parts inevitably means simpler design and a cheaper more reliable product.
3. One must take the time to become familiar with a particular microprocessor chip at the hardware and machine language level and with various peripheral chips.
4. The next stage of familiarization involves an integrated version where the processor, memory and diverse peripherals are all included in a single chip. The investment in hardware, software and how-to instructions is much less than the cost of a PC and children cannot hijack it to play idiotic video-games.
5. This apparatus can now be effectively used to enhance the machinery to be designed whenever an opportunity to simplify and economize presents itself.
6. Soon microprocessor incorporation in machines will become as intuitively easy as the routine choice of gears, bearings and mechanical drives.

The development of this sort of hardware and software is clearly the rôle and responsibility of *all* engineers, not just informaticians. *Their* task is to devise new architectures, instruction sets and the rigorous principles of software design. *Ours* is the opportunity to adapt these to innovative and productive applications.

3 An Example

All which has just been stated is like “motherhood” and “apple-pie”, mostly platitude. Let us therefore examine the creativity pertinent to software for a specific application which is essentially mechanical; motion control. It is a simple example, the generation of straight line motion between two points in the Cartesian plane. This task is encountered in driving graphics displays, digital plotters and numerically controlled machine tools. Consider that all this is important in modern industry. One may feel that drawing a line is trivial; common and beneath our concern. Indeed it is common. So much so that a computer graphic or numerical control system spends much, if not most, of its time computing these trajectories. There is a lot of difference between the efficiency of good and not-so-good procedures and I think it is instructive and even interesting to describe a few variations on this theme.

3.1 The Environment

First of all, actually drawing a line involves working in integer space. The two points to be joined are given by their x - and y -coordinates. Second, the CRT beam, plotter stylus or cutting tool point is placed at the first point coordinates. Thirdly the line is described by stepping an increment at a time in the x - or y - or both directions until the second point is reached. Although cutting tool and stylus motion is slow compared to computational speed, the generation of a complicated image in real time may often exceed the capacity of the largest super-computer. Before describing some straight line algorithms(SLA) it is important to appreciate the conditions under which these were programmed and tested. The target CRT was a crude memory-mapped raster scan CRT. Its graphics capacity is limited to 64×44 coarse, easy-to-see pixels, one of which is darkened with the BASIC command PLOT X,Y. All the SLA's can be tested with a little stub of code that accepts the two point coordinates from the keyboard.

```
100 INPUT X1
110 INPUT Y1
120 INPUT X2
130 INPUT Y2
140 GOSUB 1000
150 GOTO 100
```

3.2 Quick & Dirty

Without further ado, here is the first SLA. It is called QUICK+DIRTY because it is short in code, but inefficient in execution.

```
990 REM *****
      QUICK+DIRTY STRAIGHT LINE ALGORITHM
      TAKES 258S FOR 100 LONGEST DIAGONALS:
      (0,0) ,(63,43)
      *****
1000 LET J=X2-X1
1010 LET K=Y2-Y1
1020 LET LL=SQR(J*J+K*K)
1030 IF LL>0 THEN GOTO 1060
1040 PLOT X1,Y1
1050 RETURN
```

```

1060 FOR S=0 TO LL
1070 PLOT X1+S*J/LL,Y1+S*K/LL
1080 NEXT S
1090 RETURN

```

Notice that the routine computes the length of the diagonal line subtended by J, the x -span, and K, the y -span. It does not ignore lines of zero length but puts a pixel at the required point. It computes x - and y -coordinates for each integer interval S along the diagonal using the following relationships.

$$\begin{aligned}
X &= X_1 + S \cos \theta \\
Y &= Y_1 + S \sin \theta
\end{aligned}$$

where

$$\cos \theta = J/LL$$

and

$$\sin \theta = K/LL$$

by definition. The line “grows” from (X1,Y1) to (X2,Y2) in the correct direction because the signs, *i.e.*, + or -, of J and K are correctly computed. What more could one desire? The entire code is only 11 statements. It takes care of lines in any direction and even those of zero length. Furthermore the kernel is the three statement loop 1060–1080 with a single PLOT statement in which all the functionally repetitive computations take place. Observe however that problem preparation needs two subtractions, two multiplications, an addition and a square-root. Also, two additions, multiplications and divisions are needed in each step.

3.3 In Search of Efficiency

Surprisingly a much more efficient procedure can be devised. It is easily understood if one imagines the plane divided into octants of line direction and if we consider the example of only the first where lines proceed in compass directions between east and north-east. Note that for every x -step, the longer of the two sides of the right-angle triangle whose hypotenuse is the desired straight line, an ideal y -step of

$$\frac{Y_2 - Y_1}{X_2 - X_1}$$

should be made. Since only unit steps are possible, these are invoked only when a sufficient number of ideal y -steps have accumulated, when $E \geq 0.5$. Thus we write the following code.

```

1000 LET J=X2-X1
1010 LET K=Y2-Y1
1020 LET R=K/J
1030 LET E=0
1040 LET Y=Y1
1050 FOR X=X1 TO X2
1060 IF E<0.5 THEN GOTO 1090
1070 LET Y=Y+1
1080 LET E=E-1
1090 PLOT X,Y
1100 LET E=E+R
1110 NEXT X
1120 RETURN

```

There is more code now, 13 *-vs-* 11 statements and the remaining seven octants have not been dealt with. *But* there are only two subtractions and a division in the preparation phase, lines 1000–1020, *and* there are only two additions and a subtraction in the iterative kernel, 1050–1110. Much better. Can this be improved? Yes, it is not necessary to normalize the test as $E < 0.5$. We could use $L = J/2$ instead of 0.5.

```

1020 LET L=J/2
.
.
.
1060 IF E<L THEN GOTO 1090
1070 LET Y=Y+1
1080 LET E=E-J
1090 PLOT X,Y
1100 LET E=E+K
.

```

This gets rid of the division to evaluate R in line 1020. Since the issue is evolution, what is the next step? A division by 2 is just a “shift-right” when working with binary integers, which computers do. If not, a division by 2 is just as bad as one by J . Even with integers, if the line to be drawn is short, a division by 2 will result in serious round-off error for odd values of J . This can be fixed by scaling up again, this time by a factor of 2, an integer “arithmetic shift-left”.

```

1010 LET K=2*(Y2-Y1)
1020 LET L=2*J
.
.
.
1060 IF E<J THEN GOTO 1090
1070 LET Y=Y+1
1080 LET E=E-L
1090 PLOT X,Y
1100 LET E=E+K
.

```

3.4 Putting It All Together

Good, it is just about perfected except for the seven missing direction octants. These are cleaned up in the complete listing below.

```

990 REM *****
      REVISED BRESENHAM STRAIGHT LINE ALGORITHM
      TAKES 223S FOR 100 LONGEST DIAGONALS:
      (0,0), (63,43)
      *****
1000 LET J=X2-X1
1010 LET S=SGN J
1020 LET K=Y2-Y1
1030 LET T=SGN K
1040 LET E=0
1050 LET J=ABS J
1060 LET K=ABS K
1070 LET G=2*J
1080 LET H=2*K
1090 IF K>J THEN GOTO 1190
      1100 LET Y=Y1
      1110 FOR X=X1 TO X2 STEP S
      1120 IF E<J THEN GOTO 1150
      1130 LET Y=Y+T

```

```

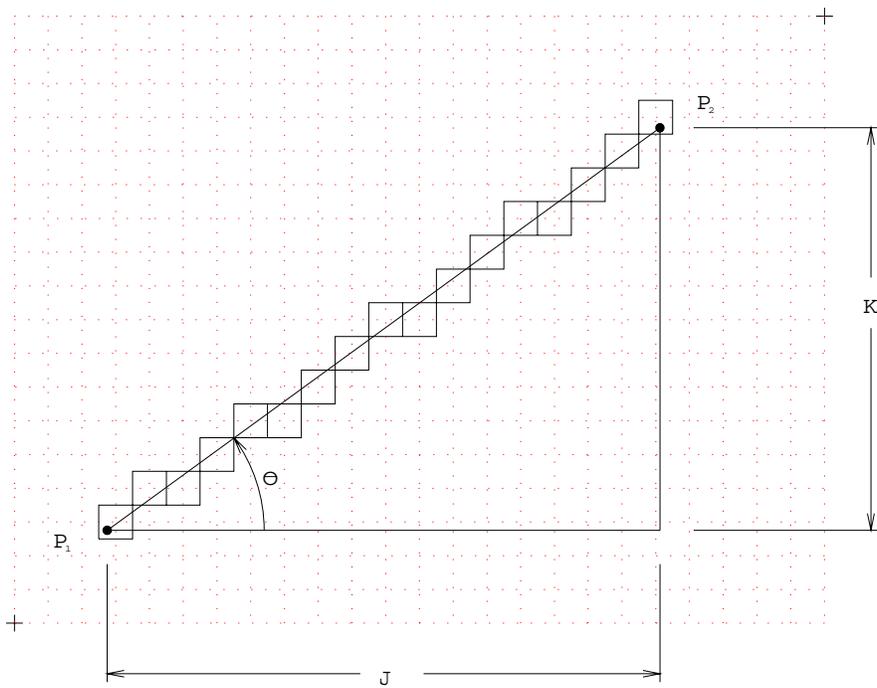
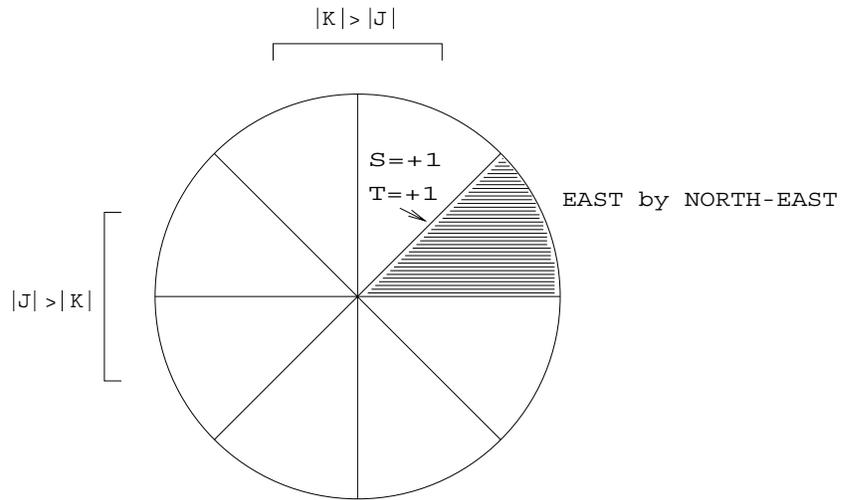
1140 LET E=E-G
1150 PLOT X,Y
1160 LET E=E+H
1170 NEXT X
1180 RETURN
      1190 LET X=X1
      1200 FOR Y=Y1 TO Y2 STEP T
      1210 IF E<K THEN GOTO 1240
      1220 LET X=X+S
      1230 LET E=E-H
      1240 PLOT X,Y
      1250 LET E=E+G
      1260 NEXT Y
      1270 RETURN

```

This routine takes 28 statements but only 19 are used to draw any given line. There are *no* divisions. The two multiplications by 2 are simply “shift-left” operations and all line directions are accommodated. The 19 statements execute 15% faster than the 10 in the “Quick & Dirty” even given the interpretive BASIC in which the tests were made. The advantage would be greater with lines of greater length than the $\sqrt{64^2 + 44^2} = 77$ pixel length to which we were limited. This margin would be *spectacular* with routines written in machine code or implemented in dedicated hardware.

4 Conclusion

The Bresenham Revised SLA is a classic; some decades old. However the development of similar principles applied to the efficient generation of approximations to curved elements in three dimensions is a very active field of research. Apart from obvious utility in machine design, progress here is vital to the planning of robotic manipulations. Fear not. There remain sufficient discoveries to be made in this area so that each and every one of us may enjoy fame and fortune in having a few stunningly elegant algorithms known to the world by our names . . .if we work hard and do our homework.



Bresenham's Algorithm, Octant & Pixel (Step) Selection

Figure 1: Octants and Steps