STUDIA UNIV. BABEŞ-BOLYAI, INFORMATICA, VOLUME XLIII, NUMBER 2, 1998

PICTURE APPROXIMATION

SIMONA MOTOGNA VASILE CIOBAN
VASILE PREJMEREAN

Abstract. In this paper we study a serie of image description models that use picture description languages. For 3 and 4 type images (represented by lines and curves, respectively by critical points modelling techniques are given, and also a way of passing from one form to another (*the determination of critical points* respectively Bezier *interpolation*). The contradiction between the dimension of the description and the finesse of the image (obtained by reducing the *asperities*) is solved using Bezier interpolation. The critical (interpolation) points are described by Π-words, and the curves obtained by interpolation form the type 3 image [5].

1. Introduction

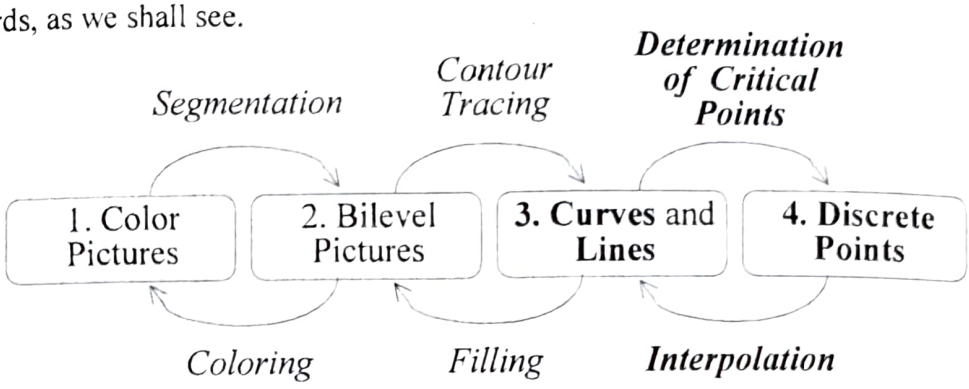
We consider the images classification (given in [5] and presented below) which groups images in four main cathegories. An image can be coloured, black&white, compound from *lines and curves* or from *points*. The techniques for transforming one image of a type in an image of another type are part of the graphics named *image processing*.

In this paper some methods of representing the images using *lines and curves* are presented, respectively a *set of critical points*, through pictures described by Π-words. There is also presented a way of converting a picture from one form to another. The Bezier curves (described in [3,5,8]) are used in the interpolation process, because they have some properties that match the drawings described by picture words.

In the following, we speak about problems referring to the 3 and 4 type images (see the diagram below), because they can be modelled using pictures described by Π-

1991 *Mathematics Subject Classification.* 68U05.
1991 *CR Categories and Subject Descriptors.* I.3.3 [Computer Graphics]: Picture/Image Generation - line and curve generation.

words, as we shall see.



2. Using Π-words to describe pictures of 3-type.

In the following a model of *approximating the type 3 pictures* [5] using Π-languages of description is presented (see lemma 2.1). Since the description of images formed from lines had already been studied [1,2,4,7] (using Π-languages), we will concentrate only on techniques for describing curves.

An image is of type 3 if it can be described completely through a finite number of lines and curves [6]. A curve can be approximated using, for example, a set of points chosen from a set of points in the 2D-plane, in a rectangular cartesian system, as in Figures 2.1 and 2.2. Connecting the points situated near the curve we will obtain a polygonal approximation formed from segments of length 1 or $\sqrt{2}$. This approximation can be represented through commands string (considering 4 or 8 directions). For example, the curve from Figure 2.1 can be approximated by the drawing described by the Π-word *urruurur*, and the curve from Figure 2.2 by *uaraura* (where *a* denotes the shifting on NE direction). This technique of passing from an image described by curves to an image described by points is called the *determination of critical points*, and the reverse process is called *interpolation* [5].

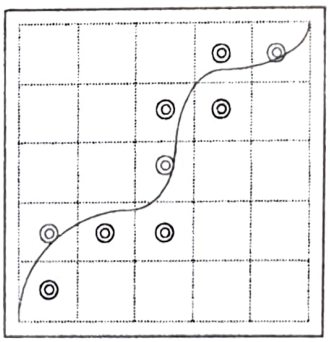


Figure 2.1

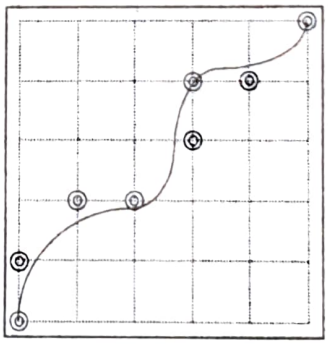


Figure 2.2

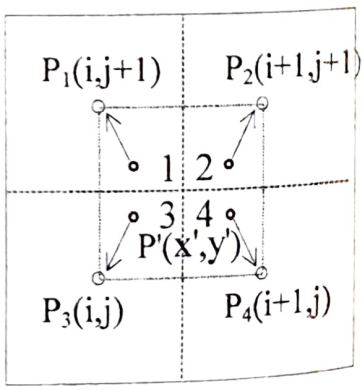


Figure 2.3

The *set of critical points* determined by the curve $c \subset R^2$ is $M_{Pc}(c)=\{Apr(P)/P \in c\}$, where $Apr:R^2 \to Z^2$, returns the point of integer coordinates that is the closest to a real coordinates point P'(x',y'), belonging to the curve (see Figure 2.3, where i=[x'] and j=[y']). The approximation function *Apr* is defined as:

$Apr(x',y') = (Round(x'), Round(y'))$, and

$Round : \mathbf{R} \to \mathbf{Z}$ is computed as $Round(x) = [x+1/2]$ (where $[x]$ denotes the integer part of x).

An image that is represented through points (type 4 image) can be improved through interpolation, yielding an image represented through curves. In order to show this, we have chosen Bezier interpolation (described in [3,5,8]), because of its basic properties:

- the curve will cross the starting and the ending points: P_1 and P_n
- the curve is tangent to the following segments: P_1P_2 and $P_{n-1}P_n$.

Figures 2.4 and 2.5 describe the curves and the determined critical points, then the descriptions of the correponding Π-words (for 4 and 8 directions) and finally the corresponding Bezier curves.

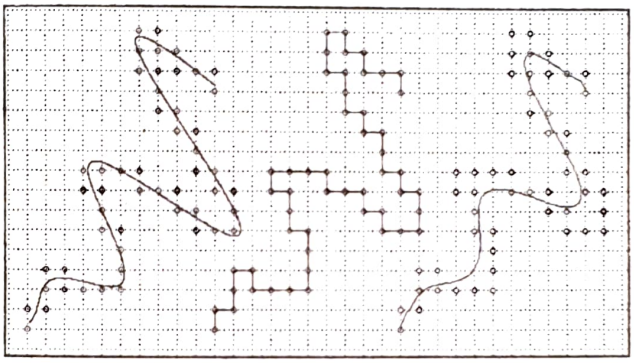


Figure 2.4

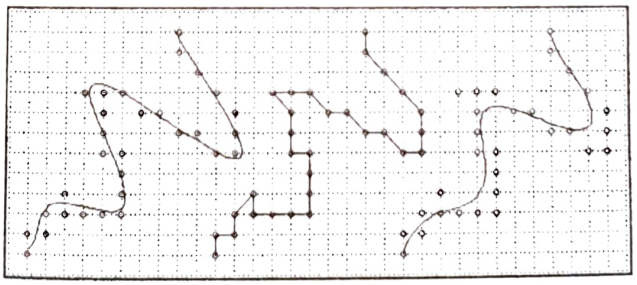


Figure 2.5

The way in which the Bezier curve (from Figure 2.6) is determined starting from the points string $P_1,P_2,...,P_n$ is described below and can be observe on Figure 2.7 for n=4.

The initial points string $P_1, P_2, \ldots, P_n$ is used in order to determine another points string $P_{1,2}, P_{2,2}, \ldots, P_{n-1,2}$ using the equation (2.1), then using the same formula we determine the points $P_{1,3}, P_{2,3}, \ldots, P_{n-2,3}$ and so on until we obtain a single point $P_{1,n}$. This point obtained for a value $\alpha \in [0,1]$ belongs to the Bezier curve. Selecting values for α from this domain we will obtain the desired approximation (in Figure 2.7, $\alpha=1/4$). One may notice that for $\alpha=0$ we obtain P_1, and for $\alpha=1$ we obtain P_n.

The equation for determining the points on level $m+1$ depending on the points from level m is the following:

$$P_{i,m+1}(\alpha)=(1-\alpha)\cdot P_{i,m}(\alpha)+\alpha\cdot P_{i+1,m}(\alpha)\ ;\ i=\overline{1,n-m};\ m=\overline{1,n-1};\ \alpha=\overline{0,1},\ 1/(s-1) \qquad (2.1)$$

As a remark, we notice that the equation is used starting from the second level, since the first level contains the given points, namely $P_{i,1}(\alpha)=P_i$, $i=\overline{1,n}$.

If we want to determine s points $B_1=P_{1,n}(\alpha_1)$, $B_2=P_{1,n}(\alpha_2)$, ... , $B_j=P_{1,n}(\alpha_j)$, ... , $B_s=P_{1,n}(\alpha_s)$ corresponding to the values $\alpha_1=0$, $\alpha_2=1/(s-1)$, ... , $\alpha_j=(j-1)/(s-1)$, ... ,$\alpha_s=1$, we will compute the coordinates of points $P_{1,n}(\alpha_j)$ for each value $\alpha_j(j=\overline{1,s})$. In order to obtain more points besides the starting and ending points (because B_1 is identical to P_1, and B_s is identical to P_n, see Figure 2.6 for s=4), we must require s to be greater than 2.

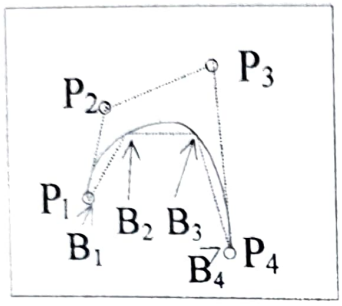


Figure 2.6

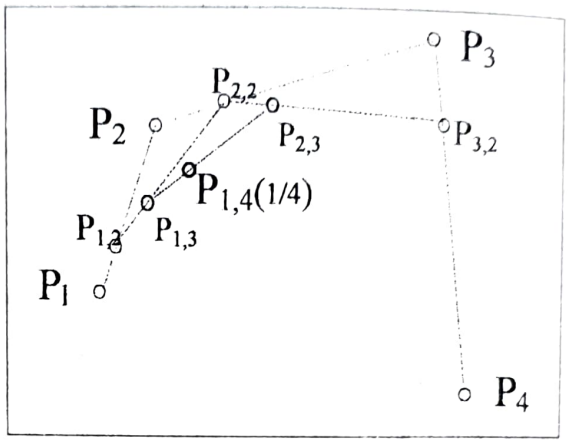


Figure 2.7

The *Bezier algorithm* for determining s points (B_j, $j=\overline{1,s}$) from a Bezier curve is:

Algorithm 2.1.

Input : P_i, $i=\overline{1,n}$

Output : B_j, $j=\overline{1,s}$

For i:=1,n *Do* $P_{i,1}:=P_i$;

For j:=1,s *Do Begin*

α:=$(j-1)/(s-1)$;

For m:=1,$n-1$ *Do*

For i:=1,$n-m$ *Do*

$P_{i,m+1} := (1-\alpha)\cdot P_{i,m} + \alpha \cdot P_{i+1,m}$;

$B_j := P_{1,n}$

End.

The result produced by this algorithm consists of a string of points $B_1,B_2,...,B_s$. Theoretically, the Bezier interpolation curve (corresponding to the points $P_1,P_2,...,P_n$) is:

$$Bezier(P_1,P_2,...,P_n) = \{ P_{1,n}(\alpha) / \alpha \in [0,1] \}.$$

Practically, the graphical representation of the curve is obtained drawing the s-1 segments determined by the string B_1, B_2, ... , B_s , namely

$Bezier_{Seg}(P_1,P_2,...,P_n) = \{ \overline{B_jB_{j+1}} / j=\overline{1,s-1} \}$ (see Figure 2.6).

If we study Figure 2.4 and 2.5, we notice that if we start from a curve c and determine the set of critical points $M_{Pc}(c)$, then, using the *Bezier* algorithm, we compute the points B_j, j=1,s , these points differ from the initial critical points, namely $M_{Pc}(c) \neq \{Apr(B_j) / j = 1,s\}$, and the curve $c' = Bezier_{Seg}(M_{Pc}(c)) \neq c$. Certainly, we would like to obtain a curve through interpolation such that it would be as close as possible to the initial curve:

$$\{Apr(P_{1,n}(\alpha)) / \alpha \in [0,1]\} = \{Apr(P) / P \in c\}. \tag{2.2}$$

Lemma 2.1.

A picture formed from lines and curves can be *approximated* by a Π-language.

Proof. We suppose that we have a curve c_0, and we want to determine the point string s_0 of closest integer coordinates. This string can be described by a Π-word w such that the set of points belonging to the drawing (see equation 2.4) described by w is identical to the set of critical points corresponding to the curve c_0, namely:

$$V(w) = \{Apr(P) / P \in c_0\} \tag{2.3}$$

where the set of points described by w is denoted by $V(w) : \Pi^* \to Z^2$ and it is defined as follows:

$$V(w) = \begin{cases} \varnothing & \text{if } w=\lambda, \\ V(z) \cup \{Sh(w)\} & \text{if } w=z\tau \ (\tau \in \Pi, z \in \Pi^*); \end{cases} \tag{2.4}$$

and $Sh : \Pi^* \to Z^2$:

$$Sh(w) = \begin{cases} P_1 & \text{if } w = \lambda, \\ \tau(Sh(z)) & \text{if } w = z\tau \ (\tau \in \Pi, z \in \Pi^*); \end{cases} \tag{2.5}$$

and the shifting through the command $\tau \in \Pi = \{r,u,l,d\}$ is defined in a classical way (see [1,4,7]):

$$r(x,y) = (x+1, y); \quad l(x,y) = (x-1, y); \quad d(x,y) = (x, y-1); \quad u(x,y) = (x, y+1).$$

According to equation (2.3), any line or curve can be approximated through a Π-word, which implies that the set of Π-approximation-words forms the Π-approximation-language of the image.

Next, if we want to determine the curve c_1, using Bezier interpolation, corresponding to the point string $s_0 = (Sh(a_1...a_i), i=\overline{1,n})$, obtained from the Π-word $w=a_1...a_n$ (using equation 2.5 for the definition of the function *Sh*), then it is very rare the case (even if we would like to) that we will obtain the same initial curve c_0.

If we apply repeatedly the same procedure we will obtain a string of curves $c_0, c_1, c_2, ..., c_p, ...$. The question that arises in a natural way is the following: "How can these transformations be done such that the curve string would be finite? ".

If we are able to obtain an interpolation curve that preserves the critical points, then $s_1=s_0$, and therefor the approximation curve will no longer be modified when the transformations are applied:

$$c_0 \xrightarrow{s_0} c_1 \xrightarrow{s_0} c_1 \xrightarrow{} \ldots \xrightarrow{}$$

namely:

$$c_p = \begin{cases} c_1 & \text{If } p \text{ is odd,} \\ s_0 & \text{If } p \text{ is even,} \end{cases} \tag{2.6}$$

Remark 2.1.

This property (2.6) involving the two transformations (interpolation and determination of critical points) has its own importance, because each transformation is the reverse of the other one. For example, after the codification of an image (for the purpose of compression), the decodification (in other words, the reverse transformation) will be achieved through an interpolation (that preserves the critical points, and also the Π-words description).

In the following we present the conditions that *conserves the critical points* (lemma 2.2) and also a description (algorithm 2.2) so that this property (useful for example in the process of compressing and de decompressing or encoding and decoding) is satisfied.

In this purpose, we present a transformation (through Bezier interpolation) that preserves the description Π-word.

In order to obtain a point string as closer as possible to the critical points, we will group the points $P_1,P_2,\ldots,P_n$ into substrings of a certain length q (for example, in Figure 2.8 the groups are formed from 5, respectively 7, points), then we will apply the *Bezier* algorithm for each substring, and the resulting curve will be obtained through reunion of the curves determined from these substrings:

$$c = Bezier\ (P_1,\ldots,P_q) \cup Bezier\ (P_q,\ldots,P_{2q-1}) \cup \ldots \cup Bezier\ (P_r,\ldots,P_n). \tag{2.7}$$

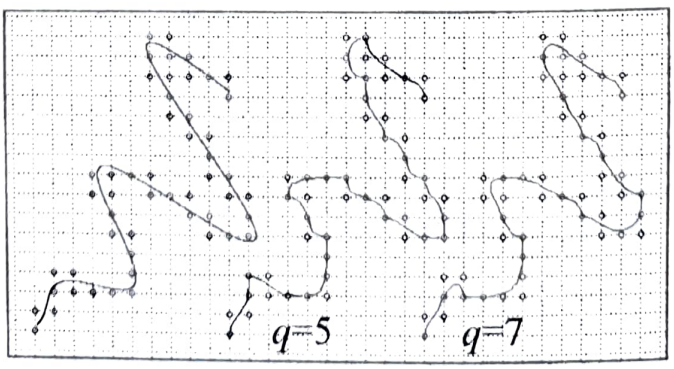


Figure 2.8

Lemma 2.2.

The transformation 2.7 (using groups of length $q<7$) *preserves the critical points* ($Apr(c)=\{P_1,P_2,...,P_n\}$), and also the description Π-word (w), if it doesn't contain *retreats* ($w\in ref(\Pi^*)=\{w\in\Pi^* / \{y / w=xyz\}\cap \{ ud, du, lr, rl \}=\varnothing \}$).

The proof will be done for each q starting from 2 to 6.

It is easy to see for $q=2$, $Apr(Bezier(P_1,P_2, \dots ,P_n)) = \{P_1,P_2, \dots ,P_n\}$, because $Bezier(P_1,P_2,...,P_n) = \{ P_iP_{i+1} / 1\le i < n \} = dpic(w)$, where w is the Π-word that describes the path $P_1,P_2,...,P_n$. The set of horizontal and vertical lines of unit length described by the Π-word w denoted by $dpic(w)$ is defined as follows:

$$dpic(w) = \begin{cases} \varnothing & \text{if } w=\lambda, \\ dpic(z)\cup\{\overline{Sh(z)\,Sh(w)}\} & \text{if } w=z\tau \ (\tau\in\Pi, z\in\Pi^*); \end{cases} \tag{2.8}$$

(Sh was defined in formula 2.5)

For $q=3$, the property is satisfied, because for any point substring P_{k-1}, P_k, P_{k+1}, these points are either collinear (in which case the curve is situated on the line determined by these points), either form a rectangular triangle (in which case, $Apr(Bezier(P_{k-1},P_k,P_{k+1})) = \{P_{k-1},P_k,P_{k+1}\}$) .

In the case in which the points are collinear, namely they belong to the path described by τ^q ($\forall\tau\in\{r,u,l,d\}$ and $\forall q\ge1$), the critical points are preserved (the Bezier curve is exactly the segment (P_1, $Sh(\tau^q)$), so we will disregard this case.

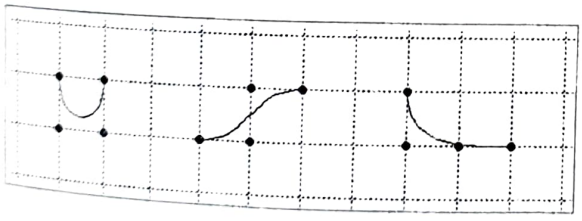


Figure 2.9

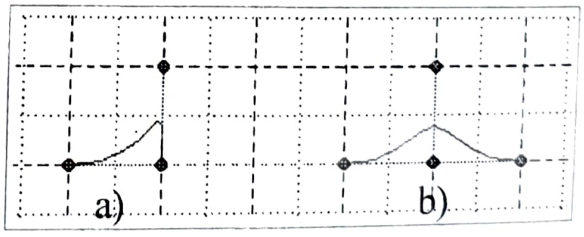


Figure 2.10

For $q=4$, the property is satisfied, because in any of the three situations presented in Figure 2.9, the critical points are preserved.

For $q\in\{4,5\}$, the Bezier curves presented in Figure 2.10, corresponding to the Π-words $w=rud$, respectively $w=rudr$, do not preserve the critical points. So, if $w\notin ref(\Pi^*)$, for $q\ge4$, the property is no longer true.

For $q=5$, and $w\in ref(\Pi^*)$ the property is satisfied (see figure 2.11), because in any of the nine cases, the property is true.

If we study the Figure 2.12 we may notice that for $q=6$, any combination of points described by a Π-word without retreats, preserves the critical points (from the 62

classes, only 25 are described through Π-words without retreats, and from these only one is described through r^5).

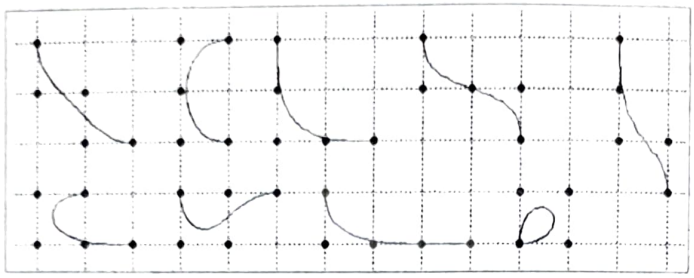


Figure 2.11

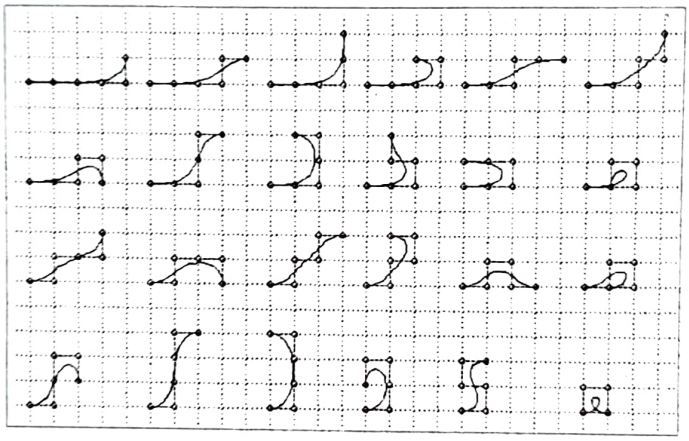


Figure 2.12

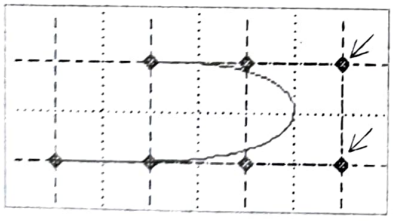


Figure 2.13

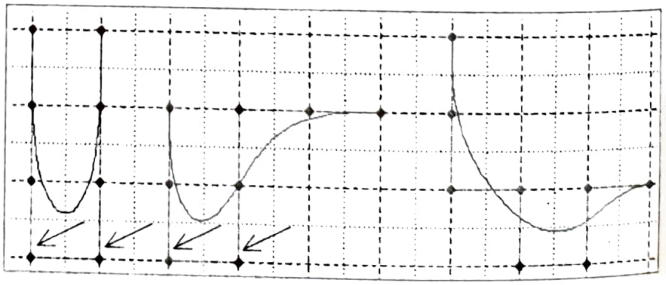


Figure 2.14

For q=7, the property is no longer valid for any description Π-word, even independent of retreats. For example, for *w=rrrull* , the situation is shown in Figure 2.13.

Obviously, for $q \geq 8$, there are situations in which some points remain "uncovered" (see the first two curves from Figure 2.14, that have only six critical points), and other situations that preserve the critical points (the last curve from the same figure).

It is much more efficient to group the initial points $P_1,P_2,...,P_n$ in as few substrings as possible (fewer Bezier curves) that conserves the critical points (without establishing a number q of points in the substrings). This means that we try to form substrings, such that their length is as big as possible. We will add a point P_{j+1} to a substring $P_i,P_{i+1},...,P_j$

that preserves the critical points only if the string $P_i,P_{i+1},...,P_jP_{j+1}$ preserves the critical points (and then we will continue with point P_{j+2} and so on), otherwise we will start to form a new group $P_j,P_{j+1},...$. In this manner, we will form substrings of length at most six, depending on the relative position of the points.

The following algorithm determines the minimal number (denoted by k) of *groups* that preserve the critical points.

Algorithm 2.2.

Input: $P_1, P_2, ..., P_n$;

Output: $Group_1, Group_2, ..., Group_k$;

Begin

k:=0; i:=1;

Repeat

j:=i+5;

While j>n *Do* j:=j−1;

While (j<n) *And* $Apr(Bezier(P_i, ..., P_j, P_{j+1})) = \{P_i, ..., P_j, P_{j+1}\}$ *Do* j:=j+1;

k:=k+1; $Group_k := (P_i, ..., P_j)$; i:=j

Until i=n;

End.

The problem of the *optimality of the description* (Remark 2.2) and *the quality of the picture* are studied below.

Remark 2.2.

The minimal number of *groups* that conserves the points P_1, P_2,..., P_n is $NrGr(1,n)$, where:

$$NrGr(i,j) = \begin{cases} 1 & \text{if } Apr(Bezier(P_i,...,P_j)) = \{P_i,...,P_j\}, \quad (a) \\ \underset{i<l<j}{Min}\{NrGr(i,l)+NrGr(l,j)\} & \text{if } Apr(Bezier(P_i,...,P_j)) \neq \{P_i,...,P_j\}. \quad (b) \end{cases}$$

If $Lmin(i,j)$ is the index l for which the minimum from the precedent formula is reached for case b) or is equal to 0 for case a), then the sequence of points $P_1,P_2,...,P_n$ can be optimally divided in two: $P_1,P_2,...,P_{Lmin(1,n)}$ and $P_{Lmin(1,n)},...,P_n$. Each of $P_i, ...,P_j$ are separated again in two by the point $P_{Lmin(i,j)}$ (if $Lmin(i,j)>0$), and so on until all the subsequences have the separating index $Lmin(i,j)$ equal to zero. We observe that the number of obtained sequences is minimal.

In order to make this algorithm faster (so some calculations won't be repeated) it is recommended to store the values $NrGr(i,j)$ already computed in a square matrix M of order n. The value $NrGr(i,j)$ will be stored in M_{ij}. We will successively determine the values of this matrix for the first diagonal parallel to the main diagonal, then for the second and so on until we get to M_{1n} which represents the value that we are looking for. Using lemma 2.2, we can avoid calculations for the first six diagonals, all of them having the values equal to one. Practically, the computation of the values starts with the seventh diagonal. In order to form the optimal subsequences (we're not interested only in the minimal number of groups given by M_{1n}) we will store in element $M_{j,i}$ the index for which the minimum of the function is obtained, for the subsequence of the points from i to j (we can do this because the elements under the main diagonal are not used).

We shall build a binary tree that contains in each node pairs (i,j), where i and j represent the limits of a determined optimal subsequence $(P_i, \ldots, P_j)$. This tree has the root $(1,n)$. If for a node (i,j) of the tree we have $Apr(Bezier(P_i, \ldots, P_j)) = \{P_i, \ldots, P_j\}$ then this node is a terminal one (leaf), otherwise it will have two subtrees with the roots (i,M_{ji}), respectively (M_{ji},j). The front of this tree will give us the optimal subsequences.

It is obvious that the algorithm 2.2 and the one presented above (using dynamic programming) do not necessarily give the same solution, but both solutions are minimal. If there are more ways of separating a subsequence of points, the separation in point P_i is preferred, if the points P_{i-1}, P_i and P_{i+1} are collinear (this assures the derivability of the Bezier curve, because it is tangent to the edges of the polygonal curve determined by the interpolation points). In this way the curve has less turning points, so less *asperities*.

3. Using Π.-words to describe line drawings.

In this paragraph we present a model of describing a type 3 image (using the classification given in [5]) using Π.-words (that contains two more symbols ↑=*pen-up* and ↓=*pen-down* denoting the lift and sink of a plotter pen). These words will describe a sequence of points that determines a curve by Bezier interpolation. This means that a set of Π.-words describes an image formed by curves, a type 3 image.

Starting from a certain point $P_1(x_1, y_1)$ (which can be even the origin, in order to define the standard representative, like in [4]), using a Π.-word of description (used in [4] for disconnected picture) $w=a_1a_2\ldots a_m$, we shall build the sequence of interpolation points $P_2(x_2, y_2)$, $P_3(x_3, y_3)$, ... , $P_n(x_n, y_n)$. From the sequence of the points crossed by the commands of the Π.-word w, only the points crossed with the pen down (↓) will be selected for the interpolation. *The position of the pen* has been specified by the function *Pc* from formula 3.1:

$$P_c : \Pi.^* \to \{\uparrow,\downarrow\}, \quad P_c(w) = \begin{cases} \uparrow & \text{if } w=z\uparrow \ (z\in\Pi.^*) \text{ or } w=\lambda, \\ \downarrow & \text{if } w=z\downarrow \ (z\in\Pi.^*), \\ P_c(z) & \text{if } w=z\tau \ (z\in\Pi.^*) \text{ and } \tau\in\Pi.^*. \end{cases} \tag{3.1}$$

The functions $Nr : \Pi.^* \to \mathbf{N}$ and $Pb : \Pi.^* \to \mathbf{Z}^2$ defined below will specify which is the last point (from the sequence of points that are to be interpolated) selected in the crossing (this being given by the number of selected points, meaning the points crossed with the pen down). The point $P_i(x_i, y_i)$ is specified by its *position* (index $i = Nr(a_1a_2...a_{j-1})$) in the sequence of interpolation points $(P_1,P_2,...,P_n)$ and its coordinates (x_i, y_i) computed using the function $Pb(a_1a_2...a_{j-1})$. These formulas are applied for each j=2,3,...,m.

$$Nr(w) = \begin{cases} 1 & \text{if } w=\lambda, \\ Nr(v) & \text{if } w\neq\lambda,\ w=v\tau \text{ and } P_c(w) = \uparrow, \\ Nr(v)+1 & \text{otherwise.} \end{cases} \quad (3.2)$$

$$Pb(w) = \begin{cases} Pb(v) & \text{if } w\neq\lambda,\ w=v\tau \text{ and } P_c(w) = \uparrow, \\ Sh.(w) & \text{otherwise.} \end{cases} \quad (3.3)$$

The function $Sh. : \Pi.^* \to \mathbf{Z}^2$ gives the last point touched by a Π.-word (named *the shifting by w*, using Π-word w) and it is defined as follows:

$$Sh.(w) = \begin{cases} P_1 & \text{if } w=\lambda, \\ Pos\,(\tau, h(Sh.(v))) & \text{if } w\neq\lambda,\ w=v\tau \text{ and } \tau\in\Pi, \\ Sh.(v) & \text{otherwise } (\tau\in\{\uparrow,\downarrow\}). \end{cases} \quad (3.4)$$

The function $Pos : \Pi^* \times \mathbf{Z}^2 \to \mathbf{Z}^2$ gives the coordinates of the last point touched by a Π.-word (called *the shifting by w*, and is defined by formula 3.5), and the erasing homomorphism $h : \Pi.^* \to \Pi^*$, defined by formula 3.6.

$$Pos\,(w, (x,y)) = \begin{cases} (x,y) & \text{if } w = \lambda, \\ Pos\,(z, \tau(x,y)) & \text{if } w = \tau z\ (\tau\in\Pi, z\in\Pi^*); \end{cases} \quad (3.5)$$

and

$$h(w) = \begin{cases} \lambda & \text{if } w = \lambda, \\ h(z) & \text{if } w = \tau z\ (\tau\in\{\uparrow,\downarrow\}, z\in\Pi.^*) \\ \tau\, h(z) & \text{if } w = \tau z\ (\tau\in\Pi, z\in\Pi.^*). \end{cases} \quad (3.6)$$

The possibility of describing a type 3 image using Π.-languages is pointed out in lemma 3.1.

Lemma 3.1.

A type 3 image can be described using a Π-language.

Proof. Every Π-word describes a point string as we presented above, through the functions *Nr* and *Pb*, that are defined in (3.2) and (3.3), and using the Bezier interpolation for the constructed strings we will obtain the curves which compose the image.

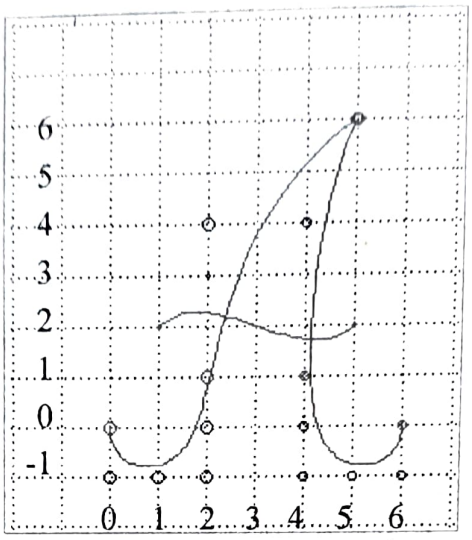


Figure 3.1

The drawing shown in Figure 3.1 can be *approximated* by the three following picture words (by which the critical points that are to be interpolated are determined):

$$w_1 = dr^2u^2\uparrow u^2\downarrow u\uparrow u^2r^2\downarrow r,$$

$$w_2 = \uparrow ld\downarrow d\uparrow d^2\downarrow d^3r^2u\uparrow u^2l$$

and

$$w_3 = \uparrow d\downarrow l\uparrow l^2u\downarrow u\uparrow l\downarrow d$$

and describe the following three point strings:

1. (0,0),(0,-1),(1,-1),(2,-1),(2,0),(2,1),(2,4),(5,6);
2. (5,6),(4,4),(4,1),(4,0),(4,-1),(5,-1),(6,-1),(6,0) and
3. (5,2),(4,1),(2,3),(1,2).

Interpolating these points, we will obtain the three curves representing the image of the *handwriting capital letter A*, from Figure 3.1.

In a similar manner one may define Π-words for describing mathematical symbols, used for example in writing equations.

REFERENCES

1. F.J. Brandemburg, M.P. Chytil, *On Picture Languages : Cycles and Syntax - Directed Transformations*, Technische Berichte der Fakultät für Mathematik und Informatik Universität Passau, MIP-9020, 1990.

2. J. Dassow, F. Hinz, *Decision problems and regular chain code picture languages*, Discrete Applied Mathematics, no.45, 1993, pp. 29-49.

3. J.D. Foley, A.V. Dam, *Fundamentals of Interactive Computer Graphics*, Addison Wesley, London, 1982.

4. H.A. Maurer, G. Rozenberg, E. Welzl, *Using String Languages to Describe Picture Languages*, Information and Control, Vol.54, Nr.3, 1982, pag.115-185.

5. T. Pavlidis, *Algorithms for Graphics and Image Processing*, Springer-Verlag, Berlin-Heidelberg, 1982.

6. A. Rosenfeld, *Picture Processing by Computer*, Academic Press, London, 1969.

7. I.H. Sudborough, E. Welzl, *Complexity and Decidability for Chain Code Picture Languages*, Universität Graz, F125, 1983.

8. Watt, *3D Computer Graphics*, Addison-Wesley, Great Britain, 1993.

Babeş-Bolyai University, Faculty of Mathematics and Informatics, RO 3400 Cluj-Napoca, str. Kogalniceanu 1, România.

E-mail address: {vcioban,motogna,per}@cs.ubbcluj.ro