New Method to Find Corner and Tangent Vertices in Sketches using parametric cubic curves approximation

F. Albert[a], D.G. Fernández-Pacheco[b], N. Aleixos[a*]

[a] *Instituto Interuniversitario de Investigación en Bioingeniería y Tecnología Orientada al Ser Humano (Universitat Politècnica de València), Camino de Vera s/n, 46022-Valencia, España {fraalgi1,naleixos@dig.upv.es}*
[b] *DEG, Universidad Politécnica de Cartagena, 30202-Cartagena, España, daniel.garcia@upct.es*

[*] *Corresponding author: Nuria Aleixos, naleixos@dig.upv.es, Tel.: +34 387 95 14, Fax: +34 387 75 19*

Abstract—Some recent approaches have been presented as simple and highly accurate corner finders in the sketches including curves, which is useful to support natural human-computer interaction, but these in most cases do not consider tangent vertices (smooth points between two geometric entities, present in engineering models), what implies an important drawback in the field of design. In this article we present a robust approach based on the approximation to parametric cubic curves of the stroke for further radius function calculation in order to detect corner and tangent vertices. We have called our approach Tangent and Corner Vertices Detection (TCVD), and it works in the following way.

First, corner vertices are obtained as minimum radius peaks in the discrete radius function, where radius is obtained from differences. Second, approximated piecewise parametric curves on the stroke are obtained and the analytic radius function is calculated. Then, curves are obtained from stretches of the stroke that have a small radius. Finally, the tangent vertices are found between straight lines and curves or between curves, where no corner vertices are previously located. The radius function to obtain curves is calculated from approximated piecewise curves, which is much more noise free than discrete radius calculation. Several tests have been carried out to compare our approach to that of the current best benchmarked, and the obtained results show that our approach achieves a significant accuracy even better finding corner vertices, and moreover, tangent vertices are detected with an Accuracy near to 92% and a False Positive Rate near to 2%.

Index Terms—corner and tangent vertices detection, hand-drawn and sketch segmentation, image object recognition, natural interfaces

1 INTRODUCTION

IN order to achieve interfaces that support natural human-computer interaction, it is necessary to develop intelligent techniques for the automatic recognition of sketches that allow users to draw as they naturally would without any constraints, like introducing the sketches in a particular order or requiring a previous training by the user to learn a set of specified symbols or shapes. Actually, sketching is an established part of the engineering culture, but current available tools for Computer Aided Sketching (CAS) supported by CAD (Computer Aided Design) applications are not yet as usable as paper-and-pencil, owing to the lack of many necessary functionalities and flexibility [1]. This work also shows the importance of sketching in conceptual design and presents the current state of the art in CAS tools by describing the main features and outstanding problems in this topic.

One important feature that should be considered by free-hand sketch systems deals with the problem/ need/ restriction of drawing complex shapes in a single stroke. In order to support this feature, it is necessary to split the stroke into its constituent primitives, what involves the development of techniques capable of finding vertices in the stroke. Once the vertices are found, the stretches between vertices could be approximated to primitives' straight lines or curves, hence, user intent design could be captured maintaining the tangency between lines and curves or between curves.

The procedure to find vertices is so called finding corners or segmentation process, and although some works have been carried out to find vertices, the segmentation of sketched shapes still remains unsolved because this is a very complex task, as stated by Company et al. [2].

Apart from sketch-based systems, poly-line corner finding is a very powerful tool for other types of applications, as Wolin et al. shows in [3]. In this work they developed an algorithm they called ShortStraw to find corners in strokes and compared the obtained results to current baseline corner finders. This algorithm was found to be highly accurate in both total correct corners and all-or-nothing corner accuracy benchmarks in strokes containing straight lines. Later, this algorithm was improved by Xiong and La Viola [4], presenting a new corner finding algorithm, IStraw, that overcome some limitations and attempted to reduce the lacks while maintaining the computational complexity. This algorithm also extended ShortStraw to deal with strokes containing curves and the obtained results showed significant improvements in all-or-nothing corner accuracy compared to ShortStraw.

Although these works have developed algorithms that offer good results in detecting corners, and have been presented as two of the most accurate and simple compared to other existing methods, they still present some drawbacks to be solved, as for instance the detection of tangent vertices. A proposal that takes into account tangent vertices was presented by Pu and Gur [5]. This is a more complex method compared to previous ones, and uses mathematical curves to approximate the stroke, considering vertices as those being the minimum set of points that can be used to reconstruct with high accuracy the stroke. This method presents two important inconveniences: first, it does not distinguish between corner vertices and tangent vertices, what is a drawback from the point of view of the continuity between parts of the stroke; second, although a high refinement post-process is done, the number of false positive in the detection of vertices is very high.

But, why it is important to detect the tangent vertices (smooth transitions) in the strokes? The main motivation for tangent-finding vertices is that the geometry of sketches has to be approximated to their corresponding primitives in order to create the out-lined section to later generate 3D models, models that in most cases have tangent transitions between planar-curved surfaces or curved-curved surfaces, what makes essential the detection of the designer intention in the sketches by means the finding of tangent vertices.

As we will see from related work, almost all methods presented find corner vertices but nearly no method is capable of finding tangent vertices (smooth transitions) acceptably in sketches. The aim of this article is to present a new approach, TCVD (Tangent and Corner Vertices Detection), based on the radius function, calculated first from differences between stroke points (discrete radius) and then from the stroke approximation to parametric cubic curves (analytic radius), to find so corner vertices as tangent vertices in sketched shapes including curves, and to obtain a recognised parametric equivalent stroke with the corresponding continuity in the tangent vertices found. This method has been compared to some of the most successful ones in the state of the art and aimed to the same objective or easily adapted, such as the already mentioned ShortStraw, IStraw and the presented by Pu and Gur. The results of comparison show our technique as the most accurate one, and with a few false positives in both corner and tangent vertices.

This paper is organised as follows. In the next section we present an overview of the state of the art in techniques to find corners and segmentation of sketches, including a justification of the methods chosen for comparison at the end of the section. A detailed description of the TCVD method is covered in section 3. Section 4 describes the experimental work carried out and discusses this method against other three approaches, including results of the temporal cost of the algorithm. Sections 5 and 6 show the conclusions and the expected further work in this field. Finally, at the end of this article, an appendix of the approximation by means of parametric cubic curves is provided.

2 RELATED WORK

In short, the challenge of replacing conventional pencil and paper sketches with a digital sketching environment exists. This new environment must be designed in such a way that it favors a natural process that does not hinder the user, while also producing its output in the form of a digital design model that can be reused in the remaining phases of the design process.

Multiple techniques are used in sketch recognition to detect or classify regular geometric shapes [6-8], handwriting characters [9, 10], fingerprints [11], electric circuits [12, 13], diagrams [14, 15], and other user command gestures. For instance, with a classic linear discriminator, Rubine [16] calculated features in order to classify single-stroke sketches as digits, letters and basic commands introduced in a specific way. Also based on similar features Ayaj et al. [17] distinguished five simple geometric shapes basing their classification on thresholds to the ratio filters established. Gross [18] described a prototype for the recognition of glyphs, but his algorithm also required sketching in a strict order. Other features that remain invariant with rotation, such as convex hull, perimeter and area scalar ratios, were studied by Fonseca et al. [19], who used ratio values in fuzzy sets to recognize a small set of shapes. Xiangyu et al. [20] and Zhengxing et al. [21] recognized simple geometric shapes by calculating the average distance from the vertices of the preset shape to the vertices of the stroke. Willems et al. [22] established different feature subsets using features as length, average curvature, initial angle, absolute curvature, number of crossings, area, rectangularity, compactness, etc. up to a total of 48 global features, compiled from various works from the literature and using Support Vector Machines (SVM), Multilayer Perceptrons (MLP) and Dynamic Time Warping (DTW) classifiers in order to classify multi-stroke gestures obtaining different performances depending on the subset used.

In most of these and similar works, the vertices of sketches have to be located, since it is an essential key to recognize or interpret a sketched shape. If we search in literature, we can find works aimed at finding corner vertices, so called corners, in figures and sketches. For instance, for detecting corners in digital objects with curves, Zhang and Zhao [23] uses a boundary-constrained morphological method for tilting closed curves into shapes, and after the morphological residues are labeled as candidates of corner sets, the definitive corners can be obtained by reducing the sets to corresponding isolated points. The main disadvantage of this method is that corners are not detected in objects with large differences in their corner sizes, and does not deal with finding vertices in smooth transitions. One of the most wide used techniques is the Gaussian scalespace [24, 25], where a set of progressively smoother versions of the shape is generated by applying series of Gaussian filters with different standard. Using the same shapes in two previous works, Neumann and Teisseron [26] detected corners in two steps after assigning support regions to the shape: first, points having higher curvature than the estimated fluctuations over the support region assigned are extracted as candidates; second, adjacent candidates are merged and a final elimination of corners on each region is performed. Like the previous work, this method neither finds vertices in smooth transitions. Arrebola and Sandoval [27] proposed a method to characterise a curve by means of a hierarchical computation of a multiresolution, that is, a successive lower resolution versions of the same shape, that is processed using a linked pyramid in order to segment and detect contour features, but with many false positive, what makes this method unsuitable for the purpose stated in this article.

Sezgin et al. [28] search peaks in curvature and speed functions, and found real corners after combining the candidate previously found as corners with high curvature and low speed values. Although some authors as [4, 28, 29] also used time information to detect corners, its use is not extended since the results are not definitive about the reliability of this information. Scale based approaches have also been used to detect corners in strokes, as in the case of Sezgin and Davis

[30] who used scaled curvature data to remove noise and locate better the corners. Kim and Kim [31] resampled the input to provide constant distance along the stroke, that is, normalised the stroke, then they calculated the curvature as the change of direction at each point and other new curvature metrics as local convexity and local monoticity to find corners. In both works the finding of vertices in smooth transitions is not stated.

Yu and Hse et al. [32, 33] used both segmentation and primitive approximation to find dividing points. The segmentation of the stroke consists of breaking down the stroke into its constituent primitives, what implicitly removes the noise from the stroke. In the case of Yu the algorithm used an iterative technique. It recursively tried to approximate the stroke to a primitive, and in the case the approximation failed, the stroke was divided in stretches, repeating this procedure until all the split stretches were approximated by a straight line or an arc. Other examples of finding vertices can be found in [34] where a stroke is broken down and then its primitives can be recognized with high accuracy, and after they are recombined using geometrical rules [35, 36].

In segmentation of strokes, the process normally done is first identifying segmentation points, then classifying the sub-strokes between each pair of adjacent segmentation points, and finally, approximating them to primitives. Examples of works that first segment the strokes before facing the corner finding are presented here. For instance, Sarkar et al. [37] used genetic algorithms to fit digital curves to line segments and circular arcs. In order to avoid noise in identifying segmentation points and to obtain a later homogeneous segmentation in sketches, Zhang et al. [38] first extracted graphical primitives from a stroke by a connected segment growing from a seed-segment and then utilised relationships between the primitives to refine their control parameters. Also with noisy curves, Nguyen and Debled-Rennesson [39] applied two methods, one based on a fixed parameter that is the width of considered maximal blurred segments, and other one (deduced from the previous one) based on a multi-width approach to obtain a non-parametric approach without thresholds. In this case the curves are always fitted to several lines what makes this method just limited to find corners. Wolin et al. [3] built a simple and effective corner finder for strokes composed only by straight lines. They called this algorithm ShortStraw, which was later modified by Xiong and La Viola [4] with their IStraw to allow the strokes containing curves. Although none of the two methods are aimed to find smooth transitions, that is, tangent vertices, their results in finding corners are highly satisfactory compared to the rest of works found in literature, and the second one, IStraw, present results on sketches including curves. Besides, both ShortStraw and IStraw can be easily implemented, what made us decide for comparing them to our method.

Thus, Qin et al. [40] presented an on-line procedure based on heuristic adaptive thresholds (local thresholds depending on drawing speed) used in all stages of the procedure, even in the subsequent refinement processes. They first remove close points, then obtain corner vertices as maximums of directional deviation, later divide the stroke in spans or stretches between consecutive corners, and finally obtain tangent vertices as changes in the sign of curvature. Apart from using a lot of thresholds, they do not avoid the noise effects and use for the evaluation a poor and limited data set: only 22 sketches, without features of engineering drawings that include tangent vertices between straight lines and curves, so there is not any indicator to assess the method proposed.

As we have seen from this state of the art, many research works just deal with polyhedral models (i.e. [1]) or reconstruct 3D models from simple sketches of isolated lines or arcs (i.e. [41]), because the main lack of obtaining curved models from sketches, necessary in most of engineering models, is that segmentation algorithms are not capable of detecting smooth transitions from straight lines to curves or between curves, and those that try to detect this kind of transitions are not robust, mainly due to the bad results obtained or to the high number of false positives they reach. In this respect, Pu and Gur [5] try to find this kind of smooth transition points in a reasonably robust way using radial basis functions. The performance of their algorithm is quite promising, but, as they say in their conclusions, further improvements and refinements have to be done in order to reduce the false positive rate using other approaches because it still remains very high (about 25%).

The new approach presented here intends to solve this problem, providing a high accuracy detection of this kind of vertices, with a low false positive ratio and without any refinement, so capturing the intent design of the user in order to allow the further creation of 3D models with tangent surfaces. Given the impossibility of implementing all methods, and in order to give objective results of the high accuracy of the presented method in the detection of vertices, it has been compared to the current best benchmarked in finding corners and tangent vertices, selecting three methods from the stated art presented here: ShortStraw, IStraw and Pu&Gur methods. ShortStraw has been selected due to its high ratio of success in finding corners, and IStraw since it improves the accuracy of ShortStraw in finding corners and also deals with shapes including curves. Other advantage is that both methods are easy to implement and have simple complexity compared to other existing ones. Pu&Gu, however, has a higher complexity, but is the only method found aimed at finding tangent vertices. Its complexity is higher and so it has not been implemented, thus the comparison is relative. In the next sub-sections the three methods selected are described briefly.

2.1 ShortStraw

In 2008, Wolin et al. introduced ShortStraw [3], "a simple and effective corner finder for polylines" achieving a high accuracy finding corners in polyline strokes, that is, strokes without curves. ShortStraw uses first a bottom-up approach to obtain the initial corner set, and then a top-down approach to find missed corners and remove false positives. ShortStraw consists of several steps. The first step is to resample the points of the stroke to be evenly spaced. The next step is to obtain the "straws". A straw for each resampled point p_i is computed as:

$$straw_i = |p_{i-W}, p_{i+W}| \quad (1)$$

Where W is a constant window set to 3, and the expression $| p_{i-W}, p_{i+W}|$ is the Euclidean distance between the points p_{i-W} and p_{i+W}. Once the straws have been calculated, the initial corner set consists of all the local minimums below a threshold based on the median straw value.

After obtaining the initial corner set, the top-down approach refines this set by means of a line test: two corners at indices a and b pass the line test if their Euclidean distance and their path distance (the sum of Euclidean distances between the resampled points) are relatively equal.

This test is applied first to find missed corners: if the line test between two consecutive corners fails, a new corner is added at the point with the minimum straw value between them. This process is repeated until no more corners are added. After, this test is applied to remove false corners: for each corner, if the line test between the two corners adjacent to it is positive, the central corner is removed. This process is repeated until no more corners are removed.

2.2 IStraw

IStraw [4] was presented in 2009 by Xiong and La Viola as a review of ShortStraw and included some improvements on it. These improvements were the addition of an extension for dealing with strokes that contain curves, the use of speed information (because users slow down on corners) to find corners, and other improvements such as dynamic thresholds.

The main improvement of IStraw is the curve test to remove false corners that appear in curves. The curve test (Fig. 1 left) is based on the different angles between a wrong corner (C_i) located on a curve, and two pairs of resampled points (*A-B* and *D-E*). The indices of *A*, *B*, *D* and *E* (determined empirically) are *i-shift*, *i+shift*, *i-(shift/3)* and *i+(shift/3)*, where $shift=min(15,C_i-C_{i-1},C_{i+1}-C_i)$, being C_{i-1} and C_{i+1} the previous and the following corners to C_i respectively. C_i is a correct corner if $(\beta-\alpha)<t_a$, where the threshold $t_a=10+800/(\alpha+35º)$ depends on the angle α (because $\beta-\alpha$ increases if α decreases) and it is determined empirically.

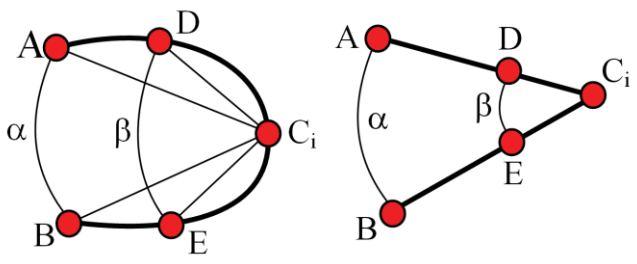


Figure 1. Difference between a false corner on a curve (left) and a correct corner between two straight segments (right)

2.3 Pu and Gur method

Pu and Gur [5] presented in 2009 their method that-uses some mathematical approach to the stroke to obtain its vertices. These are the main features:

- First, they do an approximation by using “radial basis functions” (RBFs).
- The vertices are located at the points necessary for the RBFs to fit the stroke. For this reason, they do not distinguish between corners and tangent vertices (a very important key). It is evident that neither straight lines nor curves are found and fitted to the stroke.
- Like ShortStraw and IStraw, performs an intensive post-process to refine the initial set of vertices. Even though, the false positive ratio is very high.

3 THE TCVD METHOD

As mentioned in previous section [28, 31], a common way for finding corners in strokes consists of looking for peaks of maximum curvature in absolute value, where curvature is calculated as the change of direction at each point. Instead of using the curvature, we use the radius (which is the inverse of the curvature). Although curvature has been widely used in recognition tasks, the radius gives us an advantage over curvature: its meaning is more intuitive than the curvature, and allows setting better the value of thresholds, that is, with radius we know better the meaning of the threshold. Specifically in our case two thresholds have been fixed, a smaller one for corner vertices and a larger for arcs or curves, as shown in Fig. 2.

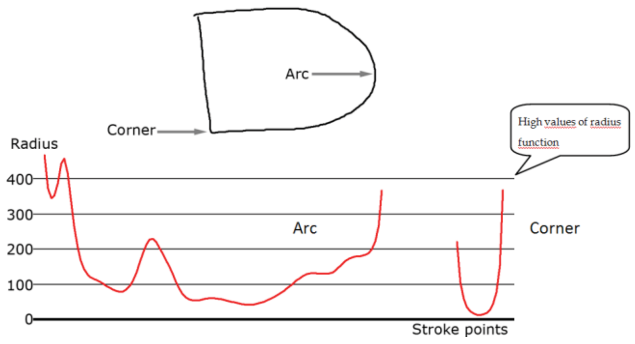
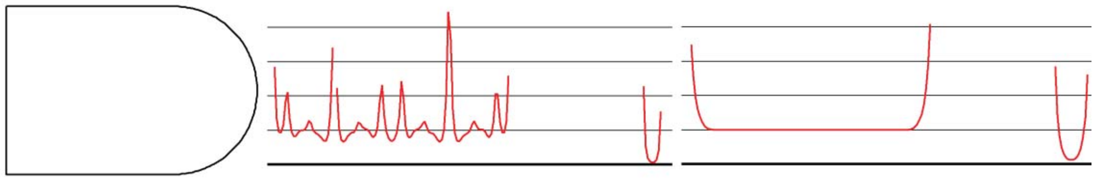


Figure 2. Radio values of a sample of a shape (high values of radio are not represented)

However, both the discrete radius and the curvature are not stable in hand-drawn sketches. Let's analyze the noise due to raster effects. Fig. 3 shows, from top to bottom: a) the same stroke in Fig. 2 with an arc and a corner vertex with coordinates obtained by arc and line analytic equa-tions; b) the discrete radius function obtained from the points of the stroke in integer coordinates even with some smoothing; and c) the discrete radius function with the same smoothing from the points of the stroke in real coordinates. As we can see, the corner vertex is a well defined peak of minimum radius in both cases, but the discrete radius function for the arc has a lot of noise due to aliasing raster effects (Fig. 3b).



a) b) c)

Figure 3. Raster effects in sketching: a) stroke with coordinates obtained from analytical form;

b) noise in the discrete radius function from points in integer coordinates from form above even with some smoothing; and c) discrete radius function without noise from points in real coordinates from analytical equations

To avoid noise due to aliasing raster effects and discontinuities in radius of curves (see Fig. 4), we obtain a piecewise parametric curve approximation of the stroke, and calculate the radius function from the mathematical expressions of the parametric curves in order to segment the stroke (get the entities in the stroke keeping the points of tangency between them). This approximation is very similar to that performed by Bein et al. [42], but in this case they fit the stroke to curves to describe best its shape in their 3D modeling system, and we use the approximation to obtain the radius more precisely.

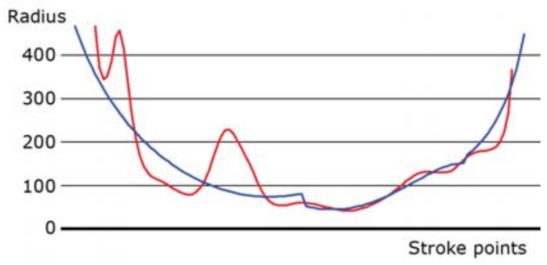


Figure 4. Difference between discrete radius obtained from differences with high smoothing (red), and analytic radius obtained from mathematical expressions of parametric curves (blue) for the arc of Fig. 2

On the other hand, is not recommendable to perform the approximation by means of parametric cubic curves before to obtain the corner vertices, since the major error in approximation is always for that kind of vertices (because we are obtaining smooth transitions for corners), distorting their radius values.

Previous to explaining in deep the method, the algorithm of the TCVD segmentation method is presented (Fig. 5). This algorithm has six differentiated parts (each inside a rectangle). On the right side of the figure appear graphical examples to illustrate the process carried out in each part of the algorithm.

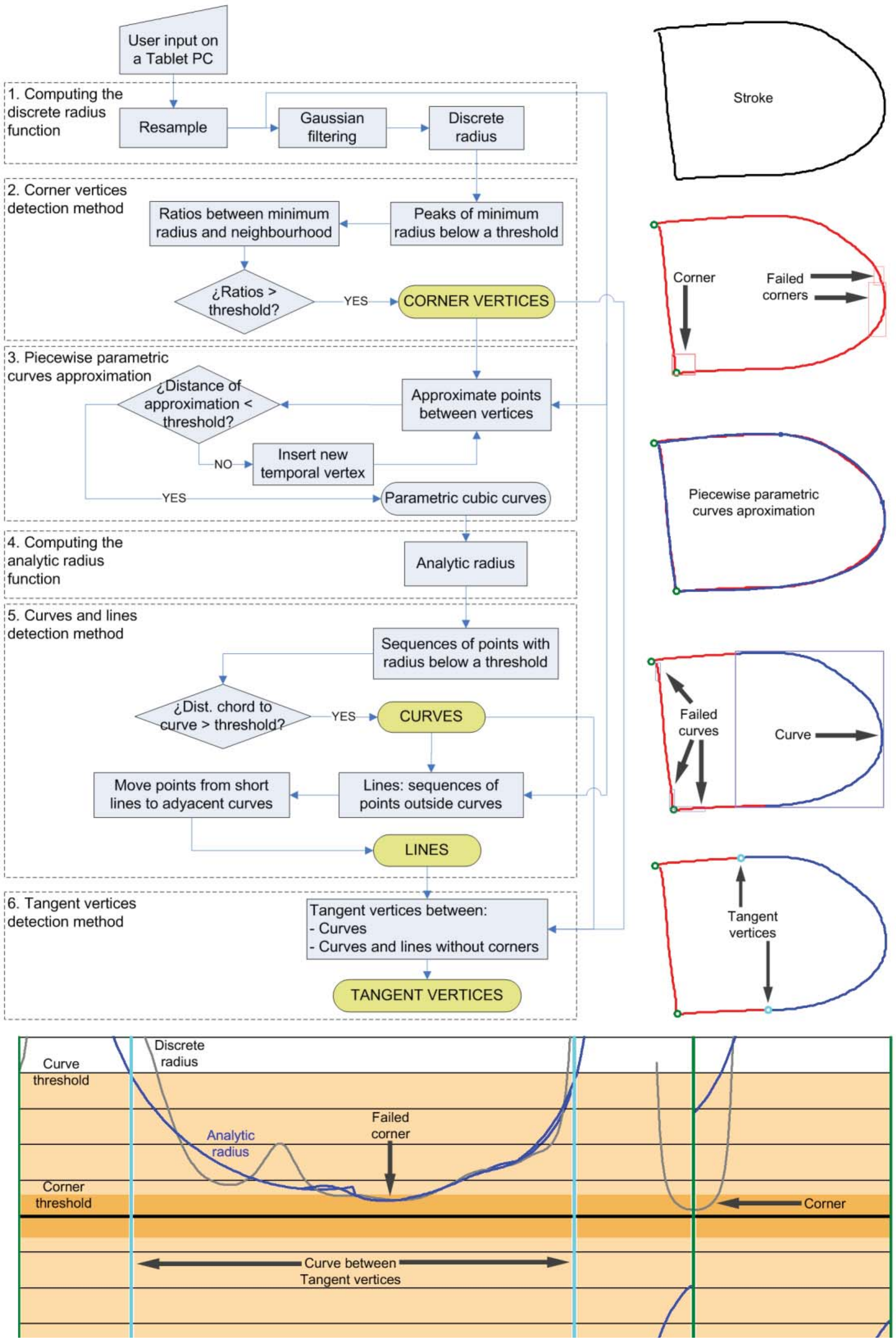

User input on a Tablet PC
1. Computing the discrete radius function
Resample
Gaussian filtering
Discrete radius
2. Corner vertices detection method
Ratios between minimum radius and neighbourhood
Peaks of minimum radius below a threshold
¿Ratios > threshold?
YES
CORNER VERTICES
3. Piecewise parametric curves approximation
¿Distance of approximation < threshold?
Approximate points between vertices
NO
Insert new temporal vertex
YES
Parametric cubic curves
4. Computing the analytic radius function
Analytic radius
5. Curves and lines detection method
Sequences of points with radius below a threshold
¿Dist. chord to curve > threshold?
YES
CURVES
Move points from short lines to adyacent curves
Lines: sequences of points outside curves
LINES
6. Tangent vertices detection method
Tangent vertices between:
- Curves
- Curves and lines without corners
TANGENT VERTICES
Stroke
Corner
Failed corners
Piecewise parametric curves aproximation
Failed curves
Curve
Tangent vertices
Curve threshold
Discrete radius
Analytic radius
Failed corner
Corner threshold
Corner
Curve between Tangent vertices

Figure 5. The flowchart algorithm of the TCVD method and radius functions

Next appears an exhaustive explanation of the steps of the TCVD segmentation method that includes the input, the output, a short description and the implementation of each step. Later, table 2 shows a compilation of the parameters (and its optimized values) for the TCVD.

3.1 Computing the discrete radius function

Input: the sketched stroke as a list of *m* points at different distances introduced by means of a graphical input device.

Output: consists of a vector of *n* evenly spaced points, which will replace the original stroke from now on, and a vector of *n* points with the radius calculated at each point, in an approximate way, by differences between neighbouring points.

Short description: first, the digitised stroke is resampled so that all points are evenly spaced and smoothed by a Gaussian filter to reduce the noise. After, the tangent vector at each point is calculated from differences between coordinates of its neighboring points, then the curvature at each point is obtained from differences of the tangent angle between neighboring points, and finally the radius at each point is calculated as the inverse of the curvature.

Implementation:

In order to compute the discrete radius function, some steps have been carried out.

Like ShortStraw and IStraw, we resample the *m* points of the stroke where *s* (INTERSPACING_DISTANCE) is the distance between resampled points. After resampling, the *n* resampled points of the stroke are evenly spaced:

$$\left.\begin{array}{l} x = x_0, x_1, \ldots, x_{n-1} \\ y = y_0, y_1, \ldots, y_{n-1} \end{array}\right\}, i \in [0, n-1] \quad (2)$$

The resampled points (x_i,y_i) are smoothed with a Gaussian filter (3) to reduce the effects of noise in the following calculations of direction, curvature and radius, where *wf* (FILTER_WINDOW) is the window used for the filter.

$$f_h = e^{-\frac{h^2}{2wf^2}}, h \in [-wf, wf] \quad (3)$$

The smoothed points (xf_i,yf_i) are obtained by discrete convolution of the resampled points with the Gaussian filter, dividing each point by the sum of filter values (4).

$$xf_i = \frac{\sum_{h=-wf}^{wf} f_h \cdot x_{i+h}}{\sum_{h=-wf}^{wf} f_h}, yf_i = \frac{\sum_{h=-wf}^{wf} f_h \cdot y_{i+h}}{\sum_{h=-wf}^{wf} f_h} \qquad (4)$$

The tangent at a point *(xf_i,yf_i)* is calculated from the differences of coordinates between the end points of a window, with *wt* (DIRECTION_WINDOW) size centered on it (5). The stroke direction α_i at a point *(xf_i,yf_i)* is the computed tangent angle in (6).

$$\left.\begin{array}{l} xf_i' = \dfrac{xf_{i+wt} - xf_{i-wt}}{2 \cdot wt \cdot s} \\ yf_i' = \dfrac{yf_{i+wt} - yf_{i-wt}}{2 \cdot wt \cdot s} \end{array}\right\} \qquad (5)$$

$$\alpha_i = \arctan\left(\frac{yf_i'}{xf_i'}\right) = \arctan\left(\frac{yf_{i+wt} - yf_{i-wt}}{xf_{i+wt} - xf_{i-wt}}\right) \qquad (6)$$

The tangent angle is in the range [-π,π] and can present discontinuities between consecutive values due to the cyclic properties of angles, so a correction of angle values is done using previous values:

- *While* $[(\alpha_i - \alpha_{i-1}) < (-\pi)]$ *do* $(\alpha_i \leftarrow \alpha_i + 2\cdot\pi)$
- *While* $[(\alpha_i - \alpha_{i-1}) > (\pi)]$ *do* $(\alpha_i \leftarrow \alpha_i - 2\cdot\pi)$

The curvature c_i at a point is calculated from the differences of direction angles between the end points of a window, with *wc* (CURVATURE_WINDOW) size centered on it (7). Finally, the radius r_i is the inverse of the curvature (8).

$$c_i = \alpha'_i = \frac{\alpha_{i+wc} - \alpha_{i-wc}}{2 \cdot wc \cdot s} \qquad (7)$$

$$r_i = \frac{1}{\alpha'_i} \qquad (8)$$

Figure 6 shows the discrete direction, curvature and radius function of shape in Fig. 2.

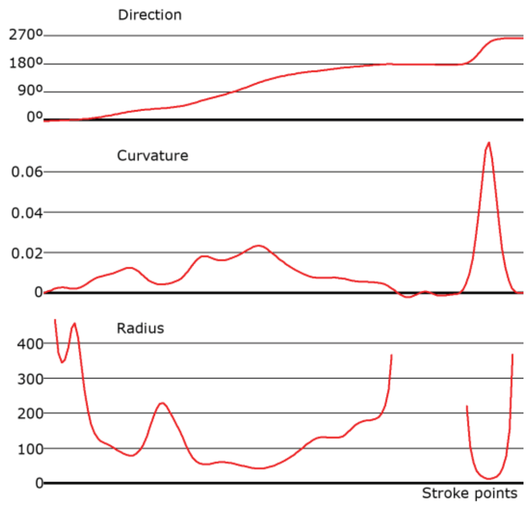


Figure 6. From top to bottom: stroke direction (corrected to avoid discontinuities), curvature and radius discrete function from shape in Fig. 2

3.2 Detection of corner vertices

Input: a vector with a stroke of *n* evenly spaced points and a vector with the radius calculated at such points.

Output: a vector of *n* points that will contain all the corner vertices, considering the first and the last point of the stroke corners. This vector is called vector of entities and will contain the type of entity for each point (straight line, curve, corner vertex and tangent vertex) at the end of the process of the TCVD algorithm.

Short description: the corner vertices are located at points with local minima of the radius, and with a radius sufficiently smaller than the points of its environment.

Implementation:

The corners are located at peaks of minimum radius in absolute value, that is, in points with the maximum curvature or maximum variation in stroke direction. Those local minimums will be (in absolute value) below a parameter set to a maximum value (MAX_CORNER_RADIUS). These points are corners if the minimum radius is much smaller than the radius located on its sides. We obtain (only for radius with the same sign) the average radius in the NEIGHBORING_WINDOW previous points, and the average radius in the same subsequent points. The ratio of both average radius and the minimum radius must be greater than a specific value (MIN_RADIUS_RATIO).

Besides, the initial and end points of the stroke are always considered corners.

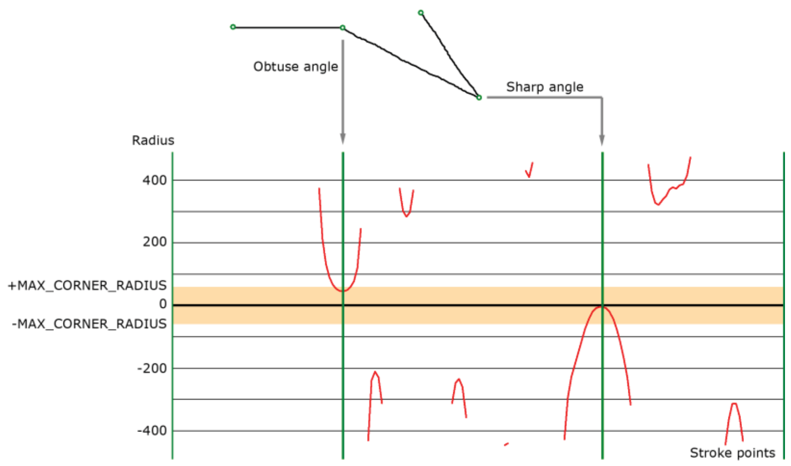


Figure 7. Corner vertices drawn in green: initial stroke point, obtuse angle, sharp angle and end stroke point. Positive radio values for right turns and negative for left turns

3.3 Piecewise parametric curves approximation

Input: the vector of evenly spaced points and the vector of entities just containing the corner vertices.

Output: several piece-wise cubic curves (the number of corner vertices minus one) that approximate, each one, the points between pairs of corner vertices.

Short description: the resampled points between pairs of corners are approximated by means of piece-wise cubic curves until the distance from every approximated point to the resampled point does not exceed a threshold. If the distance is greater, the sequence of points is halved and the process is subsequently applied to the two sides, forcing two curves to have the same tangent at the common point (the central point when the previous sequence is divided).

Implementation:

The approximation of resampled points $pr_j=(xr_j,yr_j)$, by means of parametric cubic curves has as a main goal the more accurate calculation of radius values in order to obtain the stroke curves.

A parametric cubic curve consists of two polynomial equations of 3rd degree, for x and y coordinates each. The two polynomial expressions have 4 coefficients each (in total 8 degrees of freedom) and depend on a parameter *t*, whose value is set to 0 at the beginning of the curve and to 1 at its end. The expressions of a parametric cubic curve and its first derivative are the following:

$$\left.\begin{array}{l} x(t)=a_x+b_x\cdot t+c_x\cdot t^2+d_x\cdot t^3 \\ y(t)=a_y+b_y\cdot t+c_y\cdot t^2+d_y\cdot t^3 \end{array}\right\} \quad \left.\begin{array}{l} x'(t)=b_x+2\cdot c_x\cdot t+3\cdot d_x\cdot t^2 \\ y'(t)=b_y+2\cdot c_y\cdot t+3\cdot d_y\cdot t^2 \end{array}\right\},\ t\in[0,1] \qquad (9)$$

The curve must approximate as much as possible to each of the resampled points, that is, it is expected that for each point $pr_j=(xr_j,yr_j)$, exists a t_j to accomplish:

$$\left.\begin{array}{l} a_x+b_x\cdot t_j+c_x\cdot t_j^2+d_x\cdot t_j^3=xr_j \\ a_y+b_y\cdot t_j+c_y\cdot t_j^2+d_y\cdot t_j^3=yr_j \end{array}\right\} \qquad (10)$$

To avoid errors in corner vertices (as said in 4), we approximate resampled stroke points to parametric cubic curves between pairs of corner vertices. These curves have the constraints of passing through initial and final corner vertices and approximating the points between them. In the case that the approximated curve overcomes the parameter of maximum distance (MAX_DISTANCE) to any of the resampled stroke points, the sequence of points to approximate is then half divided and two constraints for the two approximated curves are added: 1) both curves have to pass through the middle point; and 2) there must be first order continuity (equal direction of tangent in the middle point) in order to accomplish soft transition.

As this process can be applied several times, we can obtain four different ways to approximate a sequence of points by mean a parametric cubic curve (see Table 1), depending on the constraints it must accomplish.

Table 1. Constraints to be accomplished by curves

The curve passes through		The curve has the tangent at	
Initial point	Final point	Initial point	Final point
X	X		
X	X	X	
X	X		X
X	X	X	X

When a sequence of points to approximate is divided, the tangent in the midpoint (first derivative) is calculated from the discrete stroke direction in that point:

$$\left.\begin{array}{l} xr_j'=k\cdot\cos(\alpha_j) \\ yr_j'=k\cdot\sin(\alpha_j) \end{array}\right\} \qquad (11)$$

Where *k* is the module of the tangent vector, which is leaved free in order to the least square system obtains the value that bests suits the resampled points, for calculated value of tangent.

In the appendix, the approximation process is described in detail.

As we can see in Fig. 8 (right), with the valued used for MAX_DISTANCE, the difference between the resampled points and the approximated points is worthless, and also noise has been removed.

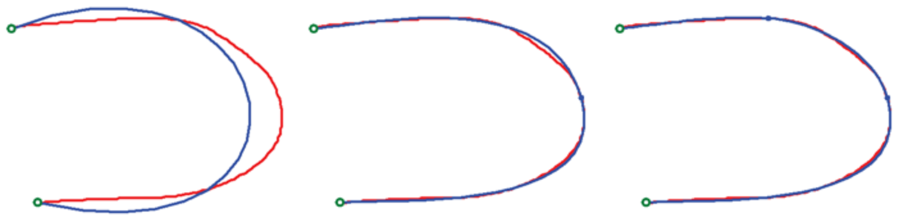


Figure 8. Resampled points of a stroke (red) and piece-wise parametric cubic curves approximation (blue). From left to right, with 1, 2 and 3 parametric cubic curves

3.4 Computing the analytic radius function

Input: the piece-wise cubic curves.

Output: a vector of *n* points with the radius, for each point of the stroke, calculated by means of derivative of the piece-wise cubic curves.

Short description: the tangent vector at each point is calculated from derivative of piece-wise cubic curves, then the curvature at each point is obtained from derivative of the tangent angle, and finally the radius at each point is calculated as the inverse of the curvature.

Implementation:

The stroke direction α(*t*) is the angle of the stroke tangent, where the tangent was obtained by means the derivative in each point of the stroke with the corresponding parametric cubic curve with the parameter t_j:

$$\alpha(t) = \arctan\left(\frac{y'(t)}{x'(t)}\right), \quad \left.\begin{aligned} x'(t) &= b_x + 2\cdot c_x \cdot t + 3\cdot d_x \cdot t^2 \\ y'(t) &= b_y + 2\cdot c_y \cdot t + 3\cdot d_y \cdot t^2 \end{aligned}\right\} \qquad (12)$$

Similar to the discrete case, the angle values of the *α(t)* function are corrected to avoid discontinuities.

The stroke curvature *c(t)* is the variation of the direction in each stroke point, when the angle changes greatly, the higher is the curvature value. The curvature is the quotient of the derivative of the tangent by the corresponding parametric cubic curve length. *α'(t)* is the derivative of the tangent angle to the stroke, obtained by means the derivative in each point of the stroke with the parametric curve with the corresponding parameter t_j.

$$c(t)=\frac{\alpha'(t)}{(k-1)\cdot s}=\frac{\left[\arctan\left(\frac{y'(t)}{x'(t)}\right)\right]'}{(k-1)\cdot s}=\frac{\frac{\left[\frac{y'(t)}{x'(t)}\right]'}{1+\left[\frac{y'(t)}{x'(t)}\right]^2}}{(k-1)\cdot s}=\frac{\frac{\frac{y''(t)\cdot x'(t)-y'(t)\cdot x''(t)}{[x'(t)]^2}}{1+\left[\frac{y'(t)}{x'(t)}\right]^2}}{(k-1)\cdot s} \tag{13}$$

Where *(k-1)·s* is the length of the parametric cubic curve, since *s* is the distance between resampled points and *k* is the number of resampled stroke points the curve approximates. So, the first and second derivatives of the parametric curve are:

$$\left.\begin{aligned}x'(t)&=b_x+2\cdot c_x\cdot t+3\cdot d_x\cdot t^2\\ y'(t)&=b_y+2\cdot c_y\cdot t+3\cdot d_y\cdot t^2\end{aligned}\right\}\quad\left.\begin{aligned}x''(t)&=2\cdot c_x+6\cdot d_x\cdot t\\ y''(t)&=2\cdot c_y+6\cdot d_y\cdot t\end{aligned}\right\} \tag{14}$$

The stroke radius *r(t)* is the inverse of the curvature. In order to get rid of divisions by zero, when the absolute value of the curve is lower than a minimal, the radius is set to a high value sharing the curvature sign.

$$r(t)=\frac{1}{c(t)} \tag{15}$$

Fig. 9 shows the difference between discrete and analytic radius, and also the difference between the centers of rotation for each stroke point. The centers of rotation are obtained perpendicular to the direction of the stroke at a distance equal to the radius. As we can see, with analytic functions, the radius values are smoother, and this can be seen clearer in the position of centers.

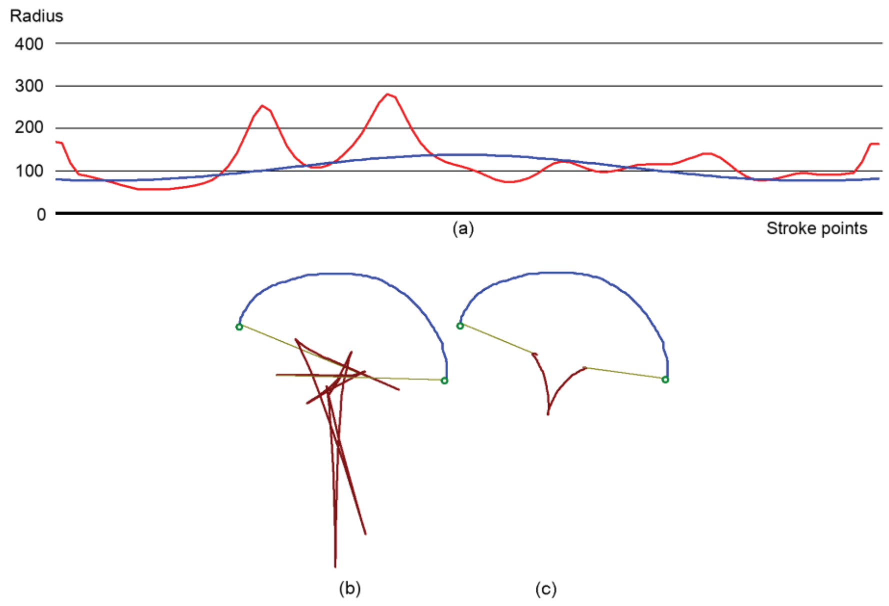


Figure 9. Radius and centers of rotation for the same example in Fig. 8: a) discrete (red) and analytic (blue) radius; b) centers of rotation (brown) for each point of the stroke calculated by means of discrete direction and radius; c) same as b) calculated by means of analytic direction and radius

3. 5 Detection of lines and curves

Input: the vector with the radius at each point, calculated from the piece-wise cubic curves, and the vector of entities just containing the corner vertices.

Output: the vector of entities that contains for each point whether it belongs to a stright line or a curve depending on its radio.

Short description: a point lies on a curve if the radius at that point is less than a threshold, otherwise the point belongs to a straight line. Therefore, a sequence of consecutive curve points is definitely a curve if the distance between the points and the straight line from first sequence point to last one is greater than a threshold.

Implementation:

The curves are located in stretches of stroke points whose radius values have the same sign and their absolute value is lower than a specific value (MAX_CURVE_RADIUS). Curves are considered as circle arcs for calculations, so the descriptors/ features for curves are the following:

- Curve: sequence of consecutive stroke points whose radius values (obtained from equa-tions of parametric cubic curves) are lower than MAX_CURVE_RADIUS.
- Radius: the median of the radius of curve points.
- Length: number of curve points multiplied by the interspacing distance.
- Angle: Length / Radius.
- Distance from chord to arc (see Fig. 10): Radius·[1 – cos (Angle / 2)].

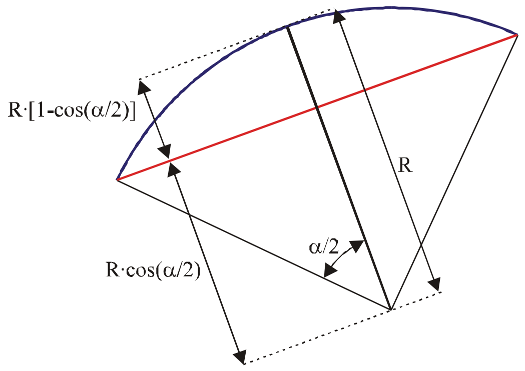


Figure 10. Distance from chord (red) to arc (blue)

The stretches candidate to be curves will be in the case their distance from chord to arc was higher than a specific value (MIN_DIST_CA). The stretches that are not curves are considered as straight lines.

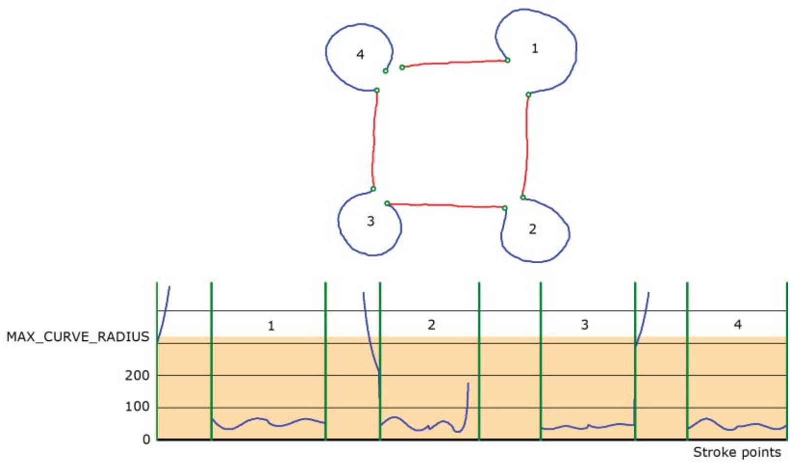


Figure 11. Shape with curves (blue) and straight lines (red) separated by corner vertices. Curves are numbered from 1 to 4. The values of radius for curves are below MAX_CURVE_RADIUS

3.6 Detection of tangent vertices

Input: the vectors with radius and entities at each point of the stroke.

Output: the vector of entities also containing the tangent vertices.

Short description: the tangent vertices are located at points of transition from straight lines to curves (and vice versa), and at points of transition between curves of radius with different sign, if corner vertices are not previously placed in such transitions.

Implementation:

After obtaining the corners, curves and straight lines, the tangent vertices can be located in transitions (without corner vertices) between straight lines and curves, or between curves. But first to location of tangent vertices, a brief consideration is done: the straight lines with a length lower than a value (MIN_LINE_LENGTH) are converted to curves if they are besides a curve and no corner vertex is in between. In the case that a line stretch has curves on both sides, it is equally shared out between both curves.

Then the tangent vertices are located at (see Fig. 12):

- Changes from a lined stretch to a curved stretch, and from a curved stretch to lined stretch, with no corner vertex in between.
- Changes from a curved stretch to another curved stretch with no corner vertex in between. In this case the tangent vertex is located in an inflexion point (that is, in a point where the sign of curvature and radius function changes) and it is usual to detect a small straight line between the curves. As mentioned before, if this line is shorter than the mentioned threshold, it is added to the curves (half to each one) placing the tangent vertex in the middle.

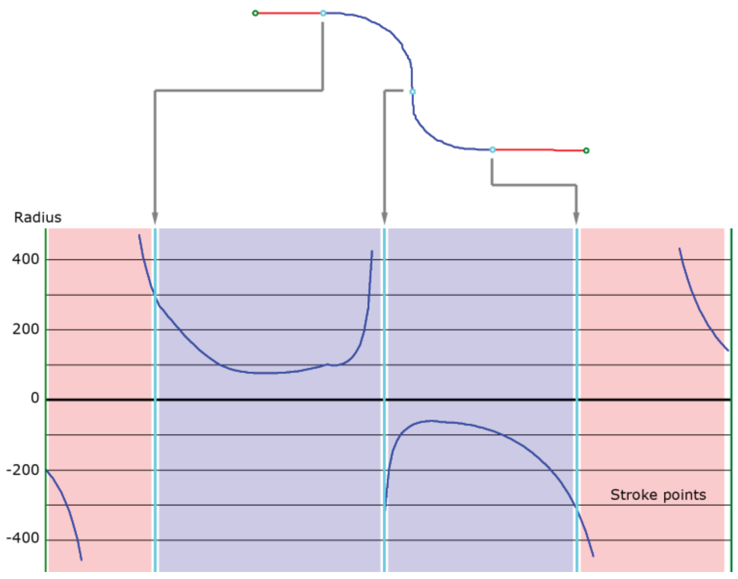


Figure 12.Tangent vertices (drawn in cyan) between curves and straight lines, and between two curves. The background of radius function is blue for curves and red for straight lines

4 EXPERIMENTAL WORK

In order to evaluate our method we have used a data set of 17 different shapes, which consists of the 11 polyline strokes in [3] (Fig. 13), and 6 curve strokes (see Fig. 14) which contains features of engineering drawings such as tangent vertices, that appear in 3 of them. All of them are open shapes, and the first and last points of the stroke are always considered as corner vertices. Both Fig. 3 and Fig. 4 show the out-lined models of the different shapes used.

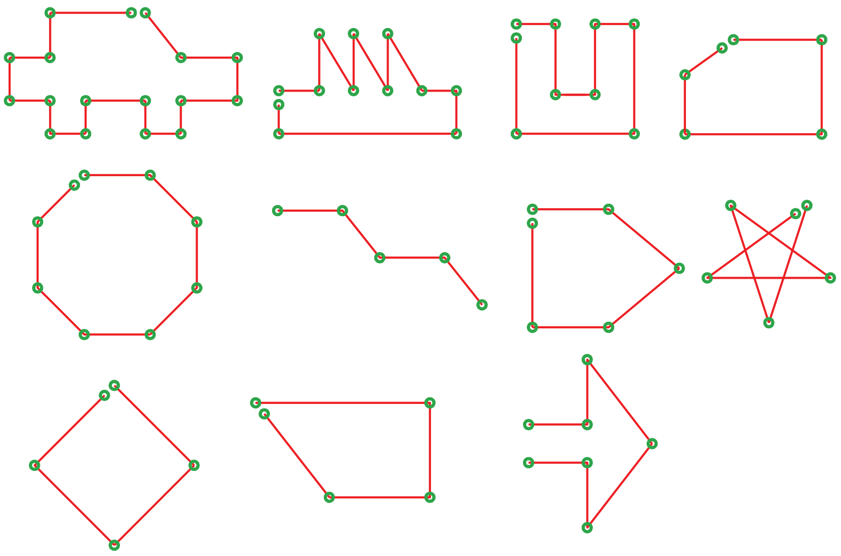


Figure 13. Strokes with straight lines (drawn in red) and corner vertices (drawn in green)

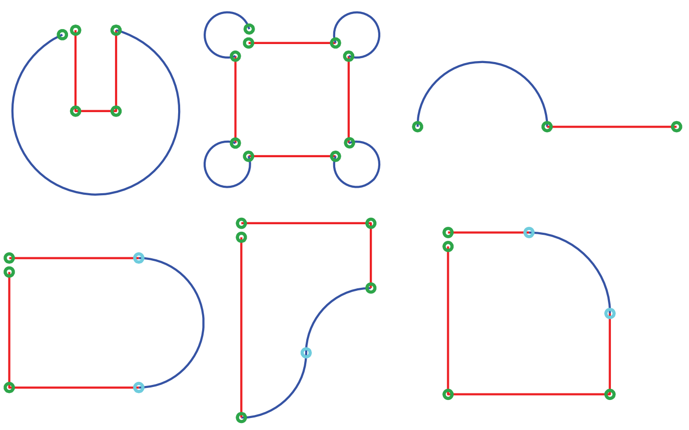


Figure 14. Strokes with straight lines (drawn in red), curves (drawn in blue), corner vertices (drawn in green) and tangent vertices (drawn in blue cyan)

We collected data from 8 different users, and each user drew 6 times each shape (816 strokes). Users were not given any indication about the accuracy, so each one drew the strokes on their way. The sketched shapes collected are available in the following address:

http:/ / personales.upv.es/ maalbor/ Files/ Data-set.rar

The parameters used in TCVD were optimised by means of Simulated Annealing algorithm in order to achieve best results, process that is explained in detail in a previous work [43]. Simulated Annealing [44] is a well known optimization method, which allows us tuning the parameters to improve segmentation results (especially when the parameters depend on each other). The tuning of parameters has been formulated as an optimization problem where the function cost is expressed as the number of errors in the segmentation of the training set. The result of several processes carried out by simulated annealing optimization shows the need for the Gaussian filter to remove noise, although it adversely affects the corners (see figure 3). Should also be noted that the maximum radius of the curves is directly dependent on the size of the drawing area (800x500 pixels in our test application).

For the determination of the optimal parameters, a training data set with 136 strokes (8 per shape) was used, and the remaining 680 were used as test data set. The training is an off line process, so the temporal cost of the convergence of the algorithm does not affect the temporal cost of the TCVD algorithm. This temporal cost has been of 10 ms per shape using a computer with an Intel Core 2 Duo E8400 3.00GHz and Windows XP, where the 86 % is for the step 3 (piecewise parametric curves approximation), the 11 % is for step 1 (computing the discrete radius function) and the remaining 3 % is for the rest of the algorithm.

The optimised parameters are shown in table 2, and these values are directly set in the TCVD algorithm:

Table 2. Parameters of the TCVD algorithm

TCVD Parameters	**Description**	**Value from SA**
INTERSPACING_DISTANCE	Interspacing distance between resampled points	2
FILTER_WINDOW	Window size for Gaussian filter	10
DIRECTION_WINDOW	Window size for stroke direction calculation	8
CURVATURE_WINDOW	Window size for stroke curvature calculation	2
MAX_CORNER_RADIUS	Maximum radius for corners	60
NEIGHBOURING_WINDOW	Window size for corners neighbouring	10
MIN_RADIUS_RATIO	Minimum ratio between radius of corners and its neighbouring	1.4
MAX_DISTANCE	Maximum distance between resampled points and parametric curve approximation	5.0
MAX_CURVE_RADIUS	Maximum radius for curves	400
MIN_CURVE_DIST_CA	Minimum distance from chord to arc (curve)	10.6
MIN_LINE_LENGTH	Minimum length of a straight line	45

As stated before, as almost all methods in literature do not find tangent vertices (smooth transitions), in order to evaluate our method we have chosen the three most relevant and recent methods in this subject, where properties of two of them are the simplicity and the high accuracy in the corner vertices detection in sketches with curves (ShortStraw and IStraw methods), and the third of them relative to tangent vertices (Pu and Gur method).

Then, for comparison we tested an implementation of ShortStraw and IStraw, and also compare results with Pu and Gur method. As the implementation of IStraw presented several drawbacks with dealing to our test data set, we made some corrections to achieve best results, being the most important one dispensing with time info because the number of errors increases when the initial corner set is made up of both, straws and time info.

The results can be found in Table 3 and 4 (only for the 440 strokes without curves) and Table 5, 6 and 7 (for all the 680 strokes in the test data set). These results are expressed in the same measures than ShortStraw and IStraw: “Correct Corners Accuracy” and “All-or-Nothing Accuracy”, the first is equal to the number of correct vertices found divided by the total number of vertices, and the second is equal to the number of correctly segmented strokes (without false positives or false negatives) divided by the total number of strokes. The “All-or-Nothing Accuracy” measure is the most significant because it takes false corners into account. In order to test separately the importance of false positive, it has been added the “False Positive Rate” which is the number of false positives divided by the total number of vertices.

Table 3. Accuracy results for 440 polyline strokes (without curves or tangent vertices)

	ShortStraw	**IStraw**	**TCVD**
False Positives	31	3	0
False Negatives	23	62	3
Correct Corners	3457	3418	3477
Total Corners	3480	3480	3480
Correct Corners Acc.	99.3%	98.2%	99.9%
False Positive Rate	0.9%	0.1%	0.0%

Table 4. All-or-nothing accuracy results for 440 polyline strokes (without curves or tangent vertices)

	ShortStraw	**IStraw**	**TCVD**
All-or-Nothing Acc.	89.8%	91.4%	99.3%

Table 5. Accuracy results for 680 polyline and curve strokes (corner vertices)

	ShortStraw	**IStraw**	**TCVD**
False Positives	1525	400	12
False Negatives	24	125	6
Correct Corners	4616	4515	4634
Total Corners	4640	4640	4640
Correct Corners Acc.	99.5%	97.3%	99.9%
False Positive Rate	32.9%	8.6%	0.3%

Table 6. Accuracy results for 680 polyline and curve strokes (tangent vertices)

	ShortStraw	**IStraw**	**TCVD**
False Positives	---	---	4
False Negatives	170	180	16
Correct Tangent Vertices	30	20	184
Total Tangent Vertices	200	200	200
Correct Tangent Vertices Acc.	15.0%	10.0%	92.0%
False Positive Rate	---	---	2%

Table 7. All-or-nothing accuracy results for 680 polyline and curve strokes

	ShortStraw	**IStraw**	**TCVD**
All-or-Nothing Acc.	58.1%	63.5%	96.6%

Finding corner vertices on polyline strokes, TCVD reaches higher all-or-nothing accuracy

(99%) than both ShortStraw and IStraw, which is already a good result (90%). ShortStraw is not ready for dealing with curves and obviously the accuracy decreases when the strokes contain curves (many false corners appear). The IStraw curve test removes many of these false corners but also some correct corners, what makes the final results are not much better than ShortStraw, and whereas accuracy remains high on TCVD (96%) largely because the number of false positives is very low.

TCVD finds most of the tangent vertices (92%) including few false tangent vertices. Neither ShortStraw nor IStraw look for tangent vertices, although they find some false corners near tangent points, because the tolerance of position must be high.

Relative to Pu and Gur method, they use complex shapes that are drawn either by hand or with CAD applications. Only a very small set of 30 hand-drawn figures is used for their tests. It must be noted that many of them are not able to be drawn at once, that is, there are complex shapes that normally are drawn by a designer in several steps (or strokes), raising the pen when it is needed and drawing again to get on with it. Moreover, our operating mode is on line, that is, the recognition is performed while drawing, and not off line (like Pu and Gur that work with scanned complete shapes), and therefore the comparison is relative.

Table 8 shows the results of this relative comparison. As Pu and Gur do not use the “All-or-Nothing Accuracy”, these results are expressed in: “Correct Corners Accuracy” (number of correct vertices found divided by the total number of vertices) and “False Positive Rate” (number of false positives divided by the total number of vertices).

Table 8. Percentage of correct and wrong vertices (corner and tangent altogether)

	Pu and Gur (CAD)	**Pu and Gur (Hand-drawn)**	**TCVD**
Correct Vertices Acc.	99.2%	97.8%	99.5%
False Positive Rate	24.5%	24.5%	0.3%

According to the values in the Table 8, the Correct Vertices Accuracy is very similar, but TCVD has a much better False Positive Rate. The high number of false positives has a negative effect in All-or-Nothing Accuracy, what makes the method powerless and causes a stressful effect on the user. In addition, Pu and Gur do not distinguish between corner and tangent vertices (what is an important point), because they do not distinguish between straight lines and curves. Besides it is odd that the False Positive Rate is the same for CAD drawings and hand drawings. Finally, note that like ShortStraw and IStraw, Pu and Gur method performs an intensive post-process.

On the other hand, analyzing TCVD, we can see that many missegmentations are due to the users were not given any indication about the accuracy in drawing the shape, therefore some of the strokes have poor quality and it is easy to confuse corner vertices with curves of small radius (Fig. 15 a-row), and straight lines with curves of large radius (Fig. 15 b-row). For shapes in segmentation column the straight lines are in red and curves in blue colour.

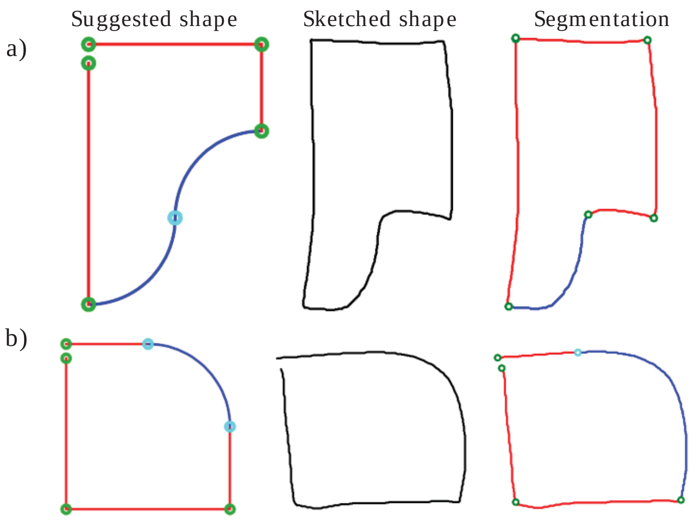


Figure 15. Poorly segmented-drawn strokes

The limitation of this method remains dealing with the following aspects:

- The size of the straight lines and curves that contains the stroke should be sufficient. In other words, strokes with small parts can be a problem.
- Curves with very large radius may be confused with straight lines. As mentioned before, the maximum radius of the curves depends on the size of the drawing area.
- Like other methods, it should be advisable for the user to draw thinking about what he/ she makes: stopping at corners (to change the direction and to avoid confusion with curves of small radius), not stopping at tangent vertices (to avoid changing the direction abruptly), and drawing straight lines and curves where appropriate (for example, in Fig. 15 b-row we can see the suggested shape on the left, the drawn shape on the center -

where a curve has been extended to include a line- and on the right the segmentation with a largest curve instead of a curve followed by a straight line).

5. CONCLUSIONS

TCVD is an important improvement in the field of free-hand sketches recognition, being the main contribution of this method the detection of tangent vertices in strokes. The accuracy of TCVD obtaining corner vertices is higher mainly because it has very few false positives, but also TCVD is able to find curves and straight lines, which allows obtaining tangent vertices between curves and between curves and straight lines, even with very few false positives.

As Fig. 16 shows, the conclusions of this method regarding to radius calculations and vertices detection are mainly the following:

- The approximation of the stroke by parametric cubic curves allows obtaining analytically the radius, eliminating most of the noise and keeping the shape of the stroke. The curvature radius stabilize and consequently so do the curvature centers (Fig. 16a,b,c,d left).
- The lack of continuity in the radius of parametric curve is due because in order to maintain the enough degrees of freedom for the curve to fit the stroke, just continuity of direction is fixed, so no curvature continuity is availed (Fig. 16a left).
- With respect other finding corners, which perform an intensive post-process, TCVD focuses on obtaining a good initial corner set by means of the radius function with no longer post-process.
- TCVD allows detecting the change in the radius of hand-draw arcs. If the arc is tangent to straight lines, the radius is very high next to the straight line (which has an infinity radius) and decreases as it approaches the center of the arc (see the centers position in Fig. 16c right). On the other hand, for isolated arcs, the radius increases as it approaches the center of the arc (see the centers position in Fig. 16d right).

Radius function	Sample of sketch with its original radius values	Radius Calculation

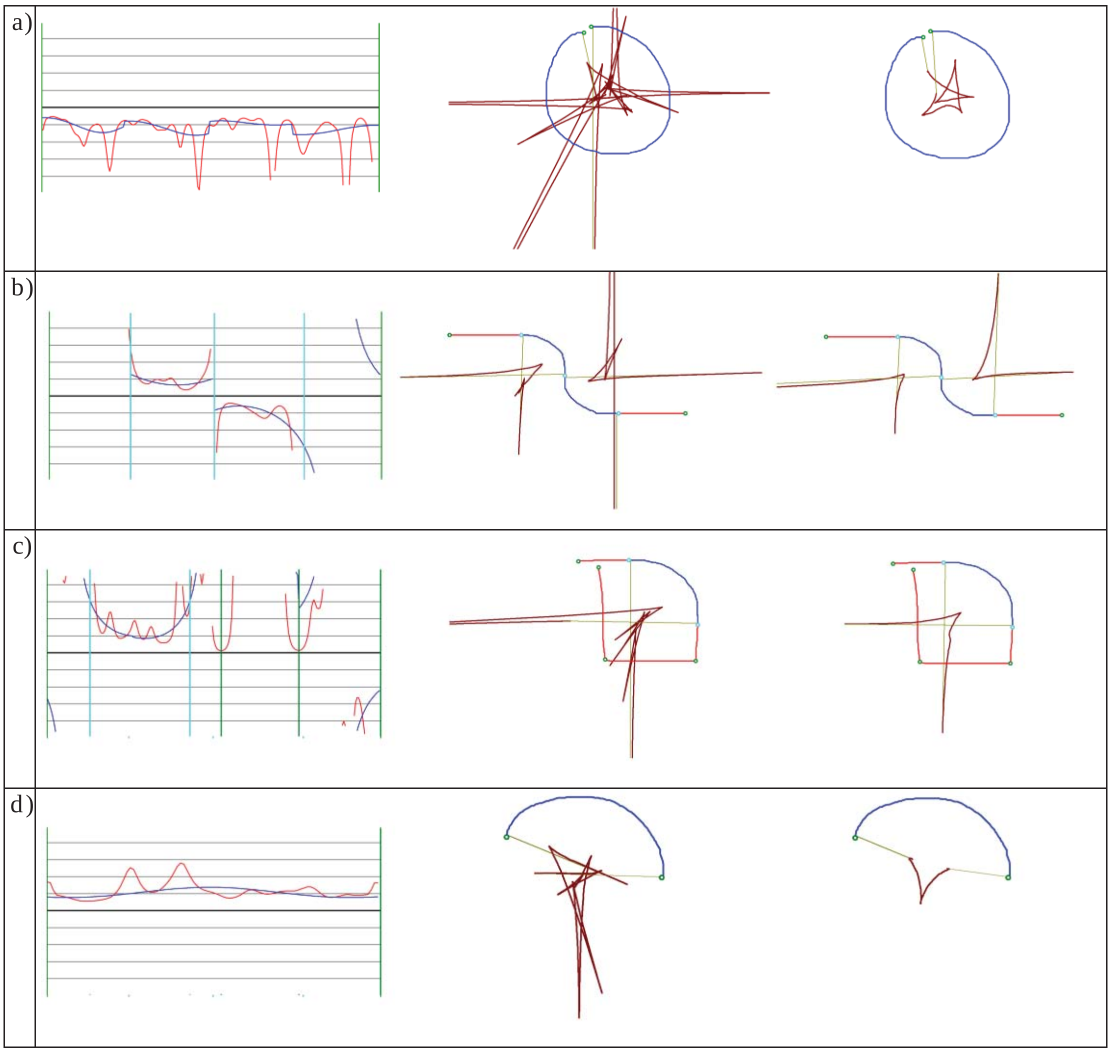


Figure 16. Left column: radius function from original sketched points (red) and from approximated parametric cubic curves (blue); Central column: radius from original sketched points represented (brown); Right column: radius from approximated parametric cubic curves represented (brown), of different samples of sketches a), b), c) and d) respectively

TCVD also proves that sketch recognition can get a lot of good from the approximation to parametric cubic curves, since most of tangent vertices not found by other methods in literature, can be found with a high accuracy, and consequently, the intent design of tangency can be captured in order to out-line further the sketch into a parametric CAD application.

6. FUTURE WORK

The main improvement is to perform the necessary extensions to deal with closed shapes, detecting when the first and last points of a stroke are very close, and considering that there is continuity between them.

A possible improvement could be to process multistrokes in order to have continuous shapes if they match some conditions, so further extrusions can generate intended 3D models. But at this stage the role of TCVD is just to segment single strokes. The multistroke processing is a task that could be performed in higher levels by using TCVD.

Appendix. Approximation by means of parametric cubic curves using least squares

As mentioned in section 3.3, the curve must approximate as much as possible to each of the resampled points, that is, it is expected that for each point $pr_j=(xr_j,yr_j)$, exists a t_j to accomplish (10).

Repeating the previous expressions for x and y coordinates of every resampled point, two linear equation systems are made, one for the x and other for the y coordinate. Each system has 4 unknown variables (the coefficients a, b, c and d) [45]. It is usual that the number of points to approximate is more than 4, so the systems are over-constrained and to solve them for the better approximation solution to the m resampled points it is necessary to use the minimum least squares method.

$$\left.\begin{array}{c} a_x + b_x \cdot t_0 + c_x \cdot t_0^2 + d_x \cdot t_0^3 = xr_0 \\ a_x + b_x \cdot t_1 + c_x \cdot t_1^2 + d_x \cdot t_1^3 = xr_1 \\ \ldots \\ a_x + b_x \cdot t_{m-1} + c_x \cdot t_{m-1}^2 + d_x \cdot t_{m-1}^3 = xr_{m-1} \end{array}\right\} \tag{16}$$

In a matrix form the expressions remains as following:

$$\begin{pmatrix} 1 & t_0 & t_0^2 & t_0^3 \\ 1 & t_1 & t_1^2 & t_1^3 \\ \ldots & \ldots & \ldots & \ldots \\ 1 & t_{m-1} & t_{m-1}^2 & t_{m-1}^3 \end{pmatrix} \begin{pmatrix} a_x \\ b_x \\ c_x \\ d_x \end{pmatrix} = \begin{pmatrix} xr_0 \\ xr_1 \\ \ldots \\ xr_{m-1} \end{pmatrix} \tag{17}$$

The values of the parameter t_j for every resample point pr_j are unknown and are calculated on an approximated way supposing that are proportional to the distances between resampled points:

- Assign to the first resample point (pr_0) a distance: $d_0 = 0$
- Assign to each other points (pr_j) the Euclidean distance to its previous point (pr_{j-1}): $d_j = \| pr_j - pr_{j-1} \|$
- Obtain the accumulate distances (da_j) of each pr_j to pr_0: $da_j = d_0 + d_1 + \ldots + d_j$

- Obtain the parameter values proportionally to the distances: $t_j = (da_j / da_{m-1})$

All the previous restrictions subtract freedom degrees to the parametric curves, that is, reduce the number of parameters and in consequence, the number of unknown variables in the equation systems. The restrictions are:

- The curve passes through the initial point (2 degrees of freedom are taken away, but we can remove from the least squares system the equation of approximation to the initial point):

$$t = t_0 = 0 \rightarrow \begin{cases} a_x + b_x \cdot 0 + c_x \cdot 0^2 + d_x \cdot 0^3 = xr_0 & \rightarrow \quad a_x = xr_0 \\ a_y + b_y \cdot 0 + c_y \cdot 0^2 + d_y \cdot 0^3 = yr_0 & \rightarrow \quad a_y = yr_0 \end{cases} \tag{18}$$

- The curve passes through the final point (2 degrees of freedom are taken away, but we can remove from the least squares system the equation of approximation to the final point):

$$t = t_{m-1} = 1 \rightarrow \begin{cases} a_x + b_x \cdot 1 + c_x \cdot 1^2 + d_x \cdot 1^3 = xr_{m-1} & \rightarrow \quad a_x + b_x + c_x + d_x = xr_{m-1} \\ a_y + b_y \cdot 1 + c_y \cdot 1^2 + d_y \cdot 1^3 = yr_{m-1} & \rightarrow \quad a_y + b_y + c_y + d_y = yr_{m-1} \end{cases} \tag{19}$$

- The curve has the specified tangent (first derivative) in the initial point (xr_0', yr_0'). This constraint affects the direction of the tangent, but not to its module, so just one degree of freedom is removed:

$$t = t_0 = 0 \rightarrow \begin{cases} b_x + 2 \cdot c_x \cdot 0 + 3 \cdot d_x \cdot 0^2 = k_0 \cdot xr_0' & \rightarrow \quad \mathrm{b}_x = k_0 \cdot xr_0' \\ b_y + 2 \cdot c_y \cdot 0 + 3 \cdot d_y \cdot 0^2 = k_0 \cdot yr_0' & \rightarrow \quad \mathrm{b}_y = k_0 \cdot yr_0' \end{cases} \tag{20}$$

- The curve has the specified tangent (first derivative) in the final point (xr_{m-1}', yr_{m-1}'). This constraint affects the direction of the tangent, but not to its module, so just one degree of freedom is removed:

$$t = t_{m-1} = 1 \rightarrow \begin{cases} b_x + 2 \cdot c_x \cdot 1 + 3 \cdot d_x \cdot 1^2 = k_{m-1} \cdot xr_{m-1}' & \rightarrow \quad \mathrm{b}_x + 2 \cdot c_x + 3 \cdot d_x = k_{m-1} \cdot xr_{m-1}' \\ b_y + 2 \cdot c_y \cdot 1 + 3 \cdot d_y \cdot 1^2 = k_{m-1} \cdot yr_{m-1}' & \rightarrow \quad \mathrm{b}_y + 2 \cdot c_y + 3 \cdot d_y = k_{m-1} \cdot yr_{m-1}' \end{cases} \tag{21}$$

Depending on the number of restrictions to apply, the linear equations will be different and will have more or less degrees of: from 4 (just with restrictions of passing through initial and final points) down to 2 (with the matching points restrictions and equal directions of tangents in initial and final points).

Besides, equations for curves without initial and final tangent, just with initial tangent, just with final tangent, and with initial and final tangent have been obtained. The final systems equations remain as following:

The curve passes through the initial and final points

$$\begin{pmatrix} (t_1^2 - t_1) & (t_1^3 - t_1) \\ (t_2^2 - t_2) & (t_2^3 - t_2) \\ \cdots & \cdots \\ (t_{n-2}^2 - t_{n-2}) & (t_{n-2}^3 - t_{n-2}) \end{pmatrix} \begin{pmatrix} c_x \\ d_x \end{pmatrix} = \begin{pmatrix} x_1 - x_0 + x_0 \cdot t_1 - x_{n-1} \cdot t_1 \\ x_2 - x_0 + x_0 \cdot t_2 - x_{n-1} \cdot t_2 \\ \cdots \\ x_{n-2} - x_0 + x_0 \cdot t_{n-2} - x_{n-1} \cdot t_{n-2} \end{pmatrix},$$

$$\begin{pmatrix} (t_1^2 - t_1) & (t_1^3 - t_1) \\ (t_2^2 - t_2) & (t_2^3 - t_2) \\ \cdots & \cdots \\ (t_{n-2}^2 - t_{n-2}) & (t_{n-2}^3 - t_{n-2}) \end{pmatrix} \begin{pmatrix} c_y \\ d_y \end{pmatrix} = \begin{pmatrix} y_1 - y_0 + y_0 \cdot t_1 - y_{n-1} \cdot t_1 \\ y_2 - y_0 + y_0 \cdot t_2 - y_{n-1} \cdot t_2 \\ \cdots \\ y_{n-2} - y_0 + y_0 \cdot t_{n-2} - y_{n-1} \cdot t_{n-2} \end{pmatrix}$$

And: $$\left.\begin{matrix} a_x = x_0 \\ a_y = y_0 \end{matrix}\right\} \quad \left.\begin{matrix} b_x = x_{n-1} - x_0 - c_x - d_x \\ b_y = y_{n-1} - y_0 - c_y - d_y \end{matrix}\right\}$$

The curve passes through the initial and final points and has the specified tangent (first derivative) in the initial point: X and Y are related by the initial tangent module *k₀*

$$\begin{pmatrix} x_0' \cdot (t_1 - t_1^2) & (t_1^3 - t_1^2) & 0 \\ y_0' \cdot (t_1 - t_1^2) & 0 & (t_1^3 - t_1^2) \\ x_0' \cdot (t_2 - t_2^2) & (t_2^3 - t_2^2) & 0 \\ y_0' \cdot (t_2 - t_2^2) & 0 & (t_2^3 - t_2^2) \\ \cdots & \cdots & \cdots \\ x_0' \cdot (t_{n-2} - t_{n-2}^2) & (t_{n-2}^3 - t_{n-2}^2) & 0 \\ y_0' \cdot (t_{n-2} - t_{n-2}^2) & 0 & (t_{n-2}^3 - t_{n-2}^2) \end{pmatrix} \begin{pmatrix} k_0 \\ d_x \\ d_y \end{pmatrix} = \begin{pmatrix} x_1 - x_0 - (x_{n-1} - x_0) \cdot t_1^2 \\ y_1 - y_0 - (y_{n-1} - y_0) \cdot t_1^2 \\ x_2 - x_0 - (x_{n-1} - x_0) \cdot t_2^2 \\ y_2 - y_0 - (y_{n-1} - y_0) \cdot t_2^2 \\ \cdots \\ x_{n-2} - x_0 - (x_{n-1} - x_0) \cdot t_{n-2}^2 \\ y_{n-2} - y_0 - (y_{n-1} - y_0) \cdot t_{n-2}^2 \end{pmatrix}$$

And: $$\left.\begin{matrix} a_x = x_0 \\ a_y = y_0 \end{matrix}\right\} \quad \left.\begin{matrix} b_x = k_0 \cdot x_0' \\ b_y = k_0 \cdot y_0' \end{matrix}\right\} \quad \left.\begin{matrix} c_x = x_{n-1} - x_0 - k_0 \cdot x_0' - d_x \\ c_y = y_{n-1} - y_0 - k_0 \cdot y_0' - d_y \end{matrix}\right\} \rightarrow \left.\begin{matrix} c_x = x_{n-1} - x_0 - b_x - d_x \\ c_y = y_{n-1} - y_0 - b_y - d_y \end{matrix}\right\}$$

The curve passes through the initial and final points and has the specified tangent (first derivative) in the final point: X and Y are related by the final tangent module *kₙ₋₁*

$$\begin{pmatrix} x_{n-1}' \cdot (t_1^2 - t_1) & (t_1 - 2 \cdot t_1^2 + t_1^3) & 0 \\ y_{n-1}' \cdot (t_1^2 - t_1) & 0 & (t_1 - 2 \cdot t_1^2 + t_1^3) \\ x_{n-1}' \cdot (t_2^2 - t_2) & (t_2 - 2 \cdot t_2^2 + t_2^3) & 0 \\ y_{n-1}' \cdot (t_2^2 - t_2) & 0 & (t_2 - 2 \cdot t_2^2 + t_2^3) \\ \cdots & \cdots & \cdots \\ x_{n-1}' \cdot (t_{n-2}^2 - t_{n-2}) & (t_{n-2} - 2 \cdot t_{n-2}^2 + t_{n-2}^3) & 0 \\ y_{n-1}' \cdot (t_{n-2}^2 - t_{n-2}) & 0 & (t_{n-2} - 2 \cdot t_{n-2}^2 + t_{n-2}^3) \end{pmatrix} \begin{pmatrix} k_{n-1} \\ d_x \\ d_y \end{pmatrix} = \begin{pmatrix} x_1 - x_0 - 2 \cdot (x_{n-1} - x_0) \cdot t_1 + (x_{n-1} - x_0) \cdot t_1^2 \\ y_1 - y_0 - 2 \cdot (y_{n-1} - y_0) \cdot t_1 + (y_{n-1} - y_0) \cdot t_1^2 \\ x_2 - x_0 - 2 \cdot (x_{n-1} - x_0) \cdot t_2 + (x_{n-1} - x_0) \cdot t_2^2 \\ y_2 - y_0 - 2 \cdot (y_{n-1} - y_0) \cdot t_2 + (y_{n-1} - y_0) \cdot t_2^2 \\ \cdots \\ x_{n-2} - x_0 - 2 \cdot (x_{n-1} - x_0) \cdot t_{n-2} + (x_{n-1} - x_0) \cdot t_{n-2}^2 \\ y_{n-2} - y_0 - 2 \cdot (y_{n-1} - y_0) \cdot t_{n-2} + (y_{n-1} - y_0) \cdot t_{n-2}^2 \end{pmatrix}$$

And: $$\left.\begin{matrix} a_x = x_0 \\ a_y = y_0 \end{matrix}\right\} \quad \left.\begin{matrix} b_x = -k_{n-1} \cdot x_{n-1}' + 2 \cdot x_{n-1} - 2 \cdot x_0 + d_x \\ b_y = -k_{n-1} \cdot y_{n-1}' + 2 \cdot y_{n-1} - 2 \cdot y_0 + d_y \end{matrix}\right\} \quad \left.\begin{matrix} c_x = -x_{n-1} + x_0 + k_{n-1} \cdot x_{n-1}' - 2 \cdot d_x \\ c_y = -y_{n-1} + y_0 + k_{n-1} \cdot y_{n-1}' - 2 \cdot d_y \end{matrix}\right\}$$

The curve passes through the initial and final points and has the specified tangent (first derivative) in the initial and final points: X and Y are related by the initial and final tangent modules *k₀* and *kₙ₋₁*

$$\begin{pmatrix} \left(x_0'\cdot t_1 - 2\cdot x_0'\cdot t_1^2 + x_0'\cdot t_1^3\right) & \left(x_{n-1}'\cdot t_1^3 - x_{n-1}'\cdot t_1^2\right) \\ \left(y_0'\cdot t_1 - 2\cdot y_0'\cdot t_1^2 + y_0'\cdot t_1^3\right) & \left(y_{n-1}'\cdot t_1^3 - y_{n-1}'\cdot t_1^2\right) \\ \left(x_0'\cdot t_2 - 2\cdot x_0'\cdot t_2^2 + x_0'\cdot t_2^3\right) & \left(x_{n-1}'\cdot t_2^3 - x_{n-1}'\cdot t_2^2\right) \\ \left(y_0'\cdot t_2 - 2\cdot y_0'\cdot t_2^2 + y_0'\cdot t_2^3\right) & \left(y_{n-1}'\cdot t_2^3 - y_{n-1}'\cdot t_2^2\right) \\ \ldots & \ldots \\ \left(x_0'\cdot t_{n-2} - 2\cdot x_0'\cdot t_{n-2}^2 + x_0'\cdot t_{n-2}^3\right) & \left(x_{n-1}'\cdot t_{n-2}^3 - x_{n-1}'\cdot t_{n-2}^2\right) \\ \left(y_0'\cdot t_{n-2} - 2\cdot y_0'\cdot t_{n-2}^2 + y_0'\cdot t_{n-2}^3\right) & \left(y_{n-1}'\cdot t_{n-2}^3 - y_{n-1}'\cdot t_{n-2}^2\right) \end{pmatrix} \begin{pmatrix} k_0 \\ k_{n-1} \end{pmatrix} = \begin{pmatrix} x_1 - x_0 - 3\cdot(x_{n-1}-x_0)\cdot t_1^2 + 2\cdot(x_{n-1}-x_0)\cdot t_1^3 \\ y_1 - y_0 - 3\cdot(y_{n-1}-y_0)\cdot t_1^2 + 2\cdot(y_{n-1}-y_0)\cdot t_1^3 \\ x_2 - x_0 - 3\cdot(x_{n-1}-x_0)\cdot t_2^2 + 2\cdot(x_{n-1}-x_0)\cdot t_2^3 \\ y_2 - y_0 - 3\cdot(y_{n-1}-y_0)\cdot t_2^2 + 2\cdot(y_{n-1}-y_0)\cdot t_2^3 \\ \ldots \\ x_{n-2} - x_0 - 3\cdot(x_{n-1}-x_0)\cdot t_{n-2}^2 + 2\cdot(x_{n-1}-x_0)\cdot t_{n-2}^3 \\ y_{n-2} - y_0 - 3\cdot(y_{n-1}-y_0)\cdot t_{n-2}^2 + 2\cdot(y_{n-1}-y_0)\cdot t_{n-2}^3 \end{pmatrix}$$

And:
$$\left\{\begin{array}{llll} \left.\begin{array}{l} a_x = x_0 \\ a_y = y_0 \end{array}\right\} & \left.\begin{array}{l} b_x = k_0\cdot x_0' \\ b_y = k_0\cdot y_0' \end{array}\right\} & \left.\begin{array}{l} c_x = 3\cdot(x_{n-1}-x_0) - k_{n-1}\cdot x_{n-1}' - 2\cdot k_0\cdot x_0' \\ c_y = 3\cdot(y_{n-1}-y_0) - k_{n-1}\cdot y_{n-1}' - 2\cdot k_0\cdot y_0' \end{array}\right\} \rightarrow & \left.\begin{array}{l} c_x = 3\cdot(x_{n-1}-x_0) - k_{n-1}\cdot x_{n-1}' - 2\cdot b_x \\ c_y = 3\cdot(y_{n-1}-y_0) - k_{n-1}\cdot y_{n-1}' - 2\cdot b_y \end{array}\right\} \\ & & \left.\begin{array}{l} d_x = k_{n-1}\cdot x_{n-1}' + k_0\cdot x_0' - 2\cdot(x_{n-1}-x_0) \\ d_y = k_{n-1}\cdot y_{n-1}' + k_0\cdot y_0' - 2\cdot(y_{n-1}-y_0) \end{array}\right\} \rightarrow & \left.\begin{array}{l} d_x = k_{n-1}\cdot x_{n-1}' + b_x - 2\cdot(x_{n-1}-x_0) \\ d_y = k_{n-1}\cdot y_{n-1}' + b_y - 2\cdot(y_{n-1}-y_0) \end{array}\right\} \end{array}\right.$$

In order to solve linear equation systems we can use any least squares method, but in our particular case we have used the Householder method that is more stable numerically that the conventional method [46, 47]. In order to verify the performed approximation, we will check out that the maximum distance between the resampled points (pr_j) and the approximated points (pa_j) by means the cubic curve with the corresponding parameters t_j is lower than the value MAX_DISTANCE:

$$\begin{gathered} pr_j = (xr_j, yr_j), \quad pa_j = \left(a_x + b_x\cdot t_j + c_x\cdot t_j^2 + d_x\cdot t_j^3, a_y + b_y\cdot t_j + c_y\cdot t_j^2 + d_y\cdot t_j^3\right) \\ d_j = \left\| pr_j - pa_j \right\| \end{gathered} \tag{22}$$

Acknowledgments

Spanish Ministry of Science and Education and the FEDER Funds, through CUESKETCH (Ref. DPI2007-66755-C02-01) and HYMAS projects (Ref. DPI2010-19457) partially supported this work.

References

[1] P. Company, M. Contero, P.A.C. Varley, N. Aleixos, F. Naya. "Computer-aided sketching as a tool to promote innovation in the new product development process". Computers in Industry 60(8): 592-603, 2009.

[2] P. Company, P.A.C. Varley, A. Piquer, M. Vergara, J. Sánchez-Rubio. "Benchmarks for Computer-based Segmentation of Sketches", In Pre-Proceedings of The Eighth IAPR International Workshop on Graphics Recognition, GREC 2009, France, July 22-23, 2009, pp. 103-114.

[3] A. Wolin, B. Eoff, T. Hammond. “Shortstraw: A simple and effective corner finder for poly-lines”. In EURO-GRAPHICS 5th Annual Workshop on Sketch-Based Interfaces and Modeling, 2008, pp. 33–40.

[4] Y. Xiong and J.J. LaViola Jr. “Revisiting ShortStraw – Improving Corner Finding in Sketch-Based Interfaces”, In EUROGRAPHICS 6th Annual Workshop on Sketch-Based Interfaces and Modeling, 2009, pp. 101-108.

[5] J. Pu and D. Gur. “Automated freehand sketch segmentation using radial basis functions”. Computer-Aided Design 41 (2009) 857-864.

[6] F. Mokhtarian, S. Abbasi, Robust automatic selection of optimal views in multi-view free-form object recognition, Pattern Recognition 38 (2005) 1021-1031.

[7] S. Jaggi, W.C. Karl, S. G. Mallat, A.S. Willsky, Silhouette recognition using high-resolution pursuit, Pattern Recognition 32 (1999) 753-771.

[8] S. Osowski, D.D. Nghia, Fourier and wavelet descriptors fro shape recognition using neural networks – a comparative study, Pattern Recognition 35 (2002) 1949-1957.

[9] G.Y. Chen, T.D. Bui, A. Krzyzak, Rotation invariant pattern recognition using ridgelets, wave-let cycle-spinning and Fourier features, Pattern Recognition 38 (2005) 2314-2322.

[10] G. Chen, T.D. Bui, Invariant Fourier-wavelet descriptor for pattern recognition, Pattern Rec-ognition 32 (1999) 1083-1088.

[11] C.H. Park, H. Park, Fingerprint classification using fast Fourier transform and non-linear discriminant analysis, Pattern Recognition 38 (2005) 495-503.

[12] L. Gennari, L.B. Kara, T.F. Stahovich, K. Shimad. “Combining geometry and domain know-ledge to interpret hand-drawn diagrams”. Computers & Graphics 29 (2005) 547–562.

[13] G. Feng, C. Viard-Gaudin, Z. Sun. “On-line hand-drawn electric circuit diagram recognition using 2D dynamic programming”. Pattern Recognition 42 (2009) 3215 – 3223.

[14] J. Mas, J. Llados, G. Sanchez, J.A.P. Jorge. A syntactic approach based on distortion-tolerant Adjacency Grammars and a spatial-directed parser to interpret sketched diagrams. Pattern Recognition 43 (2010) 4148-4164.

[15] R. Arandjelovi’c, T.M. Sezgin. Sketch recognition by fusion of temporal and image-based features. Pattern Recognition 44 (2011) 1225-1234.

[16] D.H. Rubine. “Specifying Gestures by Example”, Computer Graphics, 25 (4), 1991, pp. 329-337.

[17] A. Ajay, V. Vo, T.D. Kimura. “RecognisingMultistroke Shapes: An Experimental Evaluation”, ACM (UIST’93), Atlanta, 1993, pp. 121-128.

[18] M.D. Gross. “Recognising and Interpreting Diagrams in Design”, Proceedings of ACM (AVI’94), Italy, 1994, pp.88-94.

[19] M.J. Fonseca and J. Jorge, “Using Fuzzy Logic to Recognise Geometric Shapes Interactively”, 9th IEEE Conference on Fuzzy Systems, 1, 2000, pp. 291-296.

[20] J. Xiangyu, L. Wenyin, S. Jianyong, Z. Sun, “On-Line Graphics Recognition”. Conference on Computer Graphics and Applications, 2002, pp. 256-264.

[21] S. Zhengxing, W. Liu, P. Binbin, Z. Bin, S.Jianyong, “User Adaptation for Online Sketchy Shape Recognition”, GREC 2003, 305-316.

[22] D. Willems, R. Niels, M. van Gerven, L. Vuurpijl. “Iconic and multi-stroke gesture recognition”. Pattern Recognition 42 (2009) 3303 – 3312.

[23] X. Zhang, D. Zhao. A parallel algorithm for detecting dominant points on multiple digital curves. Pattern Recognition. Vol. 30, No. 2, pp. 239-244, 1997.

[24] B.K. Ray, K.S. Ray. Corner detection using iterative Gaussian smoothing with constant window size. Pattern Recognition, Vol. 28, No. 11, pp. 1765-1781, 1995.

[25] B.K. Ray, R. Pandyan. ACORD—an adaptive corner detector for planar curves. Pattern Recognition 36 (2003) 703-708.

[26] R. Neumann, G. Teisseron. Extraction of dominant points by estimation of the contour fluctuations. Pattern Recognition 35 (2002) 1447-1462.

[27] F. Arrebola, F. Sandoval. Corner detection and curve segmentation by multiresolution chain-code linking. Pattern Recognition 38 (2005) 1596-1614.

[28] T. Sezgin, T. Stahovich, R. Davis. “Sketch based interfaces: Early processing for sketch understanding”, In Workshop on Perceptive User Interfaces, 2001.

[29] T. Stahovich. “Segmentation of pen strokes using pen speed”. In Proceedings 2004 AAAI Fall Symposium on Making Pen-Based Interaction Intelligent and Natural, 2004.

[30] T. Sezgin and R. Davis. “Scale-space based feature point detection for digital ink”. In SIG-

GRAPH '06: ACM SIGRRAPH 2006 Courses, NY USA, 2006, ACM, p. 29.

[31] D. Kim and M-J. Kim. "A curvature estimation for pen input segmentation in sketch-based modelling". In Computer-Aided Design, 2006, 38, pp. 238-248.

[32] B. Yu. "Recognition of freehand sketches using Mean Shift". In IUI '03: Proceedings of the 8th international conference on Intelligent user interfaces, 2003, ACM, pp. 204-210.

[33] H. Hse, M. Shilman, A.R. Newton. "Robust sketched symbol fragmentation using templates". In IUI'04: Proceedings of the 9th international conference on Intelligent user interfaces, 2004, pp. 156-160.

[34] B. Paulson and T. Hammond. "Paleosketch: Accurate primitive sketch recognition and beautification". In IUI '08: Proceedings of the 13th international conference on Intelligent user interfaces, 2008, pp. 1-10.

[35] C. Alvarado and R. Davis. "Sketchread: a multi-domain sketch recognition engine". In UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology, NY USA, 2004, ACM Press, pp. 23-32.

[36] T. Hammond and R. Davis. "Ladder, a sketching language for user interface developers". Elsevier, Computers and Graphics, 28 (2005), pp. 518-532.

[37] B. Sarkar, L.K. Singh, D. Sarkar, Approximation of digital curves with line segments and circular arcs using genetic algorithms, Pattern Recognition Lett. 24 (15) (2003) 2585–2595.

[38] X. Zhang, J. Song, G. Dai, M. R. Lyu. "Extraction of Line Segments and Circular Arcs From Freehand Strokes Based on Segmental Homogeneity Features". IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART B: CYBERNETICS, VOL. 36, NO. 2, APRIL 2006.

[39] T.P. Nguyen, I. Debled-Rennesson. A discrete geometry approach for dominant point detection. Pattern Recognition 44 (2011) 32-44.

[40] S-F. Qin, D.K. Wright, I.N. Jordanov. On-line segmentation of freehand sketches by knowledge-based nonlinear thresholding operations. Pattern Recognition 34 (2001) 1885-1893.

[41] M. Masry, D. Kang, H. Lipson. "A freehand sketching interface for progressive construction of 3D objects". Computers & Graphics 29 (2005) 563–575.

[42] M. Bein and S. Havemann, A. Stork, D. Fellner. "Sketching Subdivision Surfaces". EURO-

GRAPHICS Symposium on Sketch-Based Interfaces and Modeling (2009)

[43] D. G. Fernández-Pacheco, F. Albert, N. Aleixos, J. Conesa, M. Contero. “Automated tuning of parameters for the segmentation of freehand sketches”. Proceedings of the International Conference on Computer Graphics Theory and Applications (GRAP 2011), 2011, pp. 321-329.

[44] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi. “Optimization by simulated annealing”, Science, vol. 220 (1983), no. 4598, pp. 671-680.

[45] G. Farin. “Curves and surfaces for CAGD. A practical guide”. Academic Press Ltd., San Diego, 1993.

[46] R. L. Burden, J. D. Faires. “Análisis numérico”. Grupo Editorial Iberoamérica, 1985.

[47] J. L. Goldberg. “Matrix Theory with Applications”. McGrawHill, 1991.

F. Albert is Associate Professor of Engineering Graphics, CAD, and 3D animation and Graphic Design with the Engineering Design Department at Polytechnic University of Valencia. His fields of interest are focused on graphic pattern analysis, reconstruction and design, and development of spatial abilities using new technologies.

D.G. Fernández-Pacheco is Assistant Professor of Engineering Graphics and CAD with the Graphical Expression Department at Polytechnic University of Cartagena. His fields of interest are focused on geometric modeling, computer vision, multi-agent systems, sketch recognition and calligraphic interfaces.

N. Aleixos received an award for her Ph.D, and worked for private companies and at the Public Research Institute IVIA developing machine vision systems. She is Associate Professor at Polytechnic University of Valencia. Her fields of interest are focused on computer vision systems, image analysis, multi-agent systems, sketch recognition and natural interfaces.