www.regeo.uji.es
Technical Reports. Ref. 03/2007

Geometry and Gesture Recognizers for Prescriptive Sketch Interpretation

Nuria Aleixos[1], Ferran Naya[2], Pedro Company[1], and Manuel Contero[2]

[1]Department of Mechanical Engineering and Construction, Universitat Jaume I, Spain
[2]DEGI-ETSII, Polytechnic University of Valencia, Spain

Abstract

Prescriptive sketches are used during the product development process to give detailed instructions about the geometric characteristic of the new product for 3D modeling or drawing generation. This paper analyzes the recognizers that use the ParSketch application for performing the online transformation of a prescriptive sketch into a 2D parametric drawing. The geometric recognizer RecoGeo analyzes and converts a sketch into its constituent primitives (outlined sketch). The algorithm bases its recognition on features as direction and curvature signatures. Once the designer has introduced the complete outlined sketch, it can be edited, dimensioned, constrained and lastly swept to create a solid. To carry out the actions mentioned before, a gesture recognizer –RecoGes– has been developed. RecoGes provides an alphabet of geometric/dimensional constraints and gesture commands, and bases the recognition of the intended used action on Fourier descriptors of the polar and direction signatures, resulting the final classification from a non-linear classificatory function. Discussion and results for both classifiers are commented

Categories and Subject Descriptors (according to ACM CCS): I.4.6 [Image Processing and Computer Vision]: Segmentation – edge and feature detection. H.5.2 [User Interfaces]: Interaction styles

1. Introduction

Engineering sketches play an important role during the whole product development process [Van05]. But these sketches are used with different purposes during this process. This means different contents and kinds of representations. Following the classification proposed by Ferguson [Fer92] we can distinguish among *thinking sketches* used to focus and guide non verbal thinking, *talking sketches* employed to support discussion on the design with colleagues and *prescriptive sketches* applied to give instructions to the draftsman in charge of making the finished drawing.

Thinking sketches are present especially during the conceptual design stage, where they are used as a very valuable tool to explore design alternatives and to serve as a temporary ideas repository. Computer support for this kind of sketch has been oriented to improve the functionality provided by sketching on paper, trying to provide an added value to sketching on a digital environment, for example, using a Tablet-PC or LCD graphic tablet. This extra functionality has been directed to improve the graphic quality of the sketch by means of beautification functionality, providing as output an improved 2D representation, or it has been oriented to transform the 2D sketch into a 3D model.

Two main approaches have been followed to create the 3D model from a 2D sketch: reconstruction-based systems [CNJC03], where a sketched single view projection is interpreted and a plausible 3D model is automatically generated; and gestural-based modeling [ZHH96] where modeling operations are encoded by gestures that are used to create a 3D object transforming 2D sketched sections. From these two approaches the reconstruction based is the most transparent to the user, since he has only to create the sketch and it doesn't require a priori knowledge of a gestural command set.

With respect to talking sketches, computer support for this kind of sketch can be found in the computer support collaborative work (CSCW) community. Usually these applications are focused on promoting both collaborative creation and sharing of 2D sketches during workgroup interaction [Gre91].

For the moment, there is no computer support for prescriptive sketches in the sense of offering some extra functionality with respect to plain paper sketching. This kind of sketch can include complex graphic symbols such as those represented by dimensions, geometric tolerances and surface finishes. This introduces additional complexity in order to create a computer aided sketching (CAS) applica-

tion because more sophisticated recognizers are required. So, a first step to create a CAS application to support prescriptive sketches would be the automatic beautification, then perhaps the automatic conversion into a parametric 2D drawing and finally the interpretation of the multiview representations to create a 3D model from orthographic projections.

In what follows we will present the ParSketch prototype that is a CAS application than performs an online conversion of a prescriptive sketch into a 2D parametric drawing. Then, both the geometry and gesture recognizers used by this application are described and analyzed

2. Related work

Several research works have been carried out in order to fit a free sketch by the designer into an outlined sketch. Many of them work in a similar way: when a stroke is introduced they search for changes in direction to separate pieces of the stroke and then, they approximate each stroke piece to an intended shape.

For instance, Qin et al. [QWJ01] recognize freehand sketches in a on-line system using features as curvature, speed and acceleration graphs. They used the last two graphs to smooth the sketched stroke to be processed for segmentation. Then, searching sharp zones in the stroke direction, corner points are marked and lastly, pieces of stroke are approximated to lines or arc entities depending on their circularity or straightness, determined by thresholds.

Also Yu [Yu03] has developed an algorithm capable of converting a freehand sketch into an outlined one, basing the recognition process on the direction and curvature features. Yu first smoothes the sketch using mean shift on previous features, then searches for vertexes in curvature graph and lastly approximates to geometric entities.

Other techniques are used in sketch recognition to detect symbols, diagrams, geometric shapes and other user command gestures. For instance Rubine [Rub91] classifies sketches into different gestures or characters, basing the recognition process on extracted features as ratios of area-perimeter, initial angle of the sketch, the total angle traverse, and so on.

Apte et al. [AK93] developed an algorithm for recognizing hand drawn geometric shapes (rectangles, triangles, circles, ellipses, diamonds and lines) from multistroke sketches entered without pauses. They based the recognition on what they call filters, which are ratios from features as area, perimeter, convex-hull, etc.

Other works, as carried out by Calhoun et al. [CSKK02], recognize multi-stroke symbols into its constituent primitives using both pen speed and curvature features. They use two methods: in the first one they train the system introducing the symbols in a strict order and, on the other one, they study a heuristic method based on relaxing assumptions as the sequence order, the exact number of primitives, and relative orientation, to allow the designer to introduce the symbol in a different sequence order, what is more expensive computationally.

Jorge et al. use percentiles for area and perimeter ratios to detect shapes as circles, rectangles, triangles, lines, etc. basing their recognition on fuzzy sets [PBJS*04].

Moreover recognizing known shapes and symbols, it is necessary to interpret or recognize other user sketches as gestures to convert them into CAD application commands. Some symbol recognition has been done basing their recognition process on shape descriptors as perimeter, area, ratios between them, etc. However, methods based on Fourier descriptors achieve better results than methods based on shape descriptors as used above. In this field, Zang et al. [ZL02] test and compare different Fourier descriptors using a standard shape database, and conclude that centroid distance is the best shape signature in terms of robustness, computation complexity and convergence speed of its Fourier series. Fourier descriptors were also used by Harding et al. [HE04] for the recognition of hand gesture trajectories. Other examples of applications that use Fourier descriptors are the detection of users' hand movement in a system to achieve an augmented reality tool (Licsar et al. [LS04]) or OCR applications (Hopkins et al. [HA04]).

Park and Park [PP05] use Fourier transform to describe fingerprints that are classified by means of a non-linear discriminant analysis. Although the accuracy and robustness of sketch recognising for symbols and gestures have improved recently, there are still many issues involved in real applications. Hong et al. [HLLM02] describe the drawbacks of sketch recognisers at this time.

In this work, we present the RecoGeo geometric recognizer that extracts from freehand sketches the geometric primitives they are composed of. Later, these recognized entities are parametrized.

Also in this work, we present the RecoGes gesture recognizer, which works on-line, to interpret user gestures as dimensional/geometric constraints to parametrize the previous outlined sketch, or as commands to extrude, revolve or edit the sketch. Both recognizers have been used in the ParSketch application that is presented in the following point.

3. The ParSketch application

ParSketch is a 2D sketching application that can interpret complex strokes composed by different geometric entities that are automatically segmented into its basic components (line segments and arcs). Internally this application uses the 2D DCM parametric engine from UGS' D-Cubed to provide parametric control of the shape.

Interaction with the application relies on two recognizers as shown in Figure 1. RecoGeo is used for geometry analysis and RecoGes for gesture interpretation. They are invoked using the drawing pressure as a mode discriminator (inking or gesturing).

Some of the gestures that the system supports at present are included in table 1. It is possible to add new gestures to the system as the gesture recognizer can be trained providing new gesture samples.

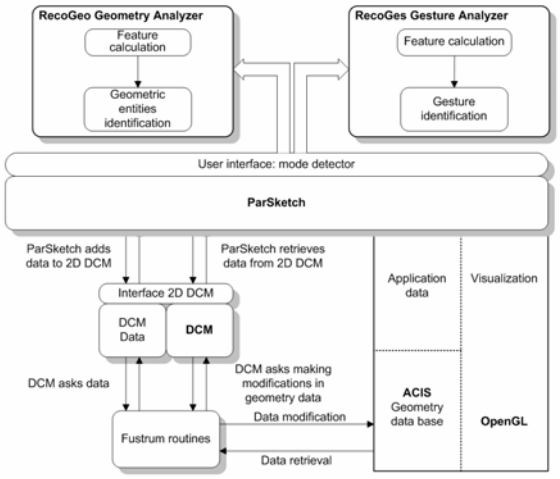


Figure 1: *Block diagram of the ParSketch application*

3.1 Calligraphic interface

The user introduces the geometry creating a freehand sketch directly onto the screen of the Tablet-PC. The geometry recognizer provides the defining parameters that control the geometric entities found in the sketch. After a preprocessing stage that cleans up input data and adjusts edges to make sure they meet precisely at common endpoints in order to get geometrically consistent figures, the application by means of the 2D DCM application provides a perfect line drawing showing all the detected geometric constraints.

Users can delete unwanted geometric constraints drawing a scratching gesture over them. They can also add new constraints drawing their associated gestures near the geometric entities where they must be applied.

Constraints are managed by the dialog box presented in Figure 2. The user can choose the constraints that the system will automatically use to control the sketch. Users can also manage the beautification action modifying the tolerance values used to decide if a geometric constraint is verified. Handwritten number recognition is provided by the Windows XP Tablet PC Edition operative system.

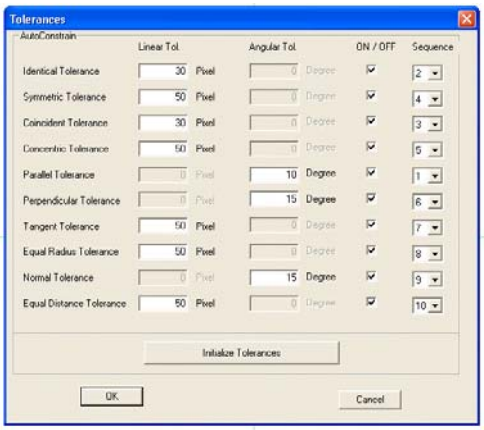


Figure 2: *Tolerance settings*

ParSketch application offers some innovative ways of controlling the shape after it has been analyzed by the RecoGeo engine and a beautified constrained model is presented to the user. In Figure 3 we present an example of dimensional control using the dimension gesture. It operates in two stages. When it is introduced by the user, the system interprets it as a measure command, providing the text dimension automatically. This can be seen in Figure 3.b. Then, if the user writes something near the dimension text it is interpreted as a new value for the dimension. This value is fed in the 2D DCM parametric engine and an updated shape is presented to the user.

Comparing the operation of the ParSketch system with a standard WIMP parametric 2D CAD application we can say that the basic functionality is practically equivalent. The main advantage of ParSketch is related to its ability to interpret complex compound strokes that include several geometric entities. This makes the input of geometry very agile requiring less interaction with the user than with a WIMP interface, where each kind of geometry entities must be explicitly indicated to the system using the proper menu or icon.

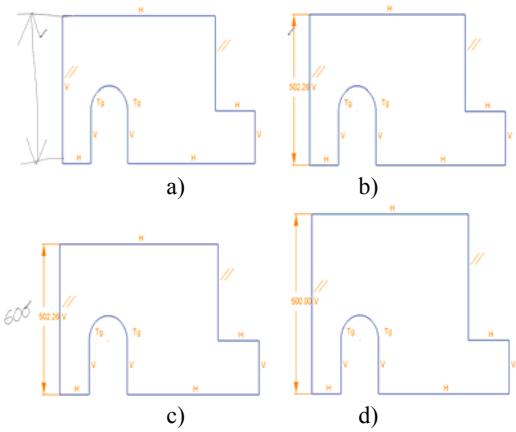


Figure 3: *ParSketch sketching sequence*

4. Geometry recognizer

In order to recognize geometric entities from freehand sketches, we have developed a geometry recognizer that we call RecoGeo, whose structure is presented in Figure 4. This algorithm analyzes a stroke when a pen up event is encountered, and fits a sketch into an outlined sketch (poly-line). The procedure consists of five steps:

1. Direction and curvature signatures of the stroke are calculated.
2. The mean shift procedure is used to smooth the two former signatures, in order to avoid shaky writing.
3. Then, changes in direction of strokes are detected by the algorithm, and vertexes are detected using peaks in curvature signature.
4. A refinement of vertexes (reduction of the number of vertexes found) is carried out when their path length is less than a desired threshold, fixed empirically.
5. Finally, the type of each entity between two pair of vertexes is decided by means of a primitive approxi-

mation method. Supported entities are lines, arcs, circles and ellipses, and its type is decided by looking into the direction signature, and analyzing the points corresponding to that piece of the stroke.

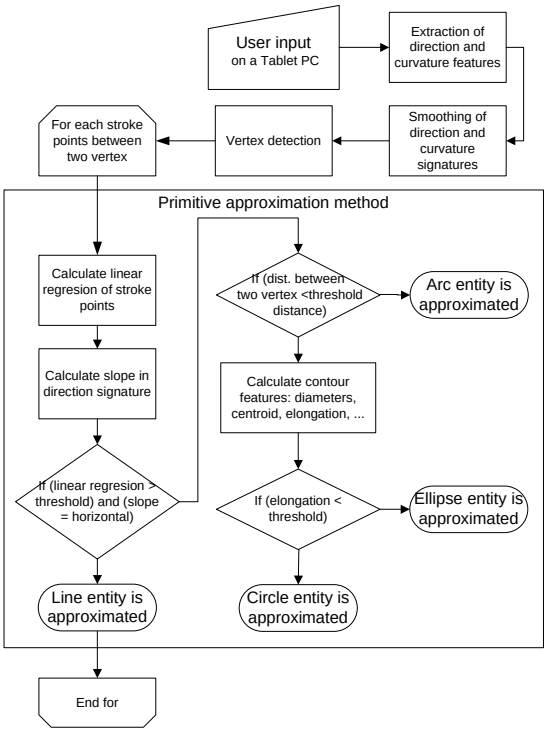


Figure 4: *Geometry recognition flowchart*

4.1 Extraction of direction and curvature signatures

The stroke is composed by a set of coordinates (*x,y*) in an interval of pen-down and pen-up events. In such a stroke, there are two visual features which can guide recognition: *direction* and *curvature*. The *direction* gives the angles between two consecutive pairs of coordinates in the range [-π, π]. The *curvature* gives the arctangent of the direction between two consecutive points, to inform on how opened or closed such piece of the stroke is.

Direction can be defined as: $$d_n = \frac{\sum_{i=n-k}^{n+k} \theta(i, i+1)}{2k+1} \quad (1)$$

And curvature as: $$c_n = \frac{\sum_{i=n-k}^{n+k-1} \phi(d_{i+1}, d_i)}{P(n-k, n+k)} \quad (2)$$

In these equations, n is defined as the number of stroke points, k is the neighbourhood size around the n-*th* stroke point, P is the path length of the current stroke point and its neighbourhood size (in our case we assume $k=1$ as the neighbourhood size of any stroke point), θ is the angle between two consecutive stroke points shifted in the range [-π, π], and ϕ is the arctangent of the angles of three consecutive points.

4.2 Smoothing the sketch

We have used the mean shift procedure to smooth the collected stroke, since it has proven to give good results in analyzing clusters in feature space, and in eliminating noise. The mean shift procedure [Yu03] can be defined as:

$$z_{j+1} = \frac{\sum_{i=1}^{n} x_i k'\left(\left\|\frac{z_j - x_i}{h}\right\|^2\right)}{\sum_{i=1}^{n} k'\left(\left\|\frac{z_j - x_i}{h}\right\|^2\right)}, j = 1,2,... \quad (3)$$

In order to apply this procedure to the two-dimensional direction-curvature joint space, we will consider x_i, $i=1,2...n$ as the input vector of the mean shift procedure, that is, the joint space d_i and c_i; and z_i, $i=1,2...n$ as the output vector of smoothed discrete values of direction and curvature, that is, the new direction and curvature signatures after smoothing. The term h is the "bandwidth", that is, the smoothing parameter, which has been adjusted to h_d and h_c, depending on direction and curvature respectively. In such a way we have the following:

$$h_d = \sqrt{\frac{\sum_{i=1}^{n-1} \phi^2 (d_{i+1} - d_i)}{n-1}} \quad h_c = \sqrt{\frac{\sum_{i=1}^{n-1} (c_{i+1} - c_i)^2}{n-1}} \quad (4)$$

Regarding to the direction-curvature joint space, we have that mean shift vector is:

$$y_{j+1} = \begin{pmatrix} d_{j+1} \\ c_{j+1} \end{pmatrix} = \frac{\sum_{i=1}^{n} \left[x_i . \exp\left(-\frac{(d_j - d_i)^2}{2h_d^2} \right) . \exp\left(-\frac{(c_j - c_i)^2}{2h_c^2} \right) \right]}{\sum_{i=1}^{n} \left[\exp\left(-\frac{(d_j - d_i)^2}{2h_d^2} \right) . \exp\left(-\frac{(c_j - c_i)^2}{2h_c^2} \right) \right]} \quad (5)$$

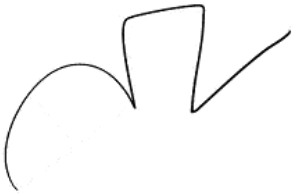
The algorithm remains as follows:

- Initialize the first vector result $y_{r,1}$ with the maximum values of direction and curvature x_r, $r=1,2...n$ (that is d_i and c_i).
- Compute $y_{r,s+1}$ iteratively according to (5) procedure for each stroke point until convergence, where the criterion for convergence is set to $\left| d_{r,s+1} - d_{r,s} \right| \le h_d / 100$ for direction and to $\left| c_{r,s+1} - c_{r,s} \right| \le h_c / 100$ for curvature.
- Assign each output convergent vector to z_r, $r=1,2...n$, so it contains smoothed direction and curvature signatures.

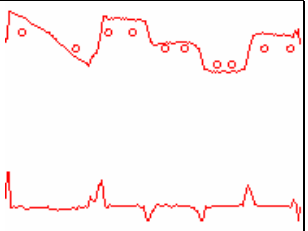
4.3 Finding vertexes

The smoothed curvature signature guides us to find vertexes in the stroke when seeking for local maxima values, that is, where curvature changes considerably (peaks in curvature signature). For this purpose, the designed algorithm to decide the location of vertexes in the stroke is:

a) Freehand sketch composed of 1 arc and 4 lines (in this order).



b) Direction (up) and curvature (down) signatures. The circles are the zones of the direction signature that are analyzed to help us to decide if it is an arc or a line.



c) Direction (up) and curvature (down) signatures after smoothing with mean shift.

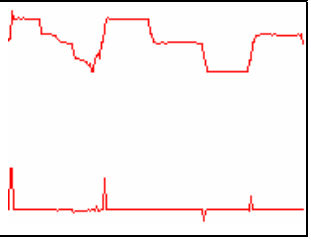


Figure 5: *An example of stroke (a); its corresponding direction and curvature signatures (b); and the smoothed former signatures (c)*

- Scan each stroke point and mark the ones as a vertex if its curvature is a local maximum or minimum, and the condition:

  $|c_i - c_{i-1}| + |c_i - c_{i+1}| \geq 2h_c$ is accomplished.

  Where the i-*th* point is the current stroke point and h_c is the smoothing parameter for curvature.
- Finish the vertexes seek, merging two consecutive vertexes when the path length between them is lower than a threshold.

Refinement of vertexes is necessary because, otherwise, small entities would be found, which are very common in freehand sketches. After refinement, in the example of Figure 6, one line, one arc and one line respectively are found, instead of two lines at the end.

Figure 6: *Merging two consecutive vertexes*

In the example of Figure 7 our algorithm finds six vertexes and detects the first entity as an arc and the following entities as lines. This has been achieved studying both, the original direction signature before mean shift procedure between each two consecutive vertexes, and the correlation with the linear regression of the stroke points of each entity. In the next section, we will explain the primitive approximation method implemented by our recognizer, to decide among line, arc, circle and ellipse.

The recognition algorithm provides the distinctive elements of any entity of the outlined sketch. For a line, the these are the two end points, for an arc they are the centre point and the two end points, for a circle they are the centre and a radius and for an ellipse they are the two axes and their orientation..

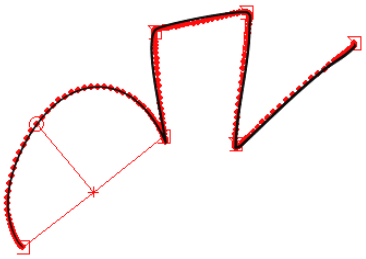


Figure 7: *Geometric recognition for sketch of Figure 5*

4.4 Primitive Approximation Method

We use two basic features to approximate the pieces of stroke to primitive entities: the original direction signature and the correlation of the stroke points from their linear regression. Next we explain the method applied to the different entities that our algorithm is able to distinguish.

LINE ENTITY.

Our analysis begins supposing that we have a line entity and, if recognition fails, then we check for other entities.

Secondly, when analyzing the stroke points, we can appreciate its curvature also looking into the morphology of the original direction graph. If the slope is found to be horizontal (value of slope <= 0,01), the portion of the stroke between the corresponding two vertexes will be approximated to a line, otherwise (value of slope > 0,01), we will check if it is an arc, circle or ellipse.

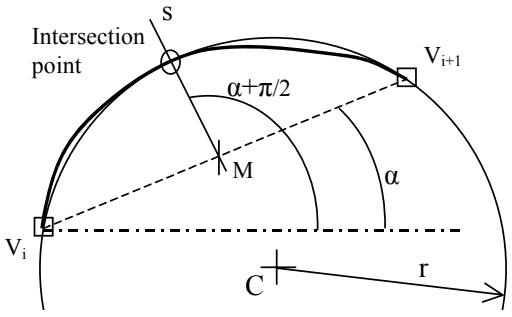


Figure 8: *Fitting a piece of sketch to an arc entity*

ARC ENTITY.

The algorithm to find an arc consists of checking first the proximity of the two vertexes of the stroke piece, considering they are far enough from each other to start primitive approximation. In this case, we compute the perpendicular line *s* to that one formed by the two vertexes, and force it to pass through the midpoint between those vertexes, as explained in Figure 8.

Once we have *s*, we scan the stroke points between the two vertexes and keep the one belonging to that line, that is, the intersection point. Now, we are provided with three points that define a circle, of which we will keep just the arc segment containing the three points, and we are already able to calculate its radius. The arc then will be computed as the arc segment of a circle through three points, with its radius *r* and centre *C*.

CIRCLE AND ELLIPSE ENTITIES

If the former possibilities to be a line or an arc are rejected, we must check for a circle or an ellipse approximation. When this happens, an algorithm based on following contour techniques is applied, extracting several features to decide whether to be a circle or an ellipse and to compute the parameters of the shape (the stroke piece). These features are:

- The centroid of the shape *C* .
- The orientation of the shape α (useful for ellipses), that will be the orientation of maximum diameter.
- The maximum diameter (radius *R*) with α orientation and minimum diameter (radius *r*) passing through the centroid *C*.
- Elongation (maximum diameter divided by minimum diameter).
- The intersection point *I* of the stroke with the maximum diameter (useful for ellipses).

Once all these features have been calculated, we base our decision on the entity type in the *elongation* feature. This feature results of dividing the two calculated radius. When elongation remains between a threshold interval (set empirically to 0,8 <= elongation <= 1,2), the entity will be interpreted as a circle, otherwise as an ellipse. In Figure 9 results obtained for a shape approximated by an oriented ellipse are shown.

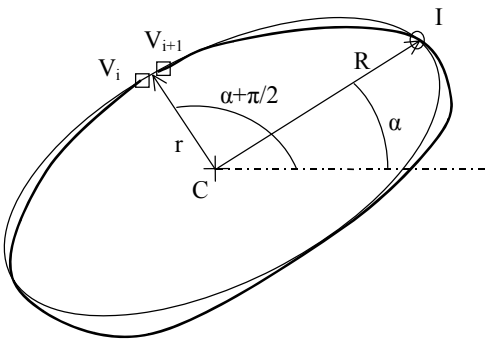


Figure 9: *Parameters of a circle/ellipse primitive*

4.5 Evaluation and discussion

Our recognizer takes a single stroke as input. The samples used to evaluate the geometric recognizer were shapes composed of solely lines, solely arcs, and lines and arcs as shown in figure 10, where the dots represent the stroke points sampled on a Tablet PC, and the little squares are the vertexes found after recognition. For approximated lines, a black straight line are painted between two vertexes and, for arcs, the centre is marked with a cross sign and the three radius are painted from centre to the two endpoints of the arc, and to a third point on arc placed near the middle

Results in figure 10 show approximated polylines that suit the sketchy form pretty well. However, there are examples in which recognition does not match the sketch as it was intended. For instance, the sketchy form of figure 10(a) approximates two arcs instead of just one arc. This is because a vertex has been found and, although both radiuses are quite similar, they are not concentric. Also, when changes in stroke direction are not sharp enough the results are not the intended primitives.

We can appreciate that when arcs are sketched not convex enough, they are approximated by more than one arc, because vertex detection finds significant peaks in curvature as shows figure 11(a, b). Also different parts of an intended arc can be approximated to a line and an arc, as shows figure 11(b). Notice the recognition result for figure 11(f) corresponding to a number "2". The difference is that in the left shape, the upper part is more convex, and the softness in sketching makes that just one vertex is found, so just one arc is approximated.

Better results can be obtained by improving the beautification of the sketch, which will rebound in the vertex detection algorithm. A final refinement can also be carried out to remove extremely short approximated entities (lines or arcs) checking neighbour entities.

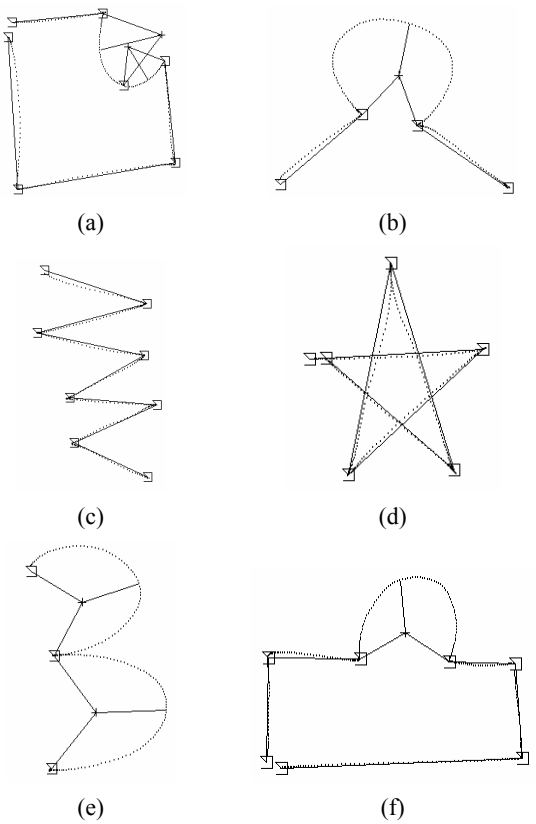


Figure 10: *Sample of shapes used for evaluation*

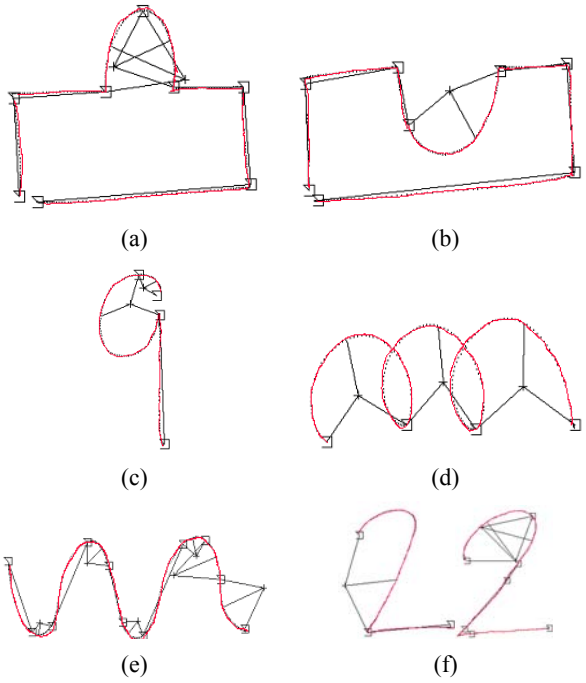


Figure 11: *Other samples for evaluation. Recognition results are in black and the sketch in red.*

5. Gesture recognizer

A gesture recognizer has been designed for interpreting gestures sketched by the user as dimensional-geometric constrains and commands. The application distinguishes gestures from geometry, analyzing the pressure that the user makes with the stylus on the LCD screen.

The algorithm consists of four steps. In the first step, we use pre-processing image analysis techniques to smooth and remove noise from the sketched gesture. In the second step, the size of the sketched gesture is normalized to provide the same concentration of digitized points throughout the gesture. Then, two signatures of the shape of the gesture are calculated: 1) as the distance of each point from the gesture centroid and 2) as changes in the direction of consecutive points. First signature is called the *polar signature*, and second one is called the *direction signature*. In the third step, we apply the fast Fourier transform (FFT) to the spectrum domain of the previous two signatures. Fourier descriptors are used due to their properties of being scale-rotation invariant. As a final step, a standard non-linear discriminant analysis is used to classify the gestures by attending to Fourier descriptors of digital signatures.

Several gestures of the alphabet used in our sketching application are shown in table 1. These gestures can be composed of several strokes, where each stroke is a set of points that are digitized by the device between pen-down and pen-up events. The last stroke for a gesture occurs when a timeout is reached from the last pen-up event. An example of multistroke gestures is presented in Figure 12 where for the diametral dimension and the parallel constrain are presented with their corresponding signatures.

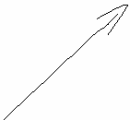
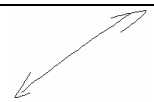
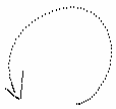
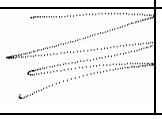
Constraint gestures	Class	Command gestures	Class
	Concentric		Extrusion
	Dimension		Revolve-right
	Diametral dimension		Revolve-left
	Parallel		Cross-out (erase)

Table 1. *Some supported gestures*

Note that the size of any part of the stroke has equal length. This is due to the normalization of each stroke of the gesture. This beautification of the sketch contributes to make the method used to recognize gestures to be scale invariant.

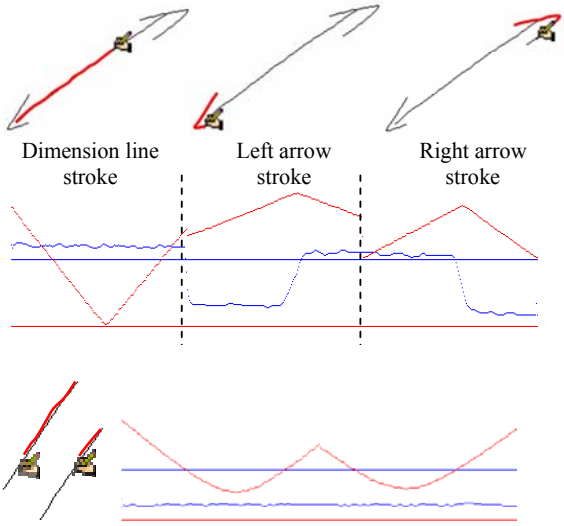


Figure 12: *Diametral dimension and parallel gestures with its polar (red) and direction (blue) signatures.*

The FFT of the normalised polar and direction signatures are calculated independently, using the resulting harmonics as variables of the classification function of the discriminant analysis. The averaged success ratio for the classification was 91.8%, which is a good result in an on-line application. When the result of classification has a probability less than a threshold (as the level of confidence) the user is asked to sketch the gesture again, what increases the success ratio to an average of 95%.

6. Conclusions

A robust algorithm to approximate hand sketches into its constituent primitives has been developed for an on-line system. Different tests demonstrated that the obtained outlined sketch is practically the intended geometry by the designer. Future improvements are expected from applying a previous smoothing of the sketch by morphological operations in order to optimize the vertexes search or a refinement of the final outlined sketch in order to simplify geometry.

A new algorithm to recognize multistroke gestures has been proposed. It is based on the shape description of the gestures using polar and direction signatures, analyzed by means of the fast Fourier transform (with rotation-scale invariant properties), whose descriptors were used as independent variables to perform the classification using non-linear discriminate analysis. Classification functions obtained were used during the process off-line to train the system. Such characteristic will allow extending the supported gesture alphabet or tailoring the recognizer behavior to a specific user's way of sketching. The averaged success ratio obtained from tests with different CAD and non CAD users was 91.8%, what is a good result in an application on-line, and increases to an average of nearly 95% when a level of confidence is used.

Both recognizers work pretty well in a sketch-based environment, and results seem to be relevant in the context we are working. The ongoing work presented in this paper demonstrates that both methods work well in an on-line application, solving major problems inherent to sketch drawing and allowing the operation in real time.

Acknowledgments

This work was partially funded by Fundació Caixa Castelló-Bancaixa through project P1-1B2004-02, "Gestual Interface for Parametric-variational Sketches, and for Definition of Assembly Conditions in Aided Design of Manufactured Products". Besides, the Spanish Ministry of Science and Education and the European Union, through the SAMBODIC project (Ref. DPI2004-01373) partially supported this work

References

[Van05] VAN DER LUGT R.: How sketching can affect the idea generation process in design group meetings. *Design Studies 26*, 2 (2005), 101–122.

[Fer92] FERGUSON E.S.: *Engineering and the Mind's Eye.* MIT Press, 1992

[CNJC03] CONTERO M., NAYA F. , JORGE J., CONESA J.: CIGRO: a minimal instruc-tion set calligraphic interface for sketch-based modeling. *Lecture Notes in Computer Science*. *2669* (2003), 549–558.

[ZHH96] ZELEZNIK, R.C., HERNDON, K.P., HUGHES, J. F.: SKETCH: an interface for sketching 3D scenes. In *Proc. SIGGRAPH '96* (1996), pp. 163–170.

[Gre91] GREENBERG S.: GroupSketch: a multi-user sketchpad for geographically-distributed small groups. In *Proc. Graphics Interface '91* (1991), pp. 207–215.

[Yu03] YU, B.: Recognition of freehand sketches using mean shift. In *Proc. of IUI '03*. (2003), pp. 204–210.

[QWJ01] QIN S-F., WRIGHT D.K., JORDANOV I.N.: On-line segmentation of freehand sketches by knowledge-based nonlinear thresholding operations. *Pattern Recognition 34* (2001) 1885–1893.

[Rub91] RUBINE D.: Specifying Gestures by Example. In *Proc. Of SIGGRAPH '91* (1991), pp. 329–337.

[AK] APTE A., KIMURA V.V.: Recognizing multistroke geometric shapes: an experimental evaluation. In *Proc. of ACM UIST'93,* (1993), pp. 121–128.

[CSKK02] CALHOUN C., STAHOVICH T.F., KURTOGLU T., KARA L.B.: Recognizing Multi-Stroke Symbols. In *Proc. of AAAI Spring Symposium on Sketch Understanding* (2002) pp.s 15–23.

[PBJS*04] PEREIRA J. P., BRANCO V. A., JORGE J. A., SILVA N. F., CARDOSO T. D., FERREIRA F. N.: Cascading recognizers for ambiguous calligraphic interaction. In *Proc. of the Eurographics Workshop on Sketch-Based Modeling (SBM'04)* (2004), pp. 63–72.

[ZL02] ZANG D., LU G.: A comparative study of fourier descriptors for shape representation and retrieval. In *Proc. of the 5th Asian Conference on Computer Vision, Melbourne, Australia,* ACCV'02 (2002) pp. 1–6.

[HE04] HARDING P.R.G., ELLIS T.J.: Recognizing hand gesture using fourier descriptors. In *Proceedings of the 17th International Conference on Pattern Recognition* (2004), vol. 3, pp. 286–289.

[LS04] LICSAR A., SZIRANYI T.: Hand gesture recognition in camera-projector system. *Lecture Notes in Computer Science 3058* (2004), 83–93.

[HA04] HOPKINS J., ANDERSEN T.: A Fourier descriptor based character recognition engine implemented under the Gamera open-source document processing framework. In *Proceedings of the SPIE* (2004) vol. 5676, pp. 111–118.

[PP05] PARK C.H., PARK H.: Fingerprint classification using fast Fourier transform and non-linear discriminant analysis, *Pattern Recognition 38* (2005) 495–503.

[HLLM02] HONG J., LANDAY J., LONG A.C., MANKOFF J.: Sketch recognizers from the end-user's, the designer's, and the programmer's perspective. *Sketch Understanding, Papers from the 2002 AAAI Spring Symposium.* (2002) pp. 73–77.