DIGITIZED SEGMENTS AND STRAIGHT LINES

E. Domínguez°, A. Francés°, A. Márquez*

° Dept. de Ingeniería Eléctrica e Informática
Facultad de Ciencias
Universidad de Zaragoza
50.009 Zaragoza (España)

* Department of Computer Science
Brown University
Providence
Rhode Island 02912-1910 (USA)

Abstract. The goal of this paper is to introduce a code, called stair code, to represent digital segments. We present a new incremental algorithm for drawing segments which is related to this code. We compare it with Bresenham's algorithm and observe that the outputs of these two algorithms give us an appropriate definition of digital segment.

Key words: Digital Geometry, digital representation, encoding, digitization, digital segment, digital straight line.

§1. Introduction. The aim of Digital Geometry is to find the most appropriate data structures and algorithms for solving geometric problems efficiently. Each data structure that encodes a given digital object requires a different algorithm to solve a particular problem. Consequently, to solve a problem, one structure may be better than another depending on its space complexity and the time complexity of its associated algorithm.

The task of finding a suitable code to process digital objects is, therefore, of great importance. The aim of this paper is to present a code for processing segments, called stair code, comparing it with other codes by means of two basic problems. These problems are the drawing of a segment, which will be called DRAWING problem and the question of deciding whether a digital point belongs to a segment which will be called BELONG problem.

The most elemental code for encoding segments is to give two points which are the segment end points. This code, which we will call *minimal code*, has a constant space complexity and it is well known that, using it, Bresenham's algorithm [1] solves the DRAWING problem in optimal time, since it is of the order of the number of pixels to be illuminated. The BELONG problem can also be solved in constant time by using integer divisions and products.

Since the number of pixels that must be illuminated in order to draw a segment on a computer screen is finite, we can interpret the enumeration of these pixels as a code that

we will call *the complete code*. This code is the structure that contains most explicit information on the segment but its storage cost is as great as the number of pixels that make the segment up. The algorithm for the DRAWING problem is trivial using this code; illuminating the pixels contained in the code is enough and its complexity has the same order as the Bresenham's algorithm complexity. The BELONG problem can be solved in constant time using sums and comparisons only if the code is supported by a suitable data structure.

A more accurate structure, which is already classical, for representing digital objects is the chain code [9]. The *chain code* of a segment consists of one point (one of its end points) and a sequence of symbols that represent horizontal and diagonal movements. These movements are those needed for reaching the other end point. The storage cost of this code is, as in the previous case, of the order of the number of pixels which make up the digital segment. Its associated drawing algorithm is also trivial and has the same time complexity as Bresenham's algorithm. However, in the worst case, the algorithm that solves the BELONG problem needs to go through all the code.

In comparison, the minimal code is the best of the above mentioned for solving the DRAWING and BELONG problems because it is the one that needs less space and its associated algorithms are optimal.

For simplicity, we will restrict the problem to drawing and encoding segments with (0,0) and (m,n) as end points, where $m > n > 0$. By s(p,q) we will denote the segment whose end points are p and q, and by $\lfloor x \rfloor$ the greatest integer which is less than the real number x.

§2. Bresenham's algorithm. On a computer screen, any picture can be obtained by illuminating elemental surface areas called pixels. The screen model on which we base this is made up of small squares within an ortogonal grid, which admit two states: black and white. As is shown in figure 1, the display of any linear object (pixels in black) looks locally like a staircase. Only the high resolution of video devices makes the representation appear, to the human eye, to have the same property of linearity as the represented object.

The mathematical model which we use to represent the screen model is an ortogonal grid ($\mathbb{Z}^2$) on which each node (point of integer coordinates) stands for one pixel (figure 2).

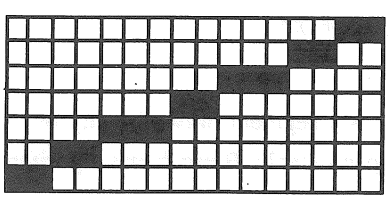

Figure 1. Screen Model

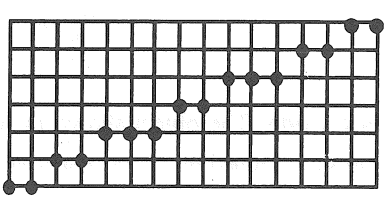

Figure 2. Mathematical Model

With the mathematical model that we have just shown, the problem of drawing a segment on a computer screen was solved by Bresenham calculating the closest integer coordinate points to it.

For each abscissa from 0 to m, Bresenham's algorithm determines, from the pixel P_{i-1} calculated in the previous step, the closest pixel to the segment. As is shown in figure 3, P_{i-1} determines the candidates T_i and S_i, and selects one or another depending on its distance from the actual segment point with this abscissa (being t_i and s_i, respectively). If $t_i \leq s_i$ then T_i is chosen as best approximation (i.e., $P_i = T_i$); otherwise $P_i = S_i$. The effective calculation of these distances requires real arithmetic, but as we only need to determine which is the shorter, we would only need to use a parameter of decision, d_i, which always takes integer values and is proportional to the difference of both distances, $t_i - s_i$. Observe that if, for a given abscissa, both candidates are equidistant from the actual segment point with this abscissa, the algorithm then chooses, by convention, the one with greater ordinate.

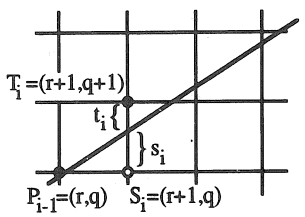

Figure 3. The Bresenham's algorithm chooses, from two candidates, the closer pixel to the segment

The algorithm begins by illuminating the pixel (0,0) and sets the parameter of decision: $d_1 = 2n-m$; then iteratively decides on the following pixel to be drawn and updates the parameter of decision. To be more precise: 1) if $d_i \geq 0$, the pixel T_i is chosen and $d_{i+1} = d_i + 2(n-m)$; 2) if $d_i < 0$ then S_i is chosen and $d_{i+1} = d_i + 2n$. Observe that the asymptotic time complexity of Bresenham's algorithm is O(m).

It can be easily proved that the points which Bresenham's algorithm draws verify the result that is presented in the following proposition. Its proof is straightforward.

Proposition 1. If Bresenham's algorithm calculates the point (p,q) where the input segment is s((0,0),(m,n)) the following inequalities can be verified:

$$\frac{(2q-1)m}{2n} \leq p < \frac{(2q+1)m}{2n}$$

§3. Stair Code. The object shown in figure 2 is a segment drawn by any one of the algorithms mentioned in the introduction and in particular by Bresenham's algorithm.

As can be seen, it looks like a staircase whose steps are generally different in size. With this type of structure, the last pixel of each step determines the rest. With this idea in mind, the list, made up of both segment end points and the last points of each step, can be interpreted as a code. Notice that from the previous proposition we can deduce that the last pixel on any step is (h_i,i) where $h_i = \lfloor (2i+1)m/2n \rfloor$, from $i = 0$ to $n-1$. Nevertheless, observe that the ordinate of the end of a step is one greater than the ordinate of the end of the previous one, so we need only calculate the abscissae of these points to obtain a code. Therefore, the *stair code* of the segment $s((p_1,q_1),(p_2,q_2))$ can be defined as the list $((p_1,q_1),(p_2,q_2), h_0, h_1,\ldots, h_{n-1})$, where $0 < n = q_2-q_1 < m = p_2-p_1$ and $h_i = \lfloor (2i+1)m/2n \rfloor$. It is easy to see that this code uses only O(n) space; that needed to store the n+4 integers which make it up.

§4. A new drawing algorithm. Bresenham's algorithm is an incremental algorithm which calculates a pixel from the information given by the one before. The new algorithm that we present also draws the pixels in an incremental way, but only needs to perform arithmetical calculations to find the ends of the steps. To be exact, this algorithm calculates the stair code of the segment which is drawn although it does not store it.

In order to reach this objective we consider the following equalities: $dn + r = m$ with $0 \leq r < n$, $h_i 2n + r_i = (2i+1)m$ with $0 \leq r_i < 2n$, where d and r (respectively, h_i and r_i) are the quotient and the rest of the integer division of m by n (respectively, of $(2i+1)m$ by $2n$). From these equalities we can deduce that if $r_i < 2n-2r$ then $h_{i+1} = h_i + d$ and $r_{i+1} = 2r + r_i$, otherwise $h_{i+1} = h_i + d + 1$ and $r_{i+1} = 2r + r_i - 2n$.

The algorithm we propose, from $i = 1$ to $n-1$, calculates the value h_i and updates the variable of decision r_i from the value h_{i-1} and the variable of decision r_{i-1}. To calculate h_0 and r_0 we can simply observe that $h_0 2n + r_0 = dn + r$. Therefore, if d is even $h_0 = d/2$ and $r_0 = r$, and if d is odd, $h_0 = (d-1)/2$ and $r_0 = r + n$.

In the figure 4 we show an implementation, in PASCAL, of the new algorithm.

It can be seen that this algorithm does not exactly illuminate the pixels that Bresenham's algorithm does. They coincide in those points (p,q) which verify the inequality $| q - np/m | < 1/2$. However, if the points (p,q) and (p,q-1) are equidistant from the segment, Bresenham's algorithm chooses the former, while NEW takes the latter. As we have already mentioned, this difference is not important as the choice of one pixel or another is by convention. In any case, a slight modification of NEW would draw the same pixels as Bresenham's algorithm would.

To compare the efficiency of both algorithms we can calculate their costs depending on the number of additions and subtractions they perform. This is a realistic measure since both draw the same number of pixels and use integer arithmetic. It is not difficult to check that the cost of Bresenham's algorithm is $2m+n$ while the NEW cost is $m+3n-2$. Therefore, both algorithms have the same worst-case asymptotic complexity. However,

from these calculations we can deduce that Bresenham's algorithm is faster for the segments whose slope is greater than 1/2, while the one we have just presented is more appropriate for those whose slopes are less than 1/2.

```
procedure NEW(n,m:integer);
var incr1_hi,incr1_ri,incr2_hi,incr2_ri,rcom,hi,ri,x,y:integer;
begin
    incr1_hi:=m div n; incr1_ri:=m mod n;      {incr1_hi*n + incr1_ri = m}
    hi:=incr1_hi div 2;                        {calculation of h0}
    if hi*2=incr1_hi                           {calculation of r0}
      then ri:=incr1_ri
      else ri:=incr1_ri+n;
    incr1_ri:=2*incr1_ri;          {increase for the rest in the case ri < 2n - 2r}
    incr2_hi:=incr1_hi+1;          {increase for hi in the case ri ≥ 2n - 2r}
    incr2_ri:=incr1_ri-2*n;        {increase for the rest in the case ri ≥ 2n - 2r}
    rcom:=-incr2_ri;
    x:=0; y:=0;
(1) while x≤hi do                  {drawing the first step}
      begin write_pixel(x,y); x:=x+1 end;
    y:=y+1;
(2) while y<n do
      begin
        if ri<rcom
          then begin hi:=hi+incr1_hi; ri:=ri+incr1_ri end      {ri < 2n - 2r}
          else begin hi:=hi+incr2_hi; ri:=ri+incr2_ri end;     {ri ≥ 2n - 2r}
(3)     while x≤hi do              {drawing the intermediate steps}
          begin write_pixel(x,y); x:=x+1 end;
        y:=y+1
      end;
(4) while x≤m do                   {drawing the last step}
      begin write_pixel(x,y); x:=x+1 end
end;  {end of NEW}
```

Figure 4. An algorithm for drawing a segment

In practice, the difference between the performance of both these algorithms is hardly noticeable. As proof of this, we have implemented both algorithms on an HP-APOLLO 9000/425t computer, calculating the pixels corresponding to segments s((0,0),(m,n)), for some values of m between 2 and 4000, and $1 \leq n \leq m$. After running each algorithm a million times the difference in speed observed was a few milliseconds in favour of one or another.

So, in practice it is not important whether we choose one algorithm or another to draw a segment. However, in certain situations, the NEW algorithm seems more convenient because it is based on a code which contains more explicit information on the segment points. On the other hand, the NEW algorithm can be modified so that, without any additional expense, pixels of the segments whose slope is more than 1/2 can be drawn by means of calculation of the pixels of the symmetric segment with regard to the main diagonal. In this way we obtain an algorithm whose running cost is less than Bresenham's whatever the slope of the segment may be.

Notice that the NEW algorithm may be modified to obtain the stair code of a segment from its end points. The only thing we need to do is replace the pixels of the segment in the output with this code. In this situation this calculation can be carried out in 3n steps, while using Bresenham's algorithm it would take 2m+n steps.

The algorithm shown draws a segment setting out from its end points. Nevertheless, should the stair code of this segment be already calculated, we can obtain an algorithm for drawing from the previous one, by eliminating the calculation of the code from it. In this case the cost of this algorithm is only m+n.

§5. The BELONG problem. We shall give the algorithm that solves the BELONG problem as an example of the use of the stair code. To decide whether a point belongs to a segment we need only check if it is on the step possibly determined by its ordinate. In order to do this it suffices to check if the ordinate is in the permitted range and then to see if the abscissa is on the corresponding step; that is, if $h_{y-1} < x \leq h_y$ is fulfilled. With this idea in mind, figure 5 shows a function written in pseudocode which uses the stair code of the segment and solves the problem.

```
Function BELONG(x,y:integer; cod:code):boolean;
begin
 if (y<y1) or (y>y2)
        then return(false)
        else if y=y1
                    then return((x1≤x) and (x≤x1+h0))
                    else if y=y2
                                then return((x1+hn-1<x) and (x≤x2))
                                else return((hy-1<x-x1) and (x-x1<hy))
end;
```

Figure 5. An algorithm that decides whether a point belongs to a segment from its stair code

The time complexity of this algorithm is constant and it uses only sum and comparison operations because the values of h_{y-1} and h_y appear explicitly in the code. Next we state the solution to this problem by the following proposition whose proof is easily deduced from the algorithm and the previous comments.

Proposition 2. The BELONG decision problem can be answered in constant time and without preprocessing, using integer products and divisions. This problem can be solved in constant time, given O(n) space and O(n) preprocessing time, using only comparisons.

§6. Digitization. Rosenfeld-Kim [11] presented an accurate notion of digital segment if the 8-adjacency is considered on the grid $\mathbb{Z}^2$. To be more precise, they define a digital segment as a digital 8-arc which is the digitization, by means of what they call 'grid digitization', of an Euclidean segment. After this, they give an intrinsic characterization of

the digital 8-arcs which are digital segments. This notion, as we shall soon see, can be given without using the previous concept of a digital arc.

An *Euclidean digitization* is a function c: $\mathbb{R}^2 \longrightarrow \mathcal{P}(\mathbb{Z}^2)$ such that c(p,q) = (p,q) for every $(p,q) \in \mathbb{Z}^2$ and $\{\tilde{c}(p,q); (p,q) \in \mathbb{Z}^2\}$ is a locally finite family of connected subspaces with compact closure of the Euclidean plane $\mathbb{R}^2$, where $\tilde{c}(p,q) = \{(x,y) \in \mathbb{R}^2; c(x,y) = (p,q)\}$ and $\mathcal{P}(\mathbb{Z}^2)$ stands for the set of all subsets of $\mathbb{Z}^2$ (see [5] for a more general definition). A *digitization of a subspace* of $\mathbb{R}^2$ is its image through this function c. A digitization of an Euclidean straight line (segment) is called a *digitized straight line* (*segment*, respectively).

If one chooses the $\tilde{c}(p,q) = \{(x,y); |p - x| + |q - x| \leq 1/2\}$ digitization, it is easy to prove that the digitization of an Euclidean straight line $r_{(m,n)}$ which passes through the origin (0,0) and the point (m,n), where $0 < n < m$, is the digitized straight line

$$L_{(m,n)} = \mathbb{Z}^2 - \{(p,q) \in \mathbb{Z}^2; \tilde{c}(p,q) \subset \mathbb{R}^2 - r_{(m,n)}\}$$

When the Euclidean straight line $r_{(m,n)}$ goes through a point like (p,q+ 1/2), where $(p,q) \in \mathbb{Z}^2$, the digitized straight line $L_{(m,n)}$ contains the points (p,q) and (p,q+1), so that, in this sense, the corresponding digital object looks thick in these points. A *digital straight line* (or a *digital segment*) is the object obtained on straightening these zones. To be more precise, in the case before the associated digital straight line is

$$R_{(m,n)} = L_{(m,n)} - \{(p,q) \in L_{(m,n)}; (p,q-1) \in L_{(m,n)}\}$$

In other words, the straightening of the digitization of an Euclidean straight line (segment) is called *digital straight line* (*digital segment*, respectively). With this definition it can be shown that the NEW algorithm presented in this paper is a drawing algorithm for digital segments and that the stair code of a segment encodes the associated digital segment.

Acknowledgements. We would like to acknowledge the partial assistance of the Diputación General de Aragón (D.G.A) (for the work of Angel Francés) and the Dirección General de Investigación Científica y Técnica (DGICYT) and the Junta de Andalucía (for the work of Alberto Márquez).

References

[1] J.E. Bresenham, Algorithm for Computer Control of a Digital Plotter, *IBM Systems Journal* 4 (1), 25-30, 1965.

[2] R. Brons, Linguistic Methods for the Description of a Straight Line Upon a Grid, *Computer Graphics and Image Processing* 3 (1984), pp. 48-62.

[3] C.M. Castle, M.L.V. Pitteway, An Application of Euclid's Algorithm to Drawing Straight Lines, in *Proc. NATO ASI on Fundamental Algorithms for Computer Graphics*, Springer-Verlag, Heidelberg, 1985.

[4] C.M. Castle, M.L.V. Pitteway, A Technique for Encoding 'Best-Fit' Straight Lines, *Computer Journal*, vol. 30, no. 2, pp. 168-175, 1987.

[5] E. Domínguez, A. Francés, A. Márquez, Digital Spaces: A Mathematical Model for Digital Image Processing, *to appear*..

[6] L. Dorst, R. Duin, A Framework for Calculations on Digitised Straight Lines, *IEEE Trans. Pattern Analysis and Machine Intelligence* PAMI-6 (5), 1984.

[7] R.A. Earnshaw, Line Tracking with Incremental Plotters, *Computer Journal* 23 (1), 1980.

[8] J.D. Foley, A. Van Dam, *Fundamentals of Interactive Computer Graphics*, The Systems Programming Series, Addison-Wesley, 1984.

[9] H. Freeman, Boundary Encoding and Processing, in *Picture Processing and Psichopictorics*, 241-266, Academic Press, New York, 1970.

[10] M.L.V. Pitteway, A. Green, Bresenham's Algorithm with Run-Line Coding Shortcut, *Computer Journal* 25 (1), 114-115, 1982.

[11] A. Rosenfeld, Ch.E. Kim, How a digital computer can tell whether a line is straight, *Amer. Math. Monthly* 89, 230-238, 1982

[12] A. Rosenfeld, R.A. Melter, Digital Geometry, *The Mathematical Intelligencer* 11 (3) 69-72, 1989