Applications of Digital Straight Segments to Economical Image Encoding

Vladimir Kovalevsky

Technische Fachhochschule Berlin
Luxemburger Str. 10, 13353 Berlin, Germany
email: kovalev@tfh-berlin.de

Abstract. A new classification of digital curves into boundary curves and visual curves of different thickness is suggested. A fast algorithm for recognizing digital straight line segments in boundary curves is presented. The algorithm is applied to encode the boundaries of homogeneous regions in digitized images. The code is economical and enables an exact reconstruction of the original image.

1 Introduction

The aim of this presentation is to demonstrate that methods of recognizing, encoding and decoding segments of digital straight lines may be successfully applied to economical encoding of digitized images and to exactly reconstruct the original image from the code. Methods of recognizing digital straight segments (DSS) are known during a long time. One of the first methods is due to Freeman [Fre74]. He suggested to analyze the regularity in the pattern of the directions in the chain code [Fre61] of a digital curve. Anderson and Kim [AndKim85] have presented a deep analysis of the properties of the DSS's and suggested a different algorithm based on calculating the convex hull of the points of the digital curve to be analyzed. The author has suggested a simple and fast version of this algorithm [Kov90] and demonstrated that the recognition may be performed as a successive, point for point, actualization of the coefficients of the linear inequality, which all points of a DSS must satisfy. Today this inequality is generally known. It was successfully used in many recent works, e.g. [RevDeb94], [Deb95], [Franc96].
The author has devoted his investigation to a special class of the DSS's: the boundary lines. The reader will find in this presentation a *new classification of digital curves* into boundary curves and visual curves. Boundary curves and lines are a useful means for fast drawing of regions defined by their boundaries. The algorithm for filling the interior of a closed boundary curve represented as a sequence of boundary cracks [Kov84], [Kov94a] is very simple and fast. It is an important tool for the reconstruction of images encoded by means of the boundaries of homogeneous regions. However the known methods of encoding boundaries without loss of information need relatively much memory space. We suggest here a *new method of encoding* digital straight segments without loss of information which is more economical then the known methods.

Sections 2 and 3 contain a short summary of the topological and geometrical background of the further sections. Section 4 is devoted to the new classification of digital curves. It also contains a short description of a new fast algorithm for drawing visual curves of arbitrary width with antialiasing when given an equality of the curve in the form $F(x,y)=0$. $F(x,y)$ may be given as any floating point function in a programming language.
Section 5 presents a new derivation of the well-known properties of the DSS's. This section is the theoretical foundation of the recognition algorithm. The algorithm described in Section 6 is not new: this is the algorithm of [Kov90]. What is new is the method of encoding the parameters of the DSS's which is presented in Section 7. Section 8 contains some experimental results.

2 Topological Background.

We consider here the digital plane as a two-dimensional cell complex [Kov89] rather then a set of pixels. Thus our digital plane contains *cracks* and *points* besides the pixels. Cracks are the sides of the pixels, the latter being considered as square areas. From the point of view of cell complexes pixels are two-dimensional and cracks are one-dimensional cells. The points are end points of the cracks and therefore the corners of the pixels. Points are zero-dimensional cells.

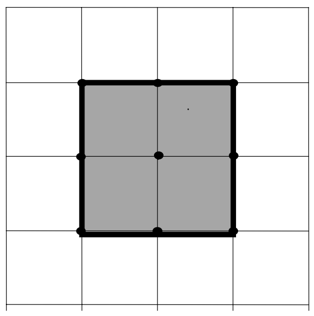
Fig. 1. Example of a two-dimensional complex with the crack boundary of the shaded subset

As demonstrated in [Kov89], considering the plane as a cell complex brings many advantages: there are no more connectivity paradoxes, a boundary becomes a thin curve with a zero area, the boundary of a region and that of its complement are the same etc. The definition and the processing of digital curves and especially that of digital straight lines becomes simpler and clearer. The most important advantage from the point of view of economical encoding and exact reconstruction of images is the possibility to fill the interior of crack boundaries by an extremely simple and fast algorithm [Kov84], [Kov94a] which cannot be applied when representing boundaries as sets of pixels.

The rest of this section contains a short summary of the topological notions important for this presentation. Please refer to [Kov89] for more details and topological foundations. The reader acquainted with cell complexes may skip the rest of this section.

An *D*-dimensional cell complex is a structure consisting of abstract elements called cells. Each cell is assigned an integer value from 0 to *D* called its dimension. There is a bounding relation imposed onto the cells: a cell of a lower dimension may bound some cells of a higher dimension. An example of a two-dimensional complex is shown in Fig. 1.

The pixels are represented in Fig. 1 as the interiors of the squares, the cracks as the sides of the squares and the points, i.e. the 0-cells, are the end points of the cracks and simultaneously the corners of the pixels.

Now let us introduce some notions which we shall need in the sequel. A *boundary crack* of a subset *S* of a complex is a crack separating a pixel belonging to *S* from another pixel not belonging to *S*. The boundary cracks of the shaded subset in Fig. 1 are drawn as bold lines. The *boundary* (also known as *crack boundary,* [RosKaak82]) of a subset *S* is the set of all boundary cracks of *S* and all end points of these cracks. A boundary contains no pixels and is therefore a „thin“ set whose area is zero. A connected subset of a boundary is called a *boundary curve*. For the notion of connectedness please refer to [Kov89].

3 The Coordinates

We consider the digital plane as a *Cartesian two-dimensional complex* [Kov94b], i.e. as a Cartesian product of two one-dimensional complexes which are the coordinate axes of the plane. The *X*-coordinate is the row number, the *Y*-coordinate is the line number. We use here the coordinate system of computer graphics, i.e. the positive *X*-axis runs from left to right, the positive *Y*-axis runs from top to bottom.

We often need Euclidean coordinates to discuss problems of digitizing Euclidean objects. To remain in the frame of cell complexes we suggest to consider Euclidean coordinates as rational numbers with a relatively great denominator. This corresponds to any computer model since the floating point variables in computers are rational numbers with great denominators.

It is possible to introduce additionally to the complex of the digital plane another cell complex, the *fine complex*, whose cells are obtained by subdividing the original cells into many smaller cells. The coordinates in the fine complex are rational numbers. Both coordinate systems of the original and the fine complex may be adjusted to each other in such a way that the 0-cells (i.e. the points) of the original complex have integer coordinates. Then we have the possibility to consider other points lying inside the pixels and cracks. These point have fractional coordinates. Particularly important for this presentation are the points in the middle of the pixels. They have „half-integer“ coordinates, i.e. fractional coordinates with an odd numerator and the denominator equal to 2: e.g. 0.5, 1.5, 2.5 etc.

4 Classification of Digital Curves

Curves are considered in the classical geometry as objects with zero width and zero area. In digital geometry we consider the number of pixels (being elementary areas) in a subset *S* as the measure of the area of *S*. A boundary curve is a sequence of cracks. It contains no pixels and thus it exactly corresponds to the classical notion of a curve. Although a boundary curve contains no pixels, it is possible to make it visible on the screen of a computer. If we are ready to make a magnified representation of a complex, then we draw a 2-cell as a small square consisting of $N \times N$ screen pixels. The cracks can be represented as thin vertical or horizontal bars one screen pixel wide and *N* screen pixels long. A digital curve is then a sequence of such vertical an horizontal bars. If one does not want to magnify the complex then each crack of a curve may be represented by a single screen pixel whose coordinates correspond to the upper or to the left end of the crack. Such a representation has two drawbacks:

1. The image of the curve is slightly displaced relative to the positions of pixels which may be simultaneously represented on the screen;
2. The image of the curve is too thick at some places where an upper end point of a vertical crack is simultaneously the left end point of a horizontal crack. The image of the curve is not a „thin" sequence of screen pixels. It is not a skeleton.

To overcome these drawbacks we suggest here to consider in computer graphics two classes of curves: the *boundary curves* as sequences of cracks and the *visual curves* as sequences of pixels. Boundary curves are important as a means to exactly specify the boundaries of subsets. They are a most suitable tool to draw areas filled by some color: the filling algorithm of [Kov94a] being applicable in this case is much simpler and faster then any algorithm suitable to fill the interior of a closed sequence of pixels. The exact visual representation of boundary curves is less important.

Visual curves must be used in the cases when a Euclidean curve, e.g. specified by its equation, must be displayed in such a way that it gives a visual impression of the mathematically specified curve as exact as possible, including the impression of homogeneous thickness. For the last purpose the so called antialiasing methods have been developed in computer graphics. The author has developed a new universal algorithm for drawing visual curves of arbitrary width with antialiasing. The algorithm may draw any curve specified by an equation of the form $F(x,y)=0$. The equation may be given as a function in a programming language while having floating point arguments and returning a floating point value. The algorithm implicitly calculates two curves displaced from $F(x,y)=0$ by the half of the desired width along and against the gradient of $F(x,y)$ and estimates the area of the part of each pixel (in the vicinity of $F(x,y)=0$) which part lies between the two displaced curves. The color of the pixel to be drawn depends upon the estimated area. The estimation is realized without use of a finer raster (as e.g. in [Whit83]) and is therefore very fast and precise.

In the following sections we consider only boundary curves and a method of subdividing a digital curve in digital straight line segments of maximal possible length. The segments are also boundary segments.

5 Properties of Digital Straight Segments.

We are speaking here about digital straight segments (DSS) rather then about digital lines because in practice we always consider finite subsets of the digital plane. Such a subset can only contain a subset of a line, i.e. a line segment.

It is natural to define a digital straight segment as a result of the digitization of a Euclidean straight segment. Various digitization schemes of digitizing curves were suggested (e.g. [AndKim85]). Most of them are based on testing the crossings of the given curve with the grid lines. It is rather difficult to mathematically justify these schemes or specify some reasons for preferring one of them. Therefore we are suggesting here one scheme more which follows from the commonly used practical way of digitizing regions in raster images.

The natural way of digitizing a region in the Euclidean plane consists in subdividing the plane in regularly spaced elementary areas corresponding to the pixels and in measuring for each pixel its portion covered by the given Euclidean region. The measured value must be compared with a threshold. A pixel must be set to 1 if the measured portion is greater than the threshold and set to 0 otherwise. We call this method *digitizing by thresholding*.

This scheme may be applied to *digitizing Jordan curves* in the following way:

1) Given a Euclidean Jordan curve *JC* construct the Euclidean region *R* which is the interior of *JC*;
2) Digitize *R* by thresholding and place the results into the 2-dimensional cell complex thus defining a digital region *DR*;
3) Find the boundary of *DR*. This is the digital image of *JC*.

The above scheme is slightly bothersome because of the necessity to measure area portions for each pixel. There is however a simple particular case of the scheme. Consider first the digitization of a straight line. The corresponding region *R* is a half-plane *H*. Let us digitize *H* with a threshold equal to 0.5. Then digitization may be performed *without calculating area portions* since a half-plane covers more than 0.5 of a rectangular elementary area iff the middle point of the area is inside *H* which may be found by substituting the coordinates of the middle point into the inequality of *H*.

The same scheme may be used for digitizing curves with restricted curvature. A simple calculation shows that the error in estimating the area portion is less than 1% if the curvature radius is greater than 6 times the side of a pixel.

Thus we arrive at the following digitization scheme for Euclidean curves (e.g. given by equations):

1) Subdivide the plane in regularly spaced squares and define the middle point of each rectangle;
2) Establish a mapping of the set of the squares onto the 2-dimensional complex while mapping the squares onto the 2-cells (pixels) and their sides onto the 1-cells (cracks).
3) Find two adjacent pixels whose middle points lie on different sides of the given Euclidean curve. If the curve is given by an equation like $F(x,y)=0$, then the values of $F(x,y)$ at the said two middle points must have different signs. The crack between the two pixels is then a boundary crack. It belongs to the digital curve.

The well-known properties of DSS's (see e.g. [AndKim85]) may be easily derived from our digitization scheme as follows. Consider a half-plane whose inequality is $H(x,y)=P\cdot x+Q\cdot y+R>0$, where P and Q are integers. The middle points of all pixels are subdivided into two subsets: those with positive and those with non-positive values of $H(x,y)$. Let us call them „positive" and „negative" middle points. For every 0-cell C belonging to the boundary of these subsets (Fig. 2) there exist four adjacent middle points whose distance to C is equal to $\sqrt{2}/2$ of the length of a pixel side. We shall consider since now the length of a pixel side as the unit of measurement. The four middle points compose two diagonal pairs (MP_1, MP_4) and (MP_2, MP_3) each having the 0-cell C in the middle. These pairs are shown in Fig.2. Consider that of the two pairs for which the absolute difference of the values of $H(x,y)$ is greater (or equal) to that for the other pair (the absolute differences are equal only for half-planes with horizontal or vertical boundaries). Let us call the corresponding diagonal direction the *main diagonal* for H. It is (MP_2, MP_3) in Fig. 2.

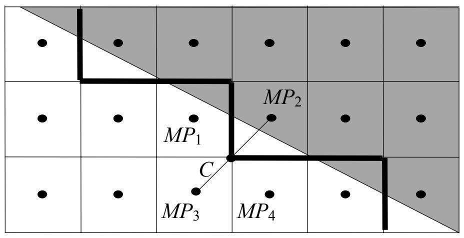

Fig. 2. The positive and the negative middle points of a half-plane and the main diagonal (MP_2, MP_3) of the half-plane

Since the vector connecting the middle points of a pair has both its X- and Y-components equal to ±1, it is easy to see that the maximum absolute difference is equal to $|P|+|Q|$. The values of the points belonging to the same pair must have different signs or, more exactly, one value must be positive and the other negative or zero. Thus the maximum positive value of $H(x,y)$ for all „positive" middle points adjacent to a boundary point like C cannot be greater than $|P|+|Q|$ and the adjacent „negative" middle points have all values strongly greater than $-(|P|+|Q|)$. Since the 0-cells of the boundary lie exactly in the middle between the middle points of each mean diagonal pair, the values of $H(x,y)$ for the 0-cells must satisfy the condition:

$$-(|P|+|Q|)/2 < H(x,y) \leq +(|P|+|Q|)/2. \qquad (1)$$

This is an important property of the points of a DSS which may be used for fast recognition of the DSS's.

The next important property of a DSS is its periodicity. Consider a 0-cell C with coordinates (X,Y) belonging to the DSS which belongs to the boundary of the half-plane $H(x,y)=P\cdot x+Q\cdot y+R\geq0$. Another 0-cell with coordinates $(X+a,Y+b)$ will have the same value of $H(x,y)$ if the condition $P\cdot a+Q\cdot b=0$ holds. Let $P=P_s\cdot GCD$ and $Q=Q_s\cdot GCD$ where GCD is the greatest common divisor of P and Q. Then the smallest values of a and b must satisfy the condition $P_s\cdot a+Q_s\cdot b=0$ or $a=Q_s$ and $b=-P_s$.

Therefore the vector (a, b) defines the period of the DSS. It is not correct to affirm that every DSS is periodic: it may consist of exactly one period. In such a case the assertion about the periodicity has no sense. A correct assertion is: any DSS has a subsequence, called its period, such that any multiple repetition of the period always produces a DSS. These properties enable the recognition of DSS's: given a digital curve *CV* as a sequence of 0-cells and cracks it is possible to find the longest segment of *CV* which is a DSS.

Let us remind some definitions introduced in [Kov90], namely that of the base of a DSS and of its starting and end points. Consider the „negative" and „positive" subsets of pixels corresponding to a given half-plane inequality $H(x,y)\geq 0$ as mentioned above. A connected subset of the boundary of these subsets is a DSS. It consists of cracks and points and contains no pixels. The Euclidean straight line containing the longest edge of the convex hull of the points is called the *base of the DSS* (the same as the „nearest support" of [AndKim85]). There is another base touching the hull on the opposite side and being parallel to the first one.

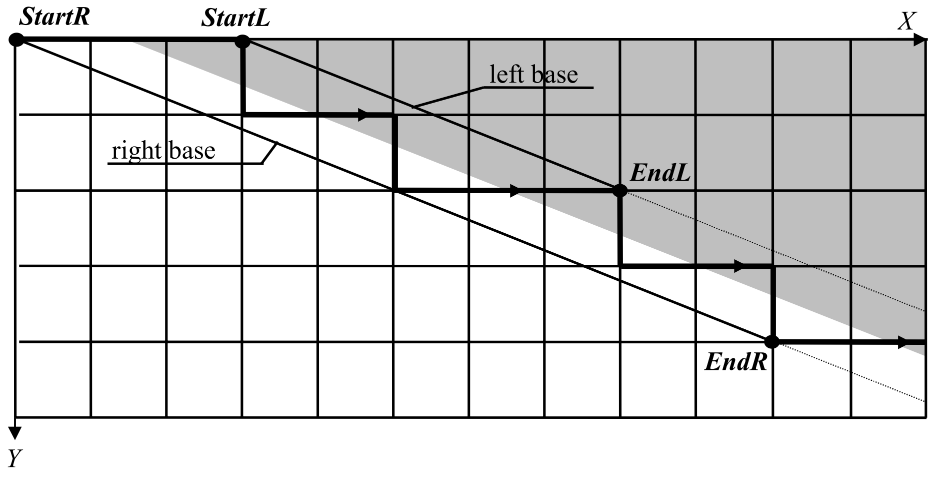

Fig.3. The bases of a DSS and their end points

Let us give the DSS such an orientation (arrows in Fig. 3) that the „positive" pixels are to the left hand side of the DSS. Correspondingly we call one of the bases the left and the other the right base. In Fig. 3 the positive half-plane is shown as the shaded region. The cracks of the corresponding DSS are drawn as bold lines. ***StartL*** and ***EndL*** are the vectors of the end points of the left base; ***StartR*** and ***EndR*** are those of the right base.

Let us choose the constant R in the inequality $H(x,y)=P\cdot x+Q\cdot y+R\geq 0$ such that $H(x,y)$ be equal to zero for points lying on the right base. The difference between the minimum and the maximum values of $H(x,y)$ at all points of the DSS remains unchanged. Therefore according to (1) the values of $H(x,y)$ must be now in the range:

$$0 \le H(x,y) < |P|+|Q|. \qquad (2)$$

If P and Q are relatively prime then the difference between the maximum value of $H(x,y)$ and $|P|+|Q|$ has the smallest possible integer value 1. Therefore for all points of a DSS the following inequality holds:

$$0 \le H(x,y) \le |P|+|Q|-1. \qquad (3)$$

The values of P and Q may be considered as the components of the normal to the Euclidean line specified by $H(x,y)=0$. It is more convenient for our purpose to consider a vector along the base of the DSS whose components are relatively prime. The vector is obviously perpendicular to the normal and has the X-component $N=-Q_s$ and the Y-component $M=P_s$. Thus we can rewrite the above inequalities as:

$$H(x,y)=M\cdot(x-X_r)-N\cdot(y-Y_r); \qquad (4)$$

$$0 \le H(x,y) \le |M|+|N|-1; \qquad (5)$$

where X_r and Y_r are coordinates of a point on the right base.

Inequalities (5) give us the possibility the decide whether a point belongs to a DSS with given parameters M, N, X_r and Y_r. A more complicated problem is the inverse one: given a sequence of points find whether there exist such values of the parameters M, N, X_r and Y_r that the sequence satisfies (5). The author has suggested a solution of this problem in [Kov90]. A short explanation of the solution follows.

The main idea of the solution consists in starting with the parameters for a trivial DSS consisting of a single crack and in actualizing the parameters after each step along the given sequence of points and cracks. To explain the actualization one more property of the DSS's must be demonstrated. We have showed that the difference of the values of $H(x,y)$ for any two points of a DSS must be less than $|M|+|N|$ (compare (5)). The value of $H(x,y)$ may be interpreted according to (4) as being proportional to the projection of the vector $\boldsymbol{P}-\boldsymbol{P_r}$ onto the normal, where $\boldsymbol{P}=(x,y)$ is the actual point of the sequence to be tested and $\boldsymbol{P_r}=(X_r, Y_r)$ is a point on the right base. If $H(x,y))$ at $\boldsymbol{P}$ becomes exactly equal to $|M|+|N|$ (or respectively to -1) while it was in the range of (5) for all other already tested points, then there is the possibility to slightly rotate the normal in such a way, that $H(x,y)$ at $\boldsymbol{P}$ becomes slightly less (respectively greater) thus satisfying the conditions (5) and the values of $H(x,y)$ for all other points remain in the permitted range. The author has proved that this rotation must be performed by moving the end point of one of the bases to the actual point $\boldsymbol{P}$. This is the end point closest to $\boldsymbol{P}$ and belonging to the base closest to $\boldsymbol{P}$. This idea is realized in the following algorithm.

6 Recognition of Digital Straight Segments.

The recognition algorithm uses the inequality (5) of a DSS. We consider all points as vectors. We denote the components of a vector ***V*** as *V.X* and *V.Y*. The pair of values (*N*, *M*) is also a vector denoted as ***TANG*** („tangent“): *TANG.X*=*N*, *TANG.Y*=*M*. The instruction „continue“ in the following description of the algorithm means „change nothing, get the next point of the sequence“. The instruction „break“ means „the actual point ***P*** does not belong to the DSS; store the point preceding ***P*** as the end point of the DSS; return a break message“. We shall denote by $HL(x,y)=0$ the equation of the left and by $HR(x,y)=0$ that of the right base. The notations ***StartL***, ***EndL*** etc. were explained in the previous section (Fig. 3).

Take the first two points $\boldsymbol{P}_1$, $\boldsymbol{P}_2$ of the given curve and set ***StartL***=***StartR***=$\boldsymbol{P}_1$; ***EndL***=***EndR***=$\boldsymbol{P}_2$. Set the starting values of ***TANG***=***EndL***–***StartL*** (*M* and *N* take the values of either to 0 or ±1). Set the half-plane values *HL*=*HR*=0.

1. Store the first two different directions of cracks appearing since the start.
2. For every next point ***P*** of the curve:
   - 2.1 Test the direction of the last step, whether it is equal to one of the two directions having appeared until now:
     If it is not, then break (the sequence is no more a DSS);
     else increase *HL* by $-(M\cdot STEP.X-N\cdot STEP.Y)$ and *HR* by $(M\cdot STEP.X-N\cdot STEP.Y)$, where ***STEP*** is the unity vector having the direction of the last step. (Note that one of the components of ***STEP*** is zero and the other is equal to ±1).
   - 2.2 Test the value of *HR*:
     HR is positive: continue;
     HR is zero: set ***EndR***=***P***, continue;
     HR is equal to −1: calculate the new values of ***TANG***, ***EndR***, ***StartL***, *HL* (the only two multiplications); set *HR*=0; continue;
     HR is less than −1: break.
   - 2.3 Test in a similar way the value of *HL*.

End of the algorithm.

After a "break" the point ***PP*** directly preceding the actual point ***P*** must be stored as the end point of the DSS. The next DSS begins at ***PP*** while ***PP*** is the first and ***P*** the second point of the new DSS and so on.

7 Encoding of curves

The Crack Code

A well-known way of encoding digital curves is that of the Freeman code [Fre61]. Boundary curves in a two-dimensional complex are sequences of cracks and points. Oriented cracks may have only four directions. Therefore they may be encoded by a Freeman code with four directions which is also well-known as the crack code (see e.g. [RosKaak82]).

This way of encoding curves is a rather economical one especially if only two bits per crack are used. Its main drawback is the difficulty of performing geometrical transformations: only a translation is easy to realize. Rotation and scaling are hardly realizable without converting the crack code into some other code more suitable for geometrical transformations.

End Points of DSS's
Another way of encoding curves consists in decomposing the curve into as long as possible DSS's and recording the coordinates of their end points. This code makes geometrical transformations easily realizable: it suffices to multiply the vectors corresponding to all end points with the matrix of the desired transformation, eventually in homogeneous coordinates. However this code does not allow an exact reconstruction of the original digital curve since there exist many different DSS's having the same end points. The distance between any two such DSS's is never greater than the pixel's diagonal. Therefore the difference between such two DSS's may be considered in many applications as negligible. In such cases this way of encoding curves is the simplest and the most economical one, especially if we record coordinate increments rather then the coordinates itself. If however an exact reconstruction is necessary than the following ways of encoding are possible.

Floating Point Coordinates
A DSS is uniquely specified by its end points and any Euclidean straight line which is a preimage of the DSS. The Euclidean line parallel to the bases and lying exactly in the middle between them is such a line. We shall call the line *the axis of the DSS*. The parameters of the axis and the location of the end points may be combined if we calculate the crossing points of the axis with the main diagonals (s. Section 5) containing the end points. The crossing points uniquely define both the axis and the integer end points which are the integer points nearest to the crossing points. The drawbacks of this way of encoding are as follows:

a) the memory demand is relatively great, e.g. two „float" coordinates of each of two crossing points with two axes of subsequent DSS makes 4×2×2 bytes per end point;
b) a common end point of two subsequent DSS's of a digital polygon may be „split" after geometrical transformations into two different integer points thus making the polygon disconnected.

There are some rather complicated ways of overcoming one of the drawbacks but not both of them.

Additional Integer Parameters of a DSS
A DSS may be uniquely specified by its end points and by one of its bases. Specifying the base by the coordinates of its end points (which are mostly different from the end points of the DSS) is rather redundant. A more economical way of encoding consists in specifying two integers M and N for the direction of the base accompanied by the distance of the starting point of the DSS from the base. The distance itself is a small value and must be represented by a floating point variable. However there is a possibility to specify a positive integer L not greater than $|M|+|N|-1$ which together with the values of M and N specifies the distance uniquely.

The coordinates of the end points of the DSS may also be encoded economically: only the coordinates of the starting point of the curve must be specified explicitly; all other coordinates may be defined by means of the number *NC* of cracks in the current DSS. Thus we need four integers per DSS: *NC*, *L*, *M* and *N*. Since the majority of the DSS's are relatively short, these integers are mostly small ones: all four integers may be packed into a word of 2 bytes. For longer DSS's a longer word of 3 or 4 bytes may be needed, but the frequency of such long words is low. Therefore the average number of bytes per one DSS lies between 2 and 3.

8 Experiments

The author has developed a program which traces the boundaries of regions with constant gray levels in an image of 1 byte per pixel, dissects the boundaries in as long as possible DSS's and encodes them by the code described in the previous section. Another program reconstructs the image from the code. The reconstructed image is always identical with the original one. Numerous experiments have shown that the average number of bytes per one DSS is of order of 2.3. This leads to an economical encoding of images: the compression rate is of the order of about 20 to 40 for images with a low density of fine details and of about 3 for images with many fine details or many gray levels. The length of the code is approximately inversely proportional to the number of gray levels in the original image. To obtain a high compression rate the image must be properly smoothed and quantized.

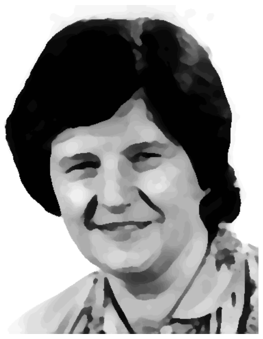

Fig. 4. An example of an encoded image

Fig. 4 shows an image of 390×480=187200 pixels with 32 gray levels. It was encoded with 60080 bytes. Thus the compression rate was about 3.1.
When encoding the same images by means of the crack code, the compression rate is about 1.5 for a low density of details and about 0.15 (no compression) for images with many fine details. Thus the encoding by means of the DSS's brings a much higher compression rate than that of crack codes. The suggested method is less efficient then the well-known ARJ method, but our method represents the image in a „geometrical" way: i.e. it is possible to extract from the code the boundaries of the regions and thus perform image analysis by means of the code.

References

[AndKim85] T. A. Andersen and C.E. Kim: Representation of digital line segments and their preimages. CVGIP 30, 279-288 (1985).

[Deb95] I. Debled-Renesson: Etude et reconnaissance de droites et plans discrets. Thése de doctorat soutenue à l'Université Louis Pasteur de Strasbourg, 1995.

[Franc96] J. Françon, J.-M. Schramm and M. Tajine: Recognizing arithmetic straight lines and planes. In: Miguet, S., Montanvert A., Ubéda S. (eds.): Discrete Geometry for Computer Imagery. Berlin Heidelberg New York Tokyo: Springer 1996 (Lecture Notes in Computer Science, vol. 1176, pp. 141-150).

[Fre61] H. Freeman: On the encoding of arbitrary geometric configurations. IRE Trans. Electron. Comput. EC-10, 260-268 (1961).

[Fre74] H. Freeman: Computer processing of line-drawing images. Comput. Surv. 6, 57-97 (1974).

[Kov84] V.A. Kovalevsky: Discrete topology and contour definition. Pattern Recognition Letters 2, 281-288 (1984)

[Kov89] V.A. Kovalevsky: Finite topology as applied to image processing, CVGIP 46, 141-161 (1989).

[Kov90] V.A. Kovalevsky: New definition and fast recognition of digital straight segments and arcs. In: Proceeding of the International Conference on Pattern Recognition (ICPR-10), Los Alamitos, IEEE Computer Society Press 1990 (vol. II, pp. 31-34).

[Kov94a] V.A. Kovalevsky: Topological foundations of shape analysis. In: Ying-Lie O, Alexander Toet, David Foster, Henk J.A.M. Heijmans, Peter Meer (eds.): Shape in Picture. Berlin Heidelberg New York Tokyo: Springer 1994 (Series F: Computer and Systems Sciences Vol. 126, pp. 21-36).

[Kov94b] V.A. Kovalevsky: A new concept for digital geometry. In: Ying-Lie O, Alexander Toet, David Foster, Henk J.A.M. Heijmans, Peter Meer (eds.): Shape in Picture. Berlin Heidelberg New York Tokyo: Springer 1994 (Series F: Computer and Systems Sciences Vol. 126, pp. 37-51).

[RevDeb94] J.P. Reveilles, I. Debled-Renesson: A linear algorithm for segmentation of discrete curves. Third International Workshop on Parallel Image Analysis: Theory and Applications, Washington, 1994.

[RosKaak82] A. Rosenfeld and A. C. Kaak: Digital Picture Processing. 2nd edition, Academic Press 1982.

[Whit83] T. Whitted: Antialiased line drawing using brush extrusion. SIGGRAPH 83, 151-156 (1983).