Classification of Sketch Strokes and Corner Detection using Conic Sections and Adaptive Clustering

M. Shpitalni, H. Lipson[1]

This paper presents a method for classifying pen strokes in an on-line sketching system. The method, based on linear least squares fitting to a conic section equation, proposes using the conic equation's natural classification property to help classify sketch strokes and identify lines, elliptic arcs, and corners composed of two lines with an optional fillet. The hyperbola form of the conic equation is used for corner detection. The proposed method has proven to be fast, suitable for real-time classification, and capable of tolerating noisy input, including cusps and spikes. The classification is obtained in o(n) time in a single path, where n is the number of sampled points. In addition, an improved adaptive method for clustering disconnected endpoints is proposed. The notion of in-context analysis is discussed, and examples from a working implementation are given.

Introduction

Freehand sketching is occupying a growing place in the realm of user-interface approaches for CAD systems. Sketching appears to be a natural communication language, enabling faster conveyance of qualitative information while not burdening the creativity of the user or disrupting the flow of ideas. Recent studies [1, 2] emphasize the importance of sketching in the mechanical design process, especially in the conceptual design stage. Several interfaces based on sketching have therefore been proposed, with stroke classification and clustering playing a major role.

User interfaces based on on-line sketching are implemented on several levels and can generally be divided into three categories, according to the level of information they intend to gather from the input sketch:

1. **Drawing Pads.** These sketchers allow basic sketching for general purpose drawings, especially in the graphic design arts. They smooth the input strokes and provide many other graphic tools but do not attempt to interpret the drawing in any way.
2. **2D sketchers.** In 2D sketchers, sketch strokes are smoothed and classified into 2D primitives, such as lines, arcs and splines. Some automatically infer constraints and relationships among the entities, such as parallelism or orthogonality, thus further refining the sketch [2,3,4,5,6].
3. **3D sketchers**. Sketches are analyzed as representing rough projections of 3D scenes. The sketcher is still required to identify the sketch strokes as basic geometrical shapes, such as lines, arcs and corners. However, since the analyzed sketch represents a rough projection of a three-dimensional scene, some of the sketch strokes do not necessarily represent what they appear to be. For instance, a circular arc in a three dimensional scene is most likely to appear as an ellipse in a projection. In addition, crossing curves in the sketch do not necessarily represent curves that actually meet. Systems for interpreting sketches as 3D scenes must confront greater problems and are less common [7,8,9].

This paper presents the basic operation of a sketcher from the third category that is used to preprocess a rough sketch representing a projection of a 3D object. The sketcher is the front-end of a 3D scene interpreter, intended to serve as a natural user-interface for 3D CAD applications. Although the sketcher is designed to make automatic decisions, the interactive environment allows the user to modify or re-sketch erroneous interpretations. Two stages can be observed in the functioning of the sketcher: (a) classification and smoothing of sketch strokes as they are drawn, and (b) linking the entities at their meeting endpoints to form a connected graph (clustering). Both stages require robust methods for analyzing the rough sketch data and overcoming ambiguities inherent in 2D drawings representing 3D scenes.

[1]Laboratory for Computer Graphics and CAD Faculty of Mechanical Engineering, Technion, Haifa 32000, Israel

Related Work

Many methods have been used to analyze line drawings and convert image data into more meaningful geometrical information [10]. However, analysis of image data is fundamentally different from analysis of on-line sketching strokes. Strokes have been classified into entity types using curvature analysis [2], inspection of local angles at points along the stroke [3], correlation with predefined templates [11], and artificial neural nets [12]. With some sketchers, the user can or must explicitly indicate which type of entity was intended [6]. Classification must be distinguished from the pure smoothing employed in applications in the "sketch-pad" category. Smoothing is typically accomplished by averaging, convoluting, or fitting to Bezier, B-Spline and conic curves [13]. While conic section fitting has been used before both for smoothing and modeling of scattered data, here we are suggesting use of the natural *classification* properties of the conic section to classify noisy sketch strokes.

In the classification process, corners are often detected by searching for sharp curvature peaks or angle changes [2,3]. Note, however, that curvature analysis is extremely susceptible to cusps, spikes and wobbles in the strokes which are common in rough sketches even after smoothing.

In addition to classification and smoothing, most sketchers link strokes at their endpoints, crossings and junctions. This phase is difficult when rough and inaccurate sketches are considered. Moreover, it poses a major hurdle when the sketch depicts a 3D scene in which endpoints that appear to be close in the sketch plane can actually correspond to endpoints that are far apart spatially. Linking has been performed using sharp distance tolerances [3], tolerance as a percentage of length of meeting entities [5], or a clustering scheme based on a threshold of the maximum interval in a group of elements [4]. In the last case, the threshold was empirically chosen according to the type of drawing and statistics.

Stroke Classification Using Conic-Curve Fitting

In the classification stage, the sketcher is required to accept sketch strokes and classify them into various geometrical entities. A stroke is defined as the path marked by a pen between the pen-down and the pen-up operations. Each stroke is assumed to correspond to a single entity (line/arc), or a sharp/filleted corner**.** In the specific case considered here, the sketcher is part of a system aimed at interpreting a drawing depicting a 3D object [7,14,15]. As asserted in [16], line drawings of manmade objects often exhibit instances of straight lines, circular arcs and ellipses, all of which are conic sections. In addition, corners are important features of line drawings. A corner closely resembles a hyperbola, which fortunately is also a conic section. The use of a hyperbola also permits detection of filleted corners. Consequently, it is assumed that the majority of entities in the sketch will indeed be either lines, elliptic arcs or corners, and we focus on their detection. Other curve types (e.g. parabolas) are not considered by our sketcher on the assumption that they are special cases and may be entered directly into the CAD system for which this sketcher is a front end.

A sketch stroke is classified by fitting it to a conic section and analyzing its coefficients. A general conic section in the x-y plane is given by:

$$Q(x,y) = Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0 \quad (1)$$

In practice, it can be assumed that F≠0; the equation can therefore be normalized with F=1. A detailed description of least-squares fitting to conic sections can be found in [13]. Briefly, a sketch stroke is denoted by $L = \{(x_1,y_1),(x_2,y_2),\ldots,(x_N,y_N)\}$. After defining an error parameter and differentiating partially with respect to the conic coefficients, the following linear system is obtained.

$$\sum_{i=1}^{N} \begin{vmatrix} x_i^4 & x_i^3 y_i & x_i^2 y_i^2 & x_i^3 & x_i^2 y_i \\ x_i^3 y_i & x_i^2 y_i^2 & x_i y_i^3 & x_i^2 y_i & x_i y_i^2 \\ x_i^2 y_i^2 & x_i y_i^3 & y_i^4 & x_i y_i^2 & y_i^3 \\ x_i^3 & x_i^2 y_i & x_i y_i^2 & x_i^2 & x_i y_i \\ x_i^2 y_i & x_i y_i^2 & y_i^3 & x_i y_i & y_i^2 \end{vmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix} + \sum_{i=1}^{N} \begin{bmatrix} x_i^2 \\ x_i y_i \\ y_i^2 \\ x_i \\ y_i \end{bmatrix} = 0 \quad (2)$$

Once the coefficients *A*...*F* have been solved, some conclusions regarding the conic section may be derived using the following definition. Denote:

$$\delta \equiv \begin{vmatrix} A & \frac{B}{2} \\ \frac{B}{2} & C \end{vmatrix} \quad \text{and} \quad \Delta \equiv \begin{vmatrix} A & \frac{B}{2} & \frac{D}{2} \\ \frac{B}{2} & C & \frac{E}{2} \\ \frac{D}{2} & \frac{E}{2} & F \end{vmatrix} \quad (3)$$

then, if δ>0, the section represents an ellipse, and if δ<0, a hyperbola. For both hyperbolas and ellipses, translation and orientation with respect to their canonical position can easily be determined,

$$x = \frac{BE - 2CD}{4AC - B^2} \qquad y = \frac{DB - 2AE}{4AC - B^2} \quad (4)$$

$$\theta = \frac{\tan^{-1}\frac{B}{A-C}}{2} \quad (5)$$

The length of their main axes can also be calculated:

$$a,b = \sqrt{\frac{|\Delta|}{|\delta|} \cdot \frac{2}{(A+C) \pm \sqrt{(A-C)^2 + B^2}}} \quad (6)$$

where the distinction between *a* and *b* is made according to the sign of *B*.

Classification is based on the specific form of the conic section; the parameters of the form define the geometrical dimensions of the shape and enable smoothing and decomposition of strokes at corners. In the rare case of obtaining a fit to a parabola, a slight perturbation of any stroke point can be used to avoid the singularity. Exact lines are also a rare form of a conic curve, occurring only in some cases where D=d=0. A linear segment will typically be fitted with a very narrow ellipse or hyperbola. Since detection of linear segments is crucial, any elliptical or hyperbolic fit with an aspect ratio (minor to major axis ratio) less than 1:20 is assumed to be linear. This ratio is heuristic and was chosen according to observations. A hyperbola fit must be further broken down into a set of two lines with a possible elliptic fillet between them. Such a fit can be obtained by scaling the hyperbola so that *a*=*b*=1. Then, an ordinary circular fillet is calculated and transformed back into the original plane.

The linear solution and classification forms are most suitable for fast on-line interpretation. Whenever a new stroke coordinate pair is generated, the matrix sums are updated and the set of equations re-solved for *A*...*F*. The time complexity of the classification process at a given point is not a function of the number of points acquired so far; the process may be executed in *O(1)* time. This fact allows continuous updating of the conic fit at the cost of inverting a 5x5 matrix, and the display may be updated as the user draws the stroke. This continuous feedback, beneficial for spotting potential misinterpretations in advance, is one of the advantages of an on-line system. The time complexity of classifying a given stroke of points is o(n) and requires a single pass.

Figure 1 shows some examples of conic sections (dashed) fitted to strokes (noisy solid).

Entity Linking and Endpoint Clustering

In order to process the classified entities as more meaningful data (in this case, as a projection of a 3D object), they must be linked at connection points. However, an examination of sketches reveals that users tend to place stroke endpoints inaccurately. A simple approach of joining endpoints that are closer than a minimal threshold distance will not suffice; a threshold that is too large may eliminate fine details, and a threshold that is too small may leave adjacent endpoints unlinked. Different tolerances may be necessary for different parts of the sketch, and certainly for different sketches made by different users.

To overcome these variations, an adaptive clustering approach is used. In essence, the method is based on computing tolerance zones around each endpoint in the drawing, where the size of the zone corresponds to the uncertainty in the endpoint position. When the size of the gap between two endpoints is less than the expected error in placement of both the endpoints, it is likely that the endpoints were meant to coincide. Based on this reasoning, endpoint pairs are clustered when each member of the pair falls within the uncertainty zone of the other member. First, a list of potential connection points, termed *raw vertices*, is created. These points are placed at the endpoints of all the entities in the sketch. A specific tolerance is computed for each *raw vertex*, according to its neighboring geometry and other parameters, as discussed below. Then, every *raw vertex* is grouped with any neighboring *raw vertices* when both endpoints fall within each other's tolerance zone. The procedure is repeated as more and more endpoints are grouped. When two endpoints belonging to different groups are clustered, their associated groups are united. This procedure results in clusters of *raw vertices*. Each cluster will finally be represented by one vertex whose coordinates are at the average of centers.

The critical phase of this algorithm is determining the size (radius, assuming the zones are circular) of the tolerance zone for each raw vertex. The size corresponds to the uncertainty in the vertex position. If the size is set to some (perhaps arbitrary) value, the result reverts to that of simple distance threshold linking. A tolerance threshold chosen according to a statistical analysis may result in the clustering obtained in [4]. If a tolerance circle is set to a percentage of the attached entity length, the result reverts to that suggested in [5].

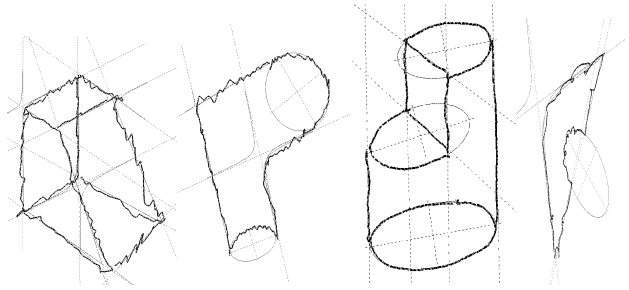

Fig. 1: Some examples of conic fitting to strokes.

It seems that a uniform tolerance or one based on a percentage of the associated line length cannot be used to obtain the most reliable result. The size of the tolerance zone should also be sensitive to the magnitude of the detail in the

close vicinity of the endpoint. To achieve this sensitivity, the average distance from the endpoint to each entity in the sketch is measured. This includes the average distance from the endpoint to its own entity, namely, half the entity length. The smallest of these averages is taken as the size of the tolerance circle. Clearly, when small and fine detail is in the close vicinity of the endpoint, the resulting tolerance circle will be appropriately small. The value obtained will be limited by some arbitrary upper bound. In addition, a tolerance circle of an endpoint can never, by definition, contain both endpoints of an entity and therefore will never obscure even the finest detail. The tolerance circles shown in Fig. 2 (a) were derived using this method.

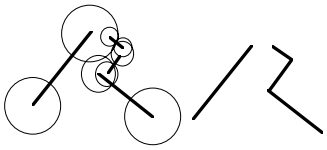

Fig 2: Raw vertices with tolerance circles and resulting clustering.

Alternative or additional criteria for determining the tolerance circle size may be considered as well, for instance, one based on pen dynamics and button-release timing.

The size of the tolerance circle can also be influenced by criteria specific to the application in which the sketcher is used. In this work, the sketch is assumed to depict a 3D object. As is often the case in trihedral manifold objects, each endpoint is connected to three edges. That is, endpoints tend to be clustered in groups of three. Thus, clustering may be biased towards such grouping; the tolerance of clusters containing multiples of three endpoints are perturbed downwards in an attempt to cause the cluster to split into groups of three. Similarly, 3D objects tend not to contain unconnected endpoints; the clustering mechanism is therefore biased against such cases by slightly enlarging the size of such disconnected single vertices. An example of a connected graph processed by the above procedure is given in Fig. 3.

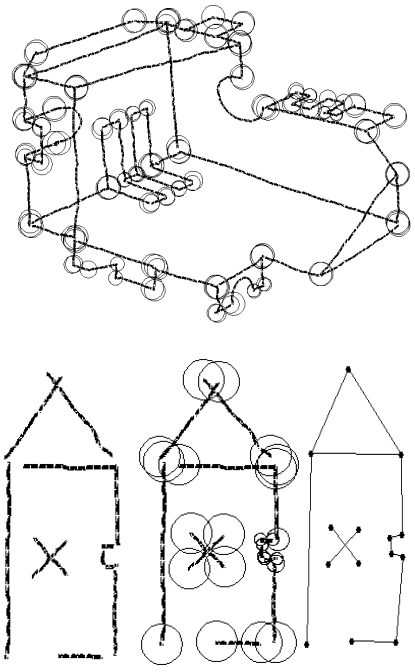

Fig 3: Tolerance circles obtained using clustering procedure and resulting linking.

To conclude, the following steps are applied by the clustering algorithm

- *Create raw vertices at all endpoints of entities in the drawing.*
- *Determine the radius of the tolerance circle around each raw vertex.*
- *Identify and group pairs of raw vertices when each member of the pair falls within the other member's tolerance circle.*
- *Iteratively group chains of pairs into clusters.*
- *Place a vertex node at the centroid of each cluster.*
- *Adjust lines and arcs accordingly.*

In-Context Classification and Clustering

The classification and clustering methods described above and those appearing in the references all attempt to classify entities using explicit geometrical information appearing in the sketch. However, examples may be constructed where the correct decision can only be made when the geometry is considered in the *global context* of the drawing. Fig. 4 illustrates two such examples. In Fig. 4 (a), an encircled junction of six lines appears in the two pictures. Although the two junctions are *geometrically identical*, they have different meanings when viewed in the drawing context: on the left, the junction represents two separate corners which accidentally coincide in the projection plane; on the right, it represents an orthogonal junction of six lines. Similarly, the classification of the horizontal entity illustrated in Fig. 4 (b) depends on the context in which it appears. On the left, it represents an arc, and on the right, it represents a straight line.

It is evident that in order to differentiate between these cases, the meaning of the sketch must be "understood." If the sketcher is part of a larger mechanism that also interprets the sketch as a whole, then making the decision *in context* is possible.

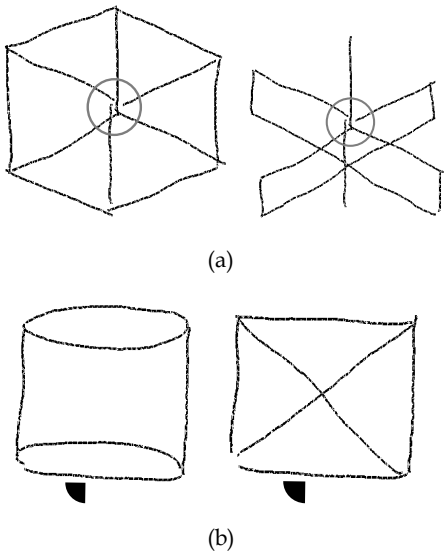

Fig. 4: (a) The same line junction appears in the two drawings, but with different meanings, (b) The same stroke appears at the bottom of each drawing, on the left as an arc and on the right as a line.

Conclusions

This paper presents a method for classifying sketch strokes acquired from on-line sketching systems. The classification is performed using conic-curve fitting and is capable of recognizing lines, arcs, elliptic arcs, sharp corners and filleted corners. The method has two basic advantages: (a) It can perform robust classification of rough input including spikes and cusps, which are troublesome for curvature-based classification, and (b) it has a short execution time that is not dependent on the length of the stroke or the number of

sample points acquired (assuming coordinates have been summed while drawing the stroke). The procedure can therefore be used to provide continuous feedback of the interpreted entity during drawing, in real time. However, in spite of this ability, it is evident that geometrical-based classification is inherently limited and a more general, context-sensitive approach must be pursued.

A new endpoint clustering scheme has also been presented based on adaptive tolerances at different parts of the sketch. The proposed formulation provides a framework for implementing various criteria for determining local thresholds, such as detail sensitive criteria, dynamic criteria, or other application specific criteria. Again, clustering can be improved using a context-sensitive approach.

Acknowledgments

This research has been supported in part by the Fund for the Promotion of Research at the Technion, Research No. 033-028.

References

1. Ullman, D. G., Wood, S., and Craig, D., 1990, "The importance of drawing in the mechanical design process," *Computers & Graphics*, Vol 14, No. 2, pp. 263-274.
2. Jenkins, D. L. and Martin, R. R., 1992, "Applying constraints to enforce user's intentions in free-hand 2-D sketches," *Intelligent Systems Engineering*, Autumn, pp. 31-49.
3. Kato, O., Iwase, H., Yoshida, M., and Tanahshi, J., 1982, "Interactive Hand-Drawn Diagram Input System," *Proc. IEEE Conference on Pattern Recognition and Image Processing (PRIP 82)*, Las Vegas, Nevada, pp. 544-549.
4. Pavlidis, T., and Van Wyk, C. J., 1985, "An Automatic Beautifier for Drawings and Illustrations," *SIGGRAPH 85*, Vol 19, No. 3, pp. 225-234.
5. Bengi, F. and Ozguc, B., 1990, "Architectural Sketch Recognition," *Architectural Science Review*, Vol. 33, pp. 3-16.
6. Eggli, L., Brüderlin, B. P., and Elber, G., 1995, "Sketching as a solid modeler tool," Third Symposium on Solid Modeling and Applications, *ACM SIGGRAPH*, pp. 313-321.
7. Lipson, H. and Shpitalni, M., 1995, "A new interface for conceptual design based on object reconstruction from a single freehand sketch," *Annals of the CIRP*, Vol 44/1, pp. 133-136.
8. Lamb, D. and Bandopadhay, A., 1990, "Interpreting a 3D object from a rough 2D line drawing," *Proc First IEEE Conf on Visualization*, 90: 59-66.
9. Marti, E., Regomc—s, J., L—pez-Krahe, J., and Villanueva, J.J., 1993, "Hand line drawing interpretation as three dimensional object," *Signal Processing*, 32: 91-110.
10. Smith, R. W., 1987, "Computer Processing of line images: A Survey," *Pattern Recognition*, Vol 20, No 1, pp 7-15.
11. Spur, G., and Jansen, H., 1984, "Automatic recognition of hand-drawn contours for CAD applications," 16th CIRP Int. Seminar on Manufacturing Systems, 63-72.
12. Koo, J. C. and Fernandez, B., 1993, "Geometrical Error Correction Using Hierarchical/ Hybrid Artificial Neural Systems," *IEEE International Conference on Neural Networks*, pp. 232-237.
13. Bookstein, F. L., 1979, "Fitting Conic Sections to Scattered Data," *Computer Graphics and Image Processing*, 9, pp. 56-71.
14. Shpitalni, M and Lipson, H, 1995, "Identification of faces in a 2D line drawing projection of a wireframe object" to appear in *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
15. Lipson, H. and Shpitalni, M., 1996 "Optimization-Based Reconstruction of a 3D Object From a Single Freehand Line Drawing," *Journal of Computer Aided Design*, Vol. 28 No 8, 651-663.
16. Nalwa, V. S., 1988, "Line-Drawing Interpretation: Straight Lines and Conic Sections," *IEEE Pattern Analysis and Machine Intelligence*, Vol 10., No. 4.
17. Faux, I. D. and Pratt, M. J., 1981, *Computational Geometry for Design and Manufacture*, John Wiley