An Efficient Structural Technique for Encoding 'Best-fit' Straight Lines

C. M. A. CASTLE AND M. L. V. PITTEWAY
Computer Science Department, Brunel University, Uxbridge UB8 3PH

A structural algorithm is proposed which uses Euclid's algorithm to control two symmetric production rules which can construct the 'best-fit' incremental line. The output is identical to that produced by Bresenham's algorithm but is, in general, produced in fewer subtract operations. The correctness of the algorithm is established, and a formula conjectured for the behaviour of the subtractive version of Euclid's algorithm.

Received August 1985

1. INTRODUCTION

The increasing use of calligraphic output peripherals, raster scan devices and incremental plotters, has accelerated the search for efficient algorithms to produce 'best-fit' straight lines. On either a plotter or a raster scan device, each straight line must be approximated by a quantised sequence of diagonal (D) and/or square (S) movements, and the 'best-fit' is obtained by applying the constraint that after each incremental step, the pixel whose centre is nearest to the 'real' line is the one selected for illumination. The sequence of such moves, which constitute the 'best-fit' grid-based approximation to any particular line, is called chain coding.[1, 7, 11, 15]

There are three common error metrics which quantify the concept of 'nearest to the real line'. They involve minimising (1) the normal (i.e. perpendicular) distance to the true line, (2) the axial distance to the true line, or (3) the function residue. Bresenham[2] has analysed these cases, and the error metric of his algorithm[3] is to minimise the normal distance. However, it is easy to show that in the case of a straight line, all three metrics are equivalent. The normal distance error metric will be used throughout this paper, and it is formally stated in Section 5.

Using the above criterion, it is possible to select the appropriate pixel with either a floating-point differential analyser, or a form of Bresenham's integer-based algorithm.[17, 3] Sproull[16] has shown that Bresenham's algorithm can be derived from the differential analyser, thus establishing that both generate identical output strings.

A significant property of both of these algorithms is that, in general, they both require one add or subtract operation per output move.

2. ANALYSIS OF THE ENCODING OF GRID-BASED LINES

Even a cursory glance at the output of a 'best-fit' algorithm reveals a profound palindromic symmetry which is at first sight most surprising. If, for example, a line is drawn from (0, 0) to any pair of mutually prime coordinates (u, v) with $u > v > 0$, the output displays palindromic symmetry about a central element which is S or D if u is odd, but which becomes a doubleton when u is even. E.g. from (0, 0) to (23, 7) produces:

S D SS D SSS D SS D SS D SSS D SS D S
(Spaces added to improve readability)

If u and v are not themselves prime, but can be expressed as a prime multiple of a common factor, it is only necessary to construct the 'best-fit' output up to the point where the 'real' line passes through a pixel centre; thereafter the same pattern will be regenerated, and the regenerated pattern will itself show reflected symmetry. More generally, following the removal of any common factor, if the remaining numbers are not themselves prime, reflection may be about a double element.

The line from (0, 0) to (28, 6), for example, produces:

SS D SSS D SSSS D SSSS D SSS D SSSS D SS

This can be rewritten as:

SS D SSS (DS) SSS D SS | SS D SSS (DS) SSS D SS
A B C

B is the place where the 'real' line passes through point (14, 3); A and C are the double elements about which the substrings are symmetrical due to the line passing through positions (7, 1.5) and (21, 4.5).

The importance of any common factor to the form of the output has been noted by Earnshaw.[9] In 1982 Pitteway and Green[14] presented an algorithm which employed Euclid's algorithm to trap the common factors. These were then used to drive appropriate move concatenation techniques which improved the operational speed of Bresenham's algorithm, while still retaining its basic structure.

In this paper we show that the subtractive version of Euclid's algorithm, attributed to Nicomachus[10, 12] can be used to control the appropriate output sequence for an integer-based 'best-fit' generator.

3. CONTROLLING THE PRODUCTION RULES WITH EUCLID'S ALGORITHM

In his analysis of the grammar for a 'best-fit' algorithm, Brons[5] showed that the Freeman chain encoding of a straight line could be generated by the production rules of a context-sensitive grammar. However, the palindromic

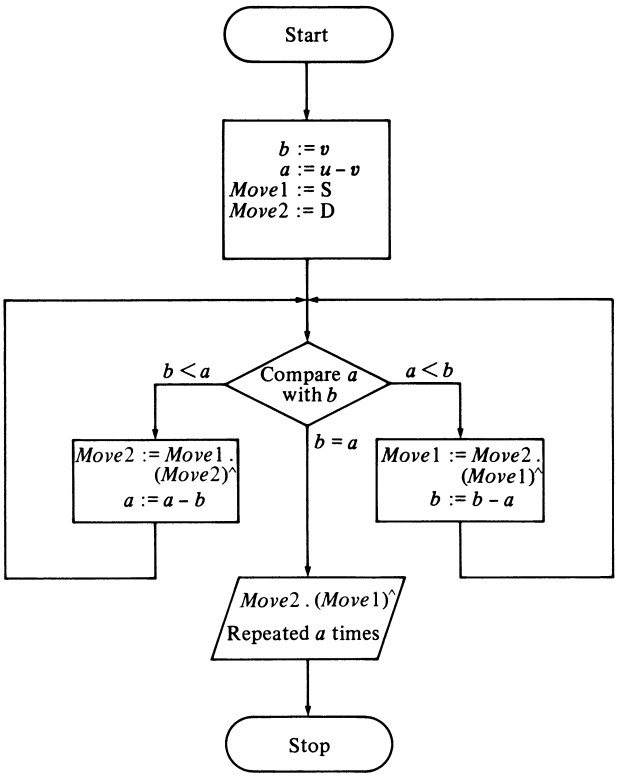

Fig. 1. The algorithm for generating 'best-fit' output, with Euclid's algorithm acting as a discriminator between two production rules

nature of the sentential form of the language intuitively suggested to us that any move concatenation technique controlled by Euclid's algorithm would require production rules of the form:

$$R := T.R^{\wedge},$$

where $T.R$ represents string T concatenated with string R, and $R^{\wedge}$ represents string R reversed; e.g. if $R :=$ SSSD, then $R^{\wedge} :=$ DSSS.

Fig. 1 shows the Castle–Pitteway algorithm,[6] which employs Euclid's algorithm as a discriminator between two symmetric production rules. *Move*1.(*Move*2)^ denotes the concatenation of *Move*1 with the reverse of string *Move*2; S is a square output move and D is a diagonal move. It is assumed that $u > v > 0$, the output in other octants being produced by symmetry. (The cases of $v = 0$, requiring u square moves, and $v = u$, requiring u diagonal moves, are trivial.) If a 'best-fit' line were to be drawn from (0, 0) to (u, v), with $u > v > 0$, the initial selection for a and b would be:

$b := v$ (the number of diagonal moves to be made)
$a := u - v$ (the number of square moves)

*Move*1 has an initial value of S; *Move*2 has an initial value D.

In the example shown in table 1, the complete 51-move 'best-fit' output stream was determined in just nine tests. If the selected values of u and v share a common factor, then the algorithm will generate the 'best-fit' output up to the point where the 'real' line passes through a pixel centre. This pattern is then to be repeated the highest common factor number of times.

The efficiency with which the algorithm computes the output for a line drawn to coordinates sharing a common factor is illustrated by the following example:

constructing the line from (0, 0) to (42, 28)

a	*b*	*Move*1	*Move*2
14	28	S	D
14	14	DS	D

OUTPUT (*Move*2).(*Move*1)^ repeated 14 times, giving:

DSD DSD DSD DSD DSD DSD DSD
DSD DSD DSD DSD DSD DSD DSD

4. SHOWING THE CORRECTNESS OF THE ALGORITHM

In order to understand the way in which the algorithm functions, it is revealing to have it produce output after each cycle. This involves including the ancillary instruction OUTPUT (*Move*2).(*Move*1)^, regardless of the result of the comparison between a and b.

Hence the algorithm not only computes the chain code for the line from (0, 0) to (51, 11), but also shows the other lines that are 'tacitly' constructed in the process. The output lines that have been produced all have gradients which are terms in the Farey Series F(51).[13] The Farey series of order N is defined as the ascending series of irreducible fractions between 0 and 1 whose denominators do not exceed N. For example:

F(6) is {0/1, 1/6, 1/5, 1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 4/5, 5/6, 1/1}.

Clearly, such a sequence is monotonically increasing.

Table 1. The output of the algorithm in constructing the line from (0, 0) to (51, 11)

a	*b*	*Move*1	*Move*2
40	11	S	D
29	11	S	SD
18	11	S	SDS
7	11	S	SSDS
7	4	SSDSS	SSDS
3	4	SSDSS	SSDSSSDS
3	1	SSDSSSDSSSSDS	SSDSSSDS
2	1	SSDSSSDSSSSDS	SSDSSSDSSSSDSSSSDSSSDS
1	1	SSDSSSDSSSSDS	SSDSSSDSSSSDSSSSDSSSDSSSSDSSSDSSSDS

OUTPUT (*Move*2).(*Move*1)^ just once (since $a = 1$), giving:
SS D SSS D SSSS D SSSS D SSS D SSSS D SSSS D SSS D SSSS D SSSS D SSS D SS

Table 2. The intermediate lines 'tacitly' constructed by the algorithm

a	b	'Tacit' line output
40	11	(2, 1)
29	11	(3, 1)
18	11	(4, 1)
7	11	(5, 1)
7	4	(9, 2)
3	4	(14, 3)
3	1	(23, 5)
2	1	(37, 8)
1	1	(51, 11)

Thus, for three successive terms $T(a)$, $T(b)$ and $T(c)$ in F(N), $T(a) < T(b) < T(c)$, and there can be no intervening term with denominator $< N$.

The ancillary lines that have been 'tacitly' created by the algorithm have gradients which are all terms in the continued fraction expansion of v/u. However, they do not all correspond to 'proper' convergents, as specified by the following important relationships between continued fraction expansion and the Farey series.

If $q(k)$ is the denominator of the kth convergent $C(k)$ of the simple continued fraction $\{a(0); a(1), \ldots, a(n)\}$, then $q(k-1) \leqslant q(k)$ for $1 \leqslant k \leqslant n$, with strict inequality when $k > 1$. The convergents with even subscripts form a strictly increasing sequence; that is: $C(0) < C(2) < C(4)\ldots$ The convergents with odd subscripts form a strictly decreasing sequence; that is: $C(1) > C(3) > C(5)\ldots$

Every convergent with an odd subscript is greater than every convergent with an even subscript.

5. LINE CONSTRUCTION LEMMAS AND BRESENHAM'S EQUAL ERROR ANOMALY

The error metric used by the Castle–Pitteway algorithm is that inherent in Bresenham's. It has been formally stated by Bresenham[4] as: '...For lines with a first octant orientation relative to an origin at the line segment's integer start point:

$$\text{line: } y = (v/u)\,x$$

raster segment: from (0, 0) to (u, v) with u and v integer $u > v > 0$ by selecting unit axial (S: square) and unit diagonal (D: diagonal) steps connecting integer grid mesh points given as:

$$X_i = I \text{ FOR } I = 0, \ldots, u$$
$$Y_i = \text{floor: } y_i = ((v/u)\,I) + 0.5\text{'}$$

The equal error anomaly arises whenever the 'real' line is equidistant from two consecutive vertical grid mesh points. This happens whenever u is even and v is odd. On these occasions the error metric will, by convention, select the upper point. Thus the chain code for the line from (0, 0) to (2, 1) is DS, although SD would be as visually acceptable.

In their work on establishing a 'recogniser' for the chain code associated with approximations to straight lines, Dorst and Duin[8] specified the conditions whereby lines drawn to different coordinates may share the same chain code. However, their work was developed for the case of 'Object Bound Quantisation', as opposed to 'Grid Bound Quantisation', and takes no account of the equal error anomaly inherent in Bresenham's error metric. The following lemmas are a generalised extension of Dorst and Duin's work to grid-bound quantisation, and do account for the equal error anomaly.

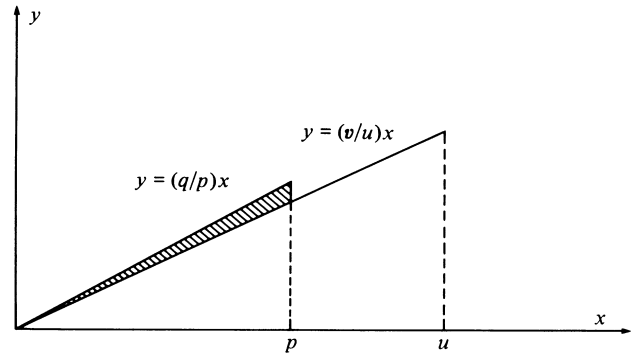

Fig. 2. q/p is the smallest upper bound on v/u in $F(u)$

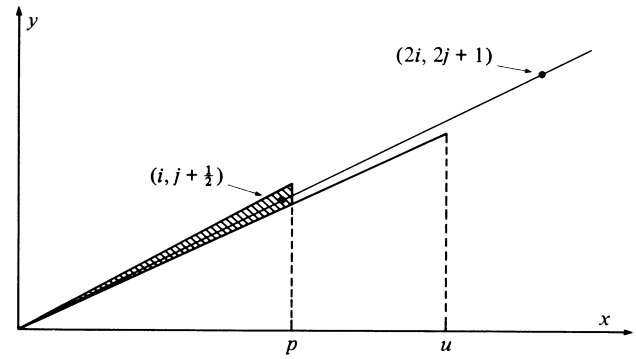

Fig. 3. The shaded triangle does not contain point $(i, j+\frac{1}{2})$

Lemma 1

If the chain code for the p elements of the line $y = (q/p)\,x$ is $L(p, q)$, and q/p is the smallest upper bound to v/u in $F(u)$, then the first p elements of the chain code for the line $y = (v/u)\,x$ will be $L(p, q)^{\wedge}$ (Assuming q/p and v/u are in reduced form, and that $q < p < u$).

Proof

Because (q/p) is the smallest upper bound on v/u in $F(u)$, there can be no pixel centre lying within the shaded triangle of Fig. 2 (otherwise any such pixel centre would itself be the smallest upper bound in $F(u)$).

The shaded triangle in Fig. 2 cannot contain any point that lies midway between two pixel centres each with the same x coordinate. If such a point existed it would have coordinates $(i, j+1/2)$ where i and j are integers with $0 < i \leqslant p$, and $v/u < j/i < q/p$. This would imply the existence of a pixel centre with coordinates $(2i, 2j+1)$ as shown in Fig. 3. Since $(2j+1)/2i$ is in reduced form, then $2i > u$, otherwise $(2j+1)/2i$ would itself be the smallest upper bound, and this a contradiction. Now, since v/u and q/p are consecutive terms in $F(u)$, the only term that can lie between them in $F(u+p)$ is $(v+q)/(u+p)$. This term is not $(2j+1)/2i$ since $(u+p) > 2i$. Thus $(i, j+1/2)$ does not lie in the shaded triangle.

Thus if p is odd, all the square moves that were appropriate to the sequence for $L(p, q)$ will be at least as appropriate to the first p moves of $L(u, v)$. (This is because the error in the square moves is reduced by the downward displacement of the lower line.) Since, by the above proof, there are no equidistant points in the shaded triangle, all the diagonal moves will also be equally appropriate.

However, if p is even, the line $y = (q/p)x$ must pass equally between two pixel centres. In this particular case, that of the equal error anomaly, Bresenham's algorithm will select the upper pixel. Nevertheless, any downward displacement from the line $y = (q/p)x$, however small, must result in the new line passing below this equidistant position. Thus the selection of the diagonal move in $L(p, q)$ is not the 'best-fit' selection for $L(u, v)$. To correct this anomalous error, all that needs to be done is to reverse the chain code. Due to the palindromic symmetry of the 'best-fit' chain code, this reversal still generates the appropriate move sequence even if the equal error anomaly has not occurred.

Thus the first p element of $L(u, v)$ is $L(p, q)\hat{}$.

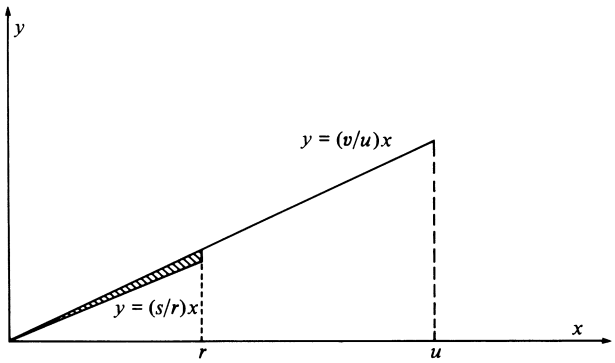

Fig. 4. s/r is the greatest lower bound on v/u in $F(u)$

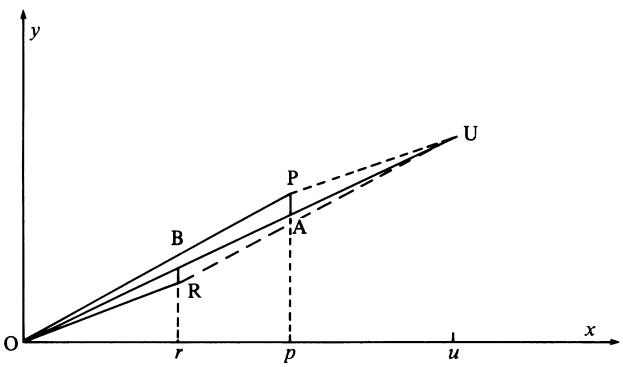

Fig. 5. $L(u, v) = L(p, q)\hat{}.L(r, s)\hat{}$

Lemma 2

If the chain code for the r elements of the line $y = (s/r)x$ is $L(r, s)$, and s/r is the greatest lower bound to v/u in $F(u)$, then the first r element of the chain code for the line $y = (v/u)x$ will be $L(r, s)$.

(assuming $s < r < u$ and that s/r and v/u are in their reduced form)

Proof

This situation is in many ways the mirror image of lemma 1. As before, there is no pixel centre in the shaded triangle in Fig. 4 since s/r is the greatest lower bound to v/u in $F(u)$.

The upward displacement of $y = (v/u)x$ from $y = (s/r)x$ implies that the diagonal moves for $L(r, s)$ are still the 'best-fit' for the first r moves of $L(u, v)$. A similar argument as before obtains to the positioning of the square moves, since no point with coordinates $(i, j+1/2)$ can exist in the shaded triangle. The equal error anomaly does not occur in this case, because the upward displacement of $y = (v/u)x$ now makes the equal error diagonal move in $y = (s/r)x$ the correct move.

Thus the first r element of $L(u, v)$ is $L(r, s)$

Table 3. The move sequences generated in constructing lines to Fibonacci co-ordinates

Branch	*Move*1	*Move*2	'Tacit' line drawn
(a)			
	S	D	(2, 1)
Right	DS	D	(3, 2)
Left	DS	DSD	(5, 3)
R	DSDSD	DSD	(8, 5)
L	DSDSD	DSDSDDSD	(13, 8)
R	DSDSDDSDDSDSD	DSDSDDSD	(21, 13)
L	DSDSDDSDDSDSD	DSDSDDSDDSDSDDSDDSDSD	(34, 21)
	DSDSDDSDDSDSDDSDDSDSDDSDSDDSDDSDSD	DSDSDDSDDSDSDDSDDSDSD	(55, 34)
(b)			
	S	D	(2, 1)
Left	S	SD	(3, 1)
Right	SDS	SD	(5, 2)
L	SDS	SDSDS	(8, 3)
R	SDSDSSDS	SDSDS	(13, 5)
L	SDSDSSDS	SDSDSSDSSDSDS	(21, 8)
R	SDSDSSDSSDSDSSDSSDSDS	SDSDSSDSSDSDS	(34, 13)
L	SDSDSSDSSDSDSSDSSDSDS	SDSDSSDSSDSDSSDSSDSDSSDSDSSDSSDSDS	(55, 21)

Lemma 3

If q/p and s/r are the least upper and greatest lower bounds on v/u in $F(u)$, then the chain code to construct the line $L(u, v)$ is given by:

$$L(u, v) = L(p, q)^{\wedge}.L(r, s)^{\wedge}$$

Proof

Since q/p and s/r are consecutive terms in either $F(p)$ or $F(r)$, it follows that $(q+s)/(p+r)$ is their median term in $F(p+r)$. Therefore, the parallelogram OPUR in Fig. 5 does not contain any pixel centre position. Thus the required chain code is (L(O to A).L(A to U)). Now L(O to A) = $L(p, q)^{\wedge}$ (from Lemma 1), and L(A to U)$^{\wedge}$ is L(U to A)$^{\wedge}$. Since L(U to A) is L(O to B), then by Lemma 2, this gives the required result:

i.e. $$L(p+r, q+s) = L(p, q)^{\wedge}.L(r, s)^{\wedge}$$

Furthermore, it is clearly the case that the necessary relationship between q/p, s/r and v/u is exactly that of consecutive terms in a Farey series, with the expected property that $v/u = (q+s)/(p+r)$.

6. THE SPECIFIC CASE OF LINES DRAWN TO FIBONACCI COORDINATES

The behaviour of the algorithm is easiest to understand when lines are drawn to coordinates which are themselves adjacent, or semi-adjacent terms in a Fibonacci sequence. This is because control passes alternately down the left and right branch of the algorithm, so the effect of Euclid's test can be excluded from consideration of the algorithm's operation.

There are two possible move sequences to be considered. The sequence selected depends upon the line coordinates being adjacent or semi-adjacent terms in the Fibonacci sequence. If they are adjacent terms, control passes first to the right branch of the algorithm and D moves predominate. If they are semi-adjacent, control passes first to the left branch and S moves predominate. Typical examples of each case are: (*a*) (0, 0) to (55, 34) and (*b*) (0, 0) to (55, 21) as shown in Table 3.

Applying the Castle–Pitteway algorithm to any of the 'tacit' lines shown in Table 3 demonstrates a revealing recursive relationship. For the line (0, 0) to (34, 21), for example, the algorithm gives:

$$L(34, 21) = M2(34, 21).M1(34, 21)^{\wedge}$$

where $M1(34, 21)$ and $M2(34, 21)$ represents the strings *Move*1 and *Move*2, and the constructed string, $L(34, 21)$, represents the chain code for the line (0, 0) to (34, 21).

However, it follows from the algorithm, and is shown in Table 3, that $M1(34, 21) = M1(21, 13)$. Thus applying the algorithm to the way in which $M1(21, 13)$ was produced:

$$M1(21, 13) = M2(13, 8).M1(13, 8)^{\wedge}$$

i.e. $$M1(21, 13) = L(13, 8)$$

similarly

$$\mathrm{M2}(34, 21) = \mathrm{M1}(21, 13).\mathrm{M2}(21, .13)^{\wedge}$$

i.e. $$M2(34, 21) = L(21, 13)^{\wedge}$$

Thus $$L(34, 21) = L(21, 13)^{\wedge}.L(13, 8)$$

Applying this analysis to other Fibonacci lines shows that:

If n is even then $L(n+1) = L(n-1)^{\wedge}.L(n)^{\wedge}$
If n is odd then $L(n+1) = L(n)^{\wedge}.L(n-1)^{\wedge}$

The different cases for odd and even n are produced by virtue of the previous observation that convergents with odd subscripts are greater than convergents with even ones. However, due to the relationship between Fibonacci numbers and any Farey sequence containing them, this recursion is always equivalent to the statement:

L(next Fibonacci coordinate) =
L(least upper bound)$^{\wedge}$.L(greatest lower bound)$^{\wedge}$,

since the next Fibonacci coordinate = least upper + greatest lower. (This is apparent from any example.)

For example, in $F(55)$
8/21 21/55 and 13/34 are consecutive terms
21/34 34/55 and 13/21 are consecutive terms.

Thus the algorithm is equivalent to Lemma 3 and can always be guaranteed to produce the correct chain code for lines whose coordinates are adjacent or semi-adjacent terms in a Fibonacci sequence.

7. CONSTRUCTING CHAIN CODE FOR LINES DRAWN TO GENERAL COORDINATES (u, v)

The initial conditions in the Castle–Pitteway algorithm include the instructions 'set *Move*1 to S' and 'set *Move*2 to D'. *Move*1 and *Move*2 always contain the two vectors which constitute the greatest and lowest bounds so far located. They are, at this stage, the lines $L(1, 0)$ and $L(1, 1)$. All lines constructed by the algorithm lie between these gradients.

In the general case of the line from (0, 0) to (u, v), with $u > v > 0$, it is necessary to consider the control sequence that is imposed upon the operation of the algorithm by the Euclid test. The process of finding the highest common factor, using the subtractive version of Euclid's algorithm, is intimately connected with the continued fraction expansion of the two integers.

The simple continued fraction expansion of 11/51, for example, gives:

$$11/51 = \{0; 4, 1, 1, 1, 3\}$$

The continued fraction expansion not only expresses increasing approximation to the original fraction, but also relates directly to the number of times control must circulate around any one branch of the algorithm before being transferred to the other. This becomes apparent when considering the continued fraction expansion of the Fibonacci golden ratio:

$$\phi = \{1; 1, 1, 1, 1, 1, 1, 1, 1, 1, \ldots\}$$

When the values of (u, v) are consecutive terms in a Fibonacci sequence, control transfers alternately between the two branches. This effect is determined by the series of 1s in the expansion.

Thus, in the case of (51, 11), the expansion is $[0; 4, 1, 1, 1, 3]$, which in its turn gives rise to the fractions:

$$1/4,\ 1/5,\ 2/9,\ 5/23 \text{ and } 11/51$$

These are the 'proper' continued fraction convergents, in that within the appropriate Farey series they fulfil the criterion of being alternately best upper and lower bounds to 11/51. However, other convergents can be obtained from the series and these are:

1/1, 1/2, 1/3, 1/4, 1/5, 2/9, 3/14, 5/23, 8/37 and 11/51

Expressing the sequence in its decimal representation (to 3 places) gives:

1.000, 0.500, 0.333, 0.250, 0.200, 0.222, 0.214, 0.217, 0.216, 0.215

The Castle–Pitteway algorithm creates legitimate lines by invoking Lemma 3. This requires locating best upper and lower bounds. Now examination of the sequence of continued fractions shows that the first five numbers are all in descending order. Now the algorithm must use the best lower bound found so far in the line construction process. This is preserved in *Move*1 as S, and represents the line $L(1, 0)$. Thus, when the algorithm is cycling around the same loop, the initial state of one of the moves is preserved. The contents of this move is the chaincode for the best bound located so far. Hence, during the first few cycles of the algorithm, Lemma 3 is tacitly invoked to construct:

$L(2, 1) = L(1, 1)\hat{}.L(1, 0)\hat{}$ (D.S)
$L(3, 1) = L(2, 1)\hat{}.L(1, 0)\hat{}$ (SD.S)
$L(4, 1) = L(3, 1)\hat{}.L(1, 0)\hat{}$ (SDS.S)
$L(5, 1) = L(4, 1)\hat{}.L(1, 0)\hat{}$ (SSDS.S)

In each case, by Lemma 3 this corresponds to the reversal of the chaincode for the least upper bound line concatenated with the reversal of that for the greatest lower bound line.

At this point the continued fraction expansion dictates a transfer to the other branch. This manifests itself directly by the variable a now being less than b, but it corresponds to reaching the end of the effect determined by the 4 in the continued fraction expansion.

Having located a greatest and lowest bound, the algorithm next transfers control to the other branch, and by Lemma 3 the line (9, 2) is correctly constructed ($\{1+1\}/\{4+5\} = 2/9$). Control is transferred again, and by Lemma 3 the line (14, 3) is constructed ($1/5 < 3/14$ whereas $2/9 > 3/14$). This process continues, with control reversing, as in the Fibonacci case, until the effect of the 3 in the continued fraction expansion is felt. This occurs in the creation of the last few lines. When $L(14, 3)$ has been constructed, variable a is 3 and b is 1. Thus *Move*1 preserves its value, which is the chaincode for $L(14, 3)$, and $L(23, 5)$ is constructed. At this point b is still less than a, so *Move*1 still contains $L(14, 3)$ and $L(37, 8)$ is constructed, then a and b are equal and so $L(51, 11)$ is constructed from $L(37, 8)$ and the contents of *Move*1, which is $L(14, 3)$. Hence $L(23, 5)$ is effectively by-passed.

Examination of the gradients shows that 3/14 is the greatest lower bound to 11/51, but since $8/37 < 5/23$, 8/37 is the smallest upper bound to 11/51, hence by Lemma 3, the algorithm has constructed the correct chaincode for $L(51, 11)$.

We have shown that the algorithm operates by invoking the proven procedure of Lemma 3 to construct lines. The application of this lemma is determined by a control process governed by Euclid's algorithm. When the algorithm is cycling around the $b < a$ branch, *Move*1 holds the chaincode for the line of greatest lower bound so far located. When the algorithm cycles around the other branch, *Move*2 holds the chaincode for the smallest upper bound line so far located. The particular branch selected at any step is determined by the values of a and b, and these two variables are direct counterparts of the continued fraction expansion. Thus the algorithm is guaranteed to produce identical output to Bresenham's algorithm or any other version of the Digital Differential Analyser.

It is worth noting that the Castle–Pitteway algorithm can be reformulated in several different ways. It is, for example, possible to avoid the process of reversing strings by employing two extra variables. These are used to replicate the contents of *Move*1 and *Move*2. Another useful reformulation can be obtained by replacing Nicomachus' subtractive version of Euclid's algorithm by Euclid's original division version. This can lead to considerable saving where efficient implementation of MOD and DIV operations is available.

8. DEVELOPMENTS IN THE ANALYSIS OF EUCLID'S ALGORITHM

Euclid's algorithm is the oldest-known documented algorithm. In its original version,[10] it appears as a technique for determining the highest common factor (or greatest common divisor), of two positive integers. The following program is a Pascal version of the algorithm:

```
program euclid (input, output);
var a, b: integer;
begin
  readln (a, b);
  while b <> a do begin
    if b < a then a:= a-b else b:= b-a
  end;
  writeln ('hcf is ', a);
end.
```

Comparison with Fig. 1 shows the control structure that Euclid's algorithm imposed upon the line constructor mechanism.

It is apparent that the Castle–Pitteway algorithm, in comparison with Bresenham's technique, will require less subtractions to generate the chain code for a given straight line. Therefore, the subtractive operation is in itself worthy of more detailed analysis. The following section presents interesting conjectures which arose from empirical observations of lengthy computer runs.

In generating the move sequence for a line of u increments drawn from some starting point (0, 0) to the point (u, v) with $0 < v < u$, Bresenham's algorithm will require a total of u subtractions. (The cases of $v = 0$ or $v = u$, requiring either all square or all diagonal moves, are trivial.) For the new algorithm, specific cases are first considered. Then the average behaviour for a given value of u, with v uniformly distributed in the range $1 < v < u$ is considered and then finally the double average over all values of u and v less than some given n.

First, for a given value of u, the worst case occurs at $v = 1$ or $v = u-1$, when it takes a total of $u-1$ subtract operations before v and u are equal. In this case there is no saving over Bresenham's algorithm. On the other

hand, the best case for u and v mutually prime occurs when u and v are adjacent or semi-adjacent terms in a Fibonacci sequence. The number of subtract operations involved is then about $2.078 \ln u$.

In computing the number of subtract operations, the numerical work indicated that it was necessary to include the case $v = u$ (which requires no subtracts), even though the algorithm as presented in Fig. 1 would fail. Thus the total sum over the range $1 \leqslant v \leqslant u-1$ was formed, but was divided by u rather than $u-1$ to obtain the required 'average'. This was found to be a more convenient fit to data for small values of u. Note, too, that there is a useful symmetry about the midpoint $u/2$: the number of subtracts required for given u and v is the same for u and $u-v$. (Care must be taken to handle the case of u even.)

The worst case now occurs when u is prime, for there will be no common factor for any value of v, and the algorithm must always run down until $v = u = 1$. Computer analysis, including a run with $u = 129434759$, leads to the conjecture that the average number of subtracts required for prime u behaves as:

$$\frac{6}{\pi^2}\{(\ln u)^2 + 2.5887 \ln u - 2.5075\}$$

The leading coefficient, $6/\pi^2$, has been established analytically by Yao and Knuth,[18] though they also suggest the possibility of subsequent terms of order $(\ln u) * \ln(\ln u)$. The conjecture presented here involves no subsequent term stronger than $\ln u$, and also postulates an exact value in place of the empirically determined 2.5887 coefficient. Moreover, we suggest that for composite u, the coefficient in $\ln(u)$, which is 2.5887 for u prime, should be reduced by two times a sum involving 'Von Mangoldt' functions, as described by Knuth in the analysis of the 'divide with remainder' version of Euclid's algorithm.[12] Thus when u is a prime multiple of 2, the coefficient 2.5887 multiplying the term in $\ln u$ is to be reduced by $\ln 2$; when u is a prime multiple of 3, it is to be reduced by $2(\ln 3)/3$, and, in general, it is to be reduced by:

$$2 \sum_{d \backslash u} \wedge(d)/d.$$

For example, if u is a prime multiple of 100, then the coefficient of $\ln u$ is reduced from 2.5887 to

$$2.5887 - 2\{\ln(2)/2 + \ln(2)/4 + \ln(5)/5 + \ln(5)/25\}.$$

(Our numerical studies indicate, surprisingly, that the last term, involving the prime itself, thus ...(ln (prime))/ prime, should not be included). Some further examples are listed in Table 4.

Our conjecture is that if u is an integer multiple of a prime number, then the average number of subtract operators is given by:

$$\frac{6}{\pi^2}\{(\ln u)^2 + (2.5887 - 2 \sum_{d \backslash u} \wedge(d)/d) \ln u + \text{constant}\} + O\left(\frac{\ln u}{u}\right)$$

The constant coefficient also requires some adjustment in each case, with examples again given in Table 4, but we offer no conjecture concerning these.

The conjecture upon the introduction of the Von Mangoldt terms into the coefficients of $\ln u$, for composite u, has been checked for all values of u to 70,000, for $u = 599{,}946$ and 1,373,766 (both prime multiples of 6), and for $u = 6{,}291{,}429$ (a prime multiple of 3).

Table 4. Typical values associated with data fitting the conjectured formula

Prime multiple	$\ln u$ term	Constant
2	1.8956	−2.68
3	1.8563	−2.25
4	1.5490	−2.29
5	1.9449	−1.79
6	1.1632	−2.17
7	2.0327	−1.60
8	1.3757	−1.85
9	1.6122	−1.63
10	1.2518	−1.74
20	0.9052	−1.24
30	0.5194	−0.99
40	0.7319	−0.74
50	1.1230	−1.13
60	0.1728	−0.36
120	−0.0005	+0.20

The formula for the average number of operations required by the divide form of Euclid's algorithm, is quoted by Knuth[12] to be:

$$\frac{12}{\pi^2} \ln 2\{\ln u - \sum_{d \backslash u} \wedge(d)/d\} + 1.47$$

Apparently the 1.47 should really be 1.4670780794..., which is given as:

$$\frac{6}{\pi^2} \ln 2\left\{3 \ln 2 + 4\gamma - \frac{24}{\pi^2}\zeta'(2) - 2\right\} - \tfrac{1}{2}$$

Thus the analysis of the divide version of Euclid's algorithm can be written as:

$$\frac{12}{\pi^2} \ln 2\left\{\ln u - \sum_{d \backslash u} \wedge(d)/d + 1.5 \ln 2 + 2\gamma - \frac{12}{\pi^2}\zeta'(2) - 1\right\} - \tfrac{1}{2}$$

Comparing this with the empirical analysis of the subtract algorithm in a completed square form (in order to avoid the 2 multiplying the Von Mangoldt sum), suggests making the comparison between our 2.5887 and

$$3 \ln 2 + 4\gamma - \frac{24}{\pi^2}\zeta'(2) - 2 - \frac{\pi^2}{12 \ln 2}$$

If the terms '$3 \ln 2$', and '$\pi^2/12 \ln 2$' are dropped, a likely match is obtained, so we conjecture that the 2.5887 coefficient should really be:

$$4\gamma - \frac{24}{\pi^2}\zeta'(2) - 2 = 2.588706632...$$

For the record, the 'best' empirically determined value was in fact 2.58867, but the conjectured value lies within the range of experimental error.

It is also worth noting that the empirical fit is a good one. The residue for prime u, for example, changes sign at $u = 47$, and there are 33 changes of residual sign as u increases to 373.

Finally, both Knuth[12] and Yao and Knuth[18] have suggested an average analysis of the subtract version of Euclid's algorithm for all u, v less than some given n. As Knuth observes, the scatter is large. However, computer

analysis with runs up to $u = 78000$, accumulating the total number of subtracts required for all $u \leqslant n$, then dividing by $\frac{1}{2}n(n+1)$, suggests an average behaviour like:

$$\frac{6}{\pi^2}\left\{(\ln n)^2 + 0.449 \ln n - 1.23\right\}$$

Acknowledgements

The authors are grateful to Anne Shrimpton of Brunel University, for programming assistance, to Dr Rick Thomas of St Mary's College Twickenham, for mathematical discussions, and to Dr Jack Bresenham (IBM) for his many detailed and constructive comments as referee.

REFERENCES

1. C. Archelli and A. Massarotti, On the parallel generation of straight digital lines. *Computer Graphics and Image Processing* **7** (1), 67–83 (1978).
2. J. E. Bresenham, An incremental algorithm for digital plotting. *Proceedings of the ACM National Conference* (1963).
3. J. E. Bresenham, Algorithm for computer control of a digital plotter. *IBM Systems Journal* **4** (1), 25–30 (1965).
4. J. E. Bresenham, Private Communication (1985).
5. R. Brons, Linguistic methods for the description of a straight line upon a grid. *Computer Graphics and Image Processing* **3** (1974), 48–62.
6. C. M. Castle and M. L. V. Pitteway, An application of Euclid's algorithm to drawing straight lines. *Proc. NATO ASI on Fundamental Algorithms for Computer Graphics*. Springer-Verlag, Heidelberg (1985).
7. R. L. Cederburg, A new method for vector generation. *Computer Graphics and Image Processing* **9**, 183–195 (1979).
8. L. Dorst and R. Duin, A framework for calculations on digitised straight lines. *IEEE Trans. Pattern Analysis and Machine Intelligence PAMI-6* (5) (1984).
9. R. A. Earnshaw, Line tracking with incremental plotters. *Computer Journal* **23** (1) (1980).
10. Euclid, *Elements* (Book 7). *Proc. Platonic Academy* **4009** (348 B.C.).
11. H. FREEMAN, Boundary encoding and processing. In *Picture Processing and Psychopictorics*, pp. 241–266. Academic Press, New York (1970).
12. D. Knuth, *Seminumerical Algorithms*, 2nd ed. Addison–Wesley, New York (1973).
13. J. G. Michel, Rational approximation. *Bulletin Inst. Maths. and its Applications* **12** (2), 44–46 (1976).
14. M. L. V. Pitteway and A. Green, Bresenham's algorithm with run-line coding shortcut. *Computer Journal* **25** (1), 114–115 (1982).
15. A. Rosenfeld, Digital straight line segments. *IEEE Transactions on Computers* **C-23 12**, 1264–1269 (1974).
16. R. F. Sproull, Using program transformations to derive line-drawing algorithms. *ACM Transations on Graphics* **1** (4) (1982).
17. F. G. Stockton, Algorithm 162 *CACM* **6** (4) (1963).
18. A. C. Yao and D. E. Knuth, Analysis of the subtractive algorithm for greatest common divisors. *Proc. Nat. Acad. Sci. USA* **72** (12), 4720–4722 (1975).