EUROGRAPHICS Workshop on Sketch-Based Interface and Modeling (2007)
M. van de Panne, E. Saund (Editors)

Implicit Geometric Constraint Detection in Freehand Sketches Using Relative Shape Histogram

J. Pu and K. Ramani

Purdue Research and Education Center for Information Systems in Engineering (PRECISE), School of Mechanical Engineering, School of Electrical Engineering, Purdue University, West Lafayette IN 47907-2024, USA

Abstract

In order to take advantage of the sketch-based interaction, many methods have been proposed to beautify freehand sketches. Most of these efforts are dedicated to sketch segmentation and recognition, while some important information implied in the sketches, such as geometric constraints, are largely ignored. Thus, the final beautified results by these methods do not fully reflect the true intentions from users. In this paper, a statistical approach called Relative Shape Histogram (RSH) is introduced to detect the implied geometric constraint in sketches. The basic idea arises from such a discovery that the same geometric constraints between two geometric primitives have similar relative shape histograms. By computing the similarity between RSHs, the implicit geometric constraints between two segmented primitives are inferred. To evaluate the performance of the proposed algorithm, a user-based experiment is conducted and the results are presented in this paper.

Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Interaction techniques

1. Introduction

Freehand sketches have been widely recognized as an efficient and natural way to communicate ideas between human being and computer. However, correct interpretation of the intent of the users making freehand sketches, i.e., sketch understanding, is still a challenge because freehand sketches are informal, ambiguous, and implicit in comparison to traditional WIMP (Window, Icon, Menu, and Pointer) user interfaces in which each interaction is predefined and there is unique correspondence between an input and the internal interpretation of a computer [Li03]. To address this issue and enable a computer to interpret freehand sketches in a unified and robust way, many sketch beautification methods have been proposed to transform freehand sketches from an informal representation to a formal representation. Most of these efforts focused on two important issues: sketch segmentation [FLK*04, Stahovich04, LS05, NM04, GKS05] and primitive (or composite) recognition [SD05, HN04, KS04, AD04], which are regarded as the two main obstacles that have hindered the development of a system with superior sketch understanding [FLK*04]. Sketch segmentation is the process by which a continuous stream of pen strokes is parsed into a series of constituent geometric primitives that are atomic geometric entities in sketches and can not be further decomposed. Frequently, only lines, circles and arcs are considered as the basic primitives that constitute a user's sketches [FLK*04, Stahovich04, HN04] . Given a segmented portion of a pen stroke, the aim of primitive or composite recognition is to determine the type of geometric entity to which it belongs.

Often, once the primitives are recognized, the freehand sketches can be beautified by assigning primitives with proper parameters. However, such a direct beautification often misses some important information implied in sketches such as geometric constraints, which are widely used in many design related applications, such as drawing programs [KB90], computer aided design [Aldefeld98, BFH*95, FH97] and graphical user interfaces [BD86], to determine the relationship between two objects. To illustrate the disadvantage of direct beautification, two examples are shown in Figure 1, where (b) is the beautified result of (a) and (d) is the beautified result of (c).

It is intuitive for users to conclude from the sketches that the two circles have the same center while the lines are

Sketch-Based Interfaces and Modeling 2007, Riverside, CA, August 02-03, 2007.
© 2007 ACM 978-1-59593-913-5/07/0008 $5.00

parallel. However, due to the fact that the parameters (i.e., center and radius) of each circle are determined separately, their relative relationship such as geometric constraint might change after the beautification operation, thus leading to the conclusion that there is no relationship between the two circles and the lines. In a geometric constraint system, this direct beautification will lead to serious error propagation for which additional efforts [WSH05] are needed. Therefore, a more reliable method is to check their constraint type directly on the basis of the original freehand sketches instead of the final beautified results.

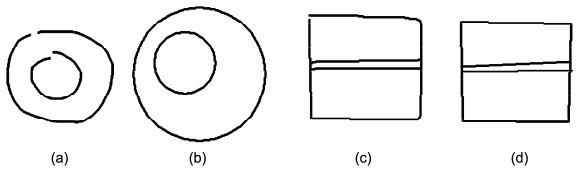

Figure 1: *Implied constraints might be different before and after beautification: (b) is the beautified result of freehand sketch (a) and (d) is the beautified result of freehand sketch (c).*

In contrast, an example is shown in Figure 2 to demonstrate the potential advantages of sketch beautification driven by implied constraints, where both explicit and implicit constraints are considered.

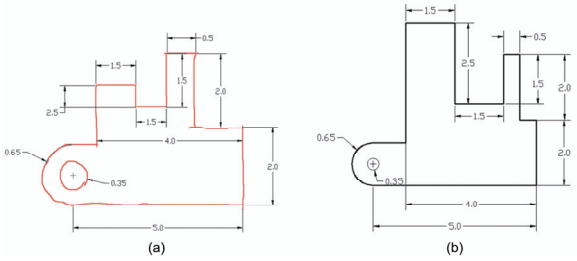

Figure 2: *A sketch parameterization example: (a) shows a sketched drawing with geometric constraints; and (b) is the parameterized result of (a) under the specified constraints.*

Explicit constraints refer to constraints that a user specifies, such as dimension, while implicit constraints are those that are implied in the sketches, such as parallelism and perpendicularity. From the above two examples, it is not difficult to infer that sketch beautification driven by geometric constraints can not only achieve designers' ideas more accurately and naturally than direct beautification but also be capable of bridging the gap between the initial conceptual design and the final detailed design. To implement such a sketch beautification paradigm, a crucial step is to detect the geometric constraints implied in freehand sketches. In the past, the detection of implied constraints in freehand sketches is largely ignored. Most of similar efforts [BS86] are dedicated to the detection of implied constraints in already beautified representations. [KWL93] is a good source for the related work in detail. Due to the informal representation of freehand sketches, it is impossible to apply these methods to the detection of implied constraints in freehand sketches.

In this paper, an algorithm is proposed to detect the geometric constraints implied in sketches with a statistical method called relative shape histogram (RSH). The remainder of the content is organized as follows. In Section 2~3, the concepts about implicit geometric constraint and the way of freehand sketch representation are explained respectively. In Section 4, the shape histogram is introduced, on the basis of which RSH is derived and the method to compute the similarity between RSHs is presented in Section 5. To evaluate the performance of these proposed algorithms, a preliminary user study is conducted in Section 6. Additionally, examples are given to demonstrate the validity of this proposed algorithm when it is combined with a geometric constraint solver to beautify freehand sketches. The geometric constraint solving will be not discussed due to the fact that it is beyond the scope of this paper and has been studied extensively in the past [Aldefeld98, BFH*95, FH97].

2. Implicit Geometric Constraint

Geometric constraints are widely used in a design process and usually can be classified as one of two types: either explicit constraints, which refer to constraints that a user specifies explicitly, such as dimension, or implicit constraints, which are those that are implied in the sketches, such as parallelism and perpendicularity. It is natural for users to express geometric constraints implicitly when they are sketching their design idea on a piece of paper. Although the implicit constraint can provide flexibility for users, its informal representation often leads to inconsistency because it is still difficult to detect them in a robust way.

Table 1: *Examples of geometric constraints between primitives.*

Popular Constraint Types between Points, Lines, and Circles					
					
(1)P-P-D	(2) P-C-D	(3) L-C-D	(4) L-L-D	(5)P-L-D	(6) L-L-A
					
(7) C-C-D	(8)L-L-CL	(9)L-L-PE	(10)L-L-PA	(11) P-L-O	(12) P-L-E
					
(13) P-L-S	(14)L-C-TC	(15)C-C-CC	(16)C-C-IT	(17) C-C-ET	(18) P-C-O
Note: "X-Y-Z": X or Y represent Points, Lines, or Circles; Z represents the type of constraint between X and Y, e.g., D stands for Distance, O for On, E for Ending point, S for Starting Point, CL for Co-Line, PE for Perpendicular, PA for Parallel, A for Angle, TC for Through Center, CC for Co-Centric, IT for Internal Tangent, ET for External Tangent.					

In Table 1, some common constraints between primitives are listed, in which (1)~(7) are examples with explicit constraints and (8)~(18) are examples with implicit constraint. In practice, users can extend this constraint list according to their needs.

3. Sketch Representation

Freehand sketches are usually composed of a series of basic geometric entities such as lines, circles, and arcs. When a user transmits his or her ideas to a computer using a pen, it is not practical to assume that each stroke only represents a single geometric primitive. On the contrary, a stroke may consist of multiple line segments and arcs. To recognize the sketches, a segmentation process is needed to parse and recognize individual primitive shapes from a user's stroke streams.

Generally, the sketches are drawn offline or online. In the case of offline sketches, the sketches consist of bitmap-like pixels. In contrast with the offline sketches, during the online sketching process, the track of a stroke such as S is usually composed of a sequence of small line segments rather than image bitmaps:

$$S = \{((x_i, y_i), (x_{i+1}, y_{i+1}), t_i) \mid 0 \le i \le n\} \qquad (1)$$

where n is the total number of line segments included in a single stroke S, (x_i, y_i) and (x_{i+1}, y_{i+1}) are the two ending points of a small line segment at time t_i. Correspondingly, the sketches of an object A are usually achieved by a sequence of strokes:

$$A = \{S_i \mid 0 \le i < m\} \qquad (2)$$

where m is the number of the strokes. The goal of sketch segmentation (ink parsing) is to define all of the segment points that parse the stroke stream into a sequence of geometric primitives, such as lines, circles, and arcs. It has been reported that the time plays an important role in this parsing process [Stahovich04]. In this paper, we assume that the sketches have already been segmented since sketch segmentation is a non-trivial problem and is out of the scope of this paper.

4. Two-dimensional Shape Histogram

Osada et al. [OFC*02] represented a three dimensional object as a shape distribution in order to measure the similarity between three dimensional models. The shape distribution is formed by random points sampled uniformly from the shape surface. Based on the principles of this method, we derived a two dimensional analog called a shape histogram and used it to recognize independent strokes representing geometric primitives. Experiments show that this derivation is proficient in recognizing geometric primitives and is independent of stroke order, number, and direction, as well as invariant to rotation, scaling, and translation of strokes. As compared to a method based on shape contexts [BMP02], this method does not need to find the correspondence points between two shapes despite the fact that both methods are based on point sampling and histogram similarity computation. Below, the two-dimensional shape histogram method is introduced.

4.1. Uniform sampling

This step uses a series of points to approximate a two dimensional shape. To ensure that the sampling process is conducted efficiently and uniformly, we design a look-up table based approach:

- Step 1: Compute the summed length of all line segments included in the freehand sketch. When each line segment is added, the summed length is saved into table **T** with size n, where n-1 is the total number of the line segments. Table **T** can be represented by a linear array as Equation (3) shows.

$$\mathbf{T} = \{t_i \mid t_i = \sum_{j=0}^{i} L((x_j, y_j), (x_{j+1}, y_{j+1})), 0 \le i \le n-1\} \qquad (3)$$

- Step 2: Generate a random real number r between 0 and the total length t_{n-1}, and then use the binary search algorithm to determine the position m where r is located in the table. This position corresponds to the line segment $((x_m, y_m), (x_{m+1}, y_{m+1}))$.

- Step 3: Generate a random real number l between 0 and 1, which is the proportionality factor between the distances from the sampling point to the two end points of the line segmentation obtained in the previous step. This process can be mathematically represented as Equation (4) and the sampled point (x_k, y_k) is saved into an array **A**.

$$\begin{cases} x_k = x_m + l \times (x_{m+1} - x_m) \\ y_k = y_m + l \times (y_{m+1} - y_m) \end{cases} \qquad (4)$$

- Step 4: Repeating Step 2~Step 3 for 2×n times, we can get n point pairs that are sampled in an unbiased manner.

In the sampling procedure, the sampling density must be considered and it has a linear relationship to the complexity of a shape. Statistically, more samples will provide a more precise approximation of the original shape.

4.2. Two-dimensional shape histogram generation

Once enough random point pairs are sampled, the next step is to build the corresponding distance histogram that

is described by a shape function. The Euclidean distance between two points is used as the shape function. Given *n* point pairs, their distances are calculated. Then by traversing each point pair $(x_i, y_i), (x_{i+1}, y_{i+1})$ in an array **A** and counting the number of sample pairs that fall into a certain distance range, a shape histogram **H** is built.

Since strokes drawn at different times usually have different geometric sizes, a normalization process is needed to account for this difference. We determined a standard value *L* used for normalization. Generally, there are two simple ways to achieve normalization. The first one uses the maximum distance among all sampled point pairs as the standard value. The second one uses the average distance of all sample point pairs as the standard value.

Figure 3 shows the shape histograms of some primitive shapes. The shape histograms for each geometric primitive such as lines, circles, or arcs are similar despite their lengths, directions, or shapes. From these examples, several conclusions can be reached: (1) Different geometric primitives have different two dimensional shape histograms; (2) The freehand strokes of the geometric primitives have similar shape histogram; and (3) The shape histogram is independent of stroke order and direction, as well as invariant to rotation, scaling, and translation of strokes. These conclusions form the basis of geometric primitive recognition. Using the representation scheme explained in this section, recognizing a primitive shape becomes a matter of computing the similarity between two histograms.

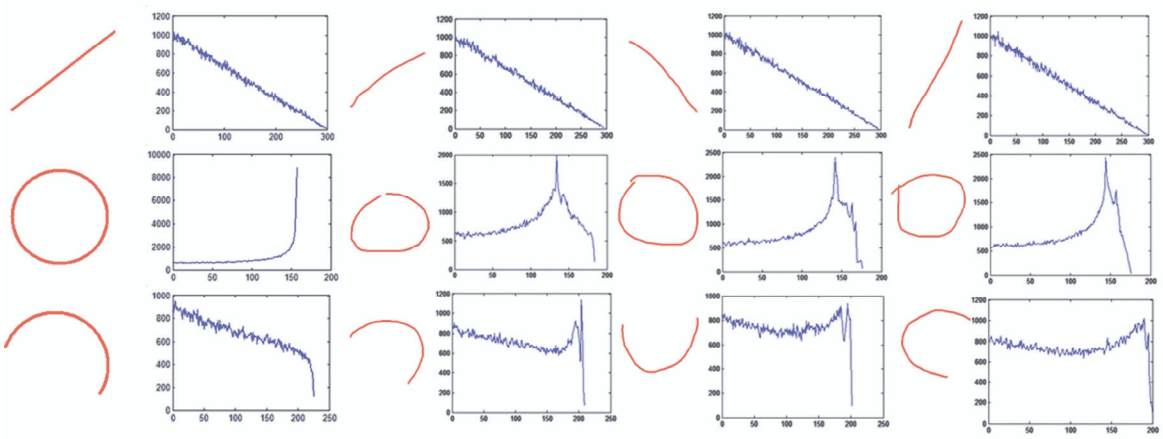

Figure 3: *Shape histograms of geometric primitives such as line, circle and arc. The horizontal axis represents the distance between two points and the vertical axis represents the number of the point pairs with the same distance.*

5. Relative Shape Histogram (RSH)

In order to detect implicit constraints, we propose a method called *Relative Shape Histogram* (RSH) to determine the relationship between two geometric primitives. RSH is similar to the shape histogram method described in Section 4. RSH has the same sketch representation, the same sampling strategy, and the same shape function as the shape histogram method. The key difference is that RSH only considers the Euclidean distances between point pairs that are sampled from different primitives. For example, (x_i, y_i) and (x_{i+1}, y_{i+1}) in Section 4.2 are located on different primitives. By following the same steps of the shape histogram method, RSHs between geometric primitives can be computed in a similar way. In Figure 4, some RSHs between sketched primitives with certain constraints are shown, in which each row represents the same constraints but different sketched shapes such as size and rotation. Based on the examples in Figure 4, several conclusions can be reached: (1) sketches representing the same kind of constraint have similar RSHs; (2) sketches representing different constraints have different RSHs; and (3) the RSH of a constraint is independent of stroke-order and -direction, as well as invariant to rotation, scaling, and translation of strokes.

In implementing this technique, two interesting phenomena are observed. The first is that certain parameters can be estimated from the RSH, as in the first row of Figure 4, where some examples of parallel constraint are shown. Along the horizontal axis, there is a certain distance between the origin and the RSH curve, which corresponds to the distance between the two lines. The same conclusion also holds for the constraints shown in the third and fourth rows.

The second phenomenon is that the RSHs for the same constraints have similar distributions of peaks and troughs. Thus, the histograms of the same constraints

have approximately the same overall component shapes. However, these shape components do not line up along the horizontal axis. As Figure 5(a) shows, the Minkowski distance is not good at computing the similarity between two RSHs because it does not build an intuitive alignment between RSHs. In order to find the similarity between such sequences, a "warp" operation is needed to achieve an intuitive alignment. Below, an approach based on dynamic time warping (DTW) [SK83] is introduced to compute the similarity between RSHs.

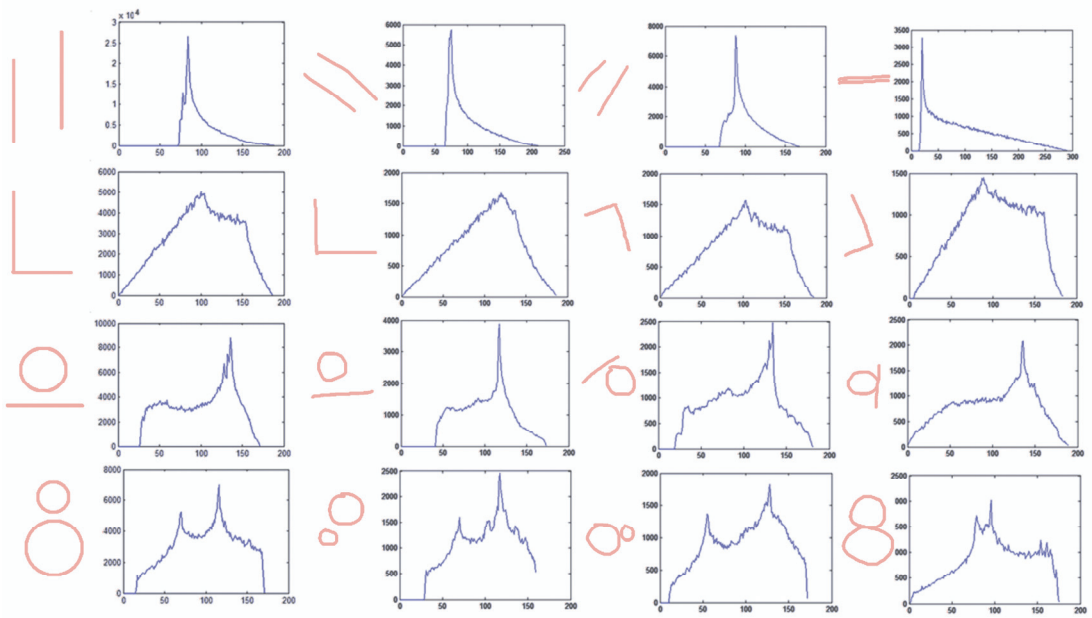

Figure 4: *Examples of relative shape histograms (RSH). The horizontal axis represents the distance between two points from two different strokes and the vertical axis represents the number of the point pairs with the same distance.*

We use DTW to determine the constraint type of the RSH. DTW was originally developed to align two spectral sequences of speech and is now being widely used in speech processing, bio-informatics, and handwriting recognition. Figure 5(b) shows such an alignment by DTW.

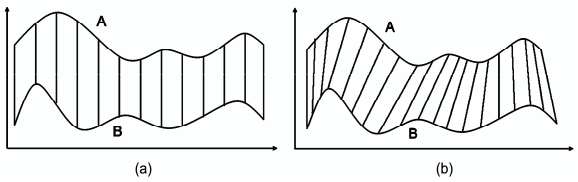

Figure 5: *Two different alignment methods for similar shape histograms A and B: (a) Minkowski distance, and (b)DTW.*

Suppose there are two time series S and T with length m and n respectively:

$$S = \{s_1, s_1, \cdots, s_m\}, \qquad T = \{t_1, t_1, \cdots, t_n\} \tag{5}$$

To align S and T, we need to construct an $m \times n$ matrix **M** whose element at position (i, j) is the distance D between the two points s_i and t_j. Each matrix element (i, j) corresponds to the alignment between the points s_i and t_j. Next, a warping path is found that represents the correspondence between the two sequences S and T. In particular, the DTW is defined as the warping path that minimizes the warping cost, which can be calculated by a dynamic programming approach as:

$$D(i,j) = \min \begin{Bmatrix} D(i,j-1) \\ D(i-1,j) \\ D(i-1,j-1) \end{Bmatrix} + d(s_i, t_j) \tag{6}$$

According to the DTW theory, the following steps are utilized in order to determine the constraint type of an RSH:

- Step 1: Determine a set of templates as standard sequences for each constraint type. The aim is to ensure that the constraints that designers use are not omitted, so that a high level of accuracy can be achieved.
- Step 2: Exclude certain constraint types by checking the distance between the origin and an RSH. For example, if the distance is larger than a predefined tolerance ε, such as the first RSH in the third row of

Figure 4, then we can conclude that the line and the circle are not tangential to each other at all.

- Step 3: Move the histogram of an RSH along the horizontal axes to ensure that the first point s_1 or t_1 is located at the origin.
- Step 4: Compute the DTW warping path between an RSH and a template, where the template with the minimum warping cost represents the desired constraint type. For simplicity, we use the Euclidean distance to compute the distance between two points.

6. Experiments

To qualitatively demonstrate the validity of our proposed method, a preliminary user study was conducted, in which nine participants were asked to sketch two-dimensional drawings freely. Totally, sixty three sketched drawings were collected and used to evaluate the performances of implicit constraint detection. In Figure 6, some sketched samples are listed.

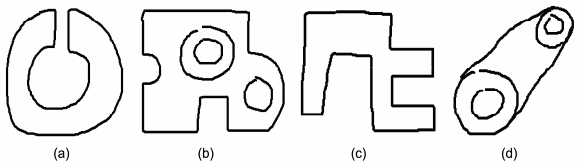

Figure 6: *Sketched examples*.

Given two segmented primitives, four kinds of constraints are checked, including parallelism, concentric, perpendicularity and tangency. The results are presented in Table 2. It can be seen that there are no false negative results while there is a high rate of false positive results. This is due to the fact that some constraints, which are not intended by sketchers, are incorrectly detected, since the detection process of implicit constraint is purely based on the similarity computation of RSHs and no qualitative factor such as the types of the involved geometric entities.

Table 2: *Implicit Constraint Detection*.

Constraint Type	Parallelism	Co-center	Perpendicular	Tangent
True Positive	57	22	78	17
False Positive	14	6	19	4
False Negative	0	0	0	0
Total	71	28	97	21
Accuracy (%)	80.3%	78.6%	80.4%	80.9%

Actually, when the types of the involved geometric entities are considered, the false positive can be decreased obviously because constraint types have a direct relationship with the geometric entities. For example, perpendicular constraint will never occur for a line and a circle.

With the help of the method introduced in this paper, together with a constraint solver, freehand sketches can be beautified in a more accurate way. Two examples of beautification driven by constraints are presented in Figure 7, where the explicit geometric constraints such as dimensions are specified by users interactively. For more detailed information about constraint solvers, please refer to [Aldefeld98, BFH*95, FH97]. In the example shown in Figure 7(a), the implied tangent constraints are detected automatically and users have no need to specify them explicitly. It can be seen from the two examples that the sketched design schemes can be transformed to the final design efficiently with the help of the constraint detection component and a constraint solver.

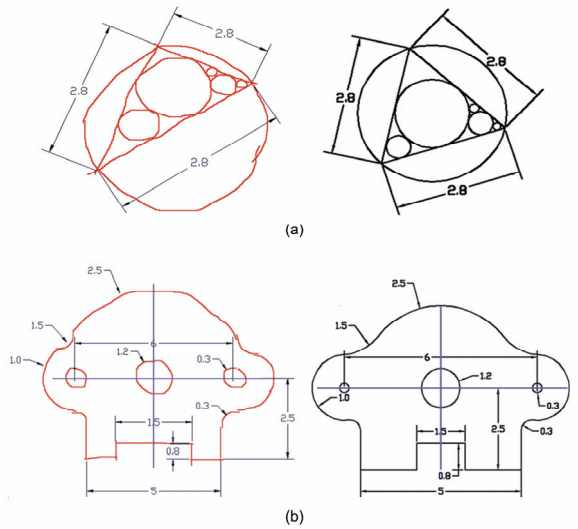

Figure 7: *Examples of sketch beautification driven by geometric constraints*.

7. Conclusion

Freehand sketching is widely believed to be the most natural human-computer interaction manner that has potential applications in many fields, such as computer aided design, geometric modeling, geometric theorem proofs, tolerance analysis, and robotics. In this paper, an algorithm is proposed to detect the implicit geometric constraint in freehand sketches so that more intent from users can be used in sketch beautification. The prior condition is that the freehand sketches should be segmented as the input of this algorithm, usually is used together with sketch segmentation for beautification purpose. As a statistical method, the basic idea of the proposed method is to represent the relationship between two segmented sketches in the form of histogram that has some valuable advantages, such as transformation-invariance, stroke-speed, and curvature independence. We are currently trying to incorporate this beautification into a new paradigm for computer-aided design, which is

expected to provide users with great flexibility, high efficiency, and naturalness in product design.

Acknowledgements

We acknowledge the early support of 21st Century Research and Technology funds in Shape Search. This research is partly supported by the National Science Foundation under grant IIS No. 0535156. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[Li03] LI Y.: Incremental sketch understanding for intention extraction in sketch-based user interfaces. CS Technical Report, University of California, Berkeley (October 2003). UCB//CSD-03-1284.

[FLK*04] FORBUS K., LOCKWOOD K., KLENK M., TOMAI E., USHER J.: Open-domain sketch understanding: the nuSketch approach. In *Proc. AAAI Fall Symposium on Making Pen-based Interaction Intelligent and Natural* (2004), pp.58-63.

[Stahovich04] STAHOVICH T. F.: Segmentation of pen strokes using pen speed. In *Proc. 2004 AAAI Fall Symposium on Making Pen-based Interaction Intelligent and Natural* (2004), pp.152-158.

[LS05] LANK E., SAUND E.: Sloppy selection: providing an accurate interpretation of imprecise selection gestures. *Computers and Graphics 29*, 4 (2005) (Special Issue on Pen Computing), pp.183-192.

[NM04] NOTOWIDIGDO M., MILLER R. C.: Off-line sketch interpretation. In *Proc. AAAI Fall Symposium Series 2004: Making Pen-Based Interaction Intelligent and Natural* (2004), pp.120-126.

[GKS05] GENNARI L., KARA L. B., STAHOVICH T. F.: Combining geometry and domain knowledge to interpret hand-drawn diagrams. *Computers & Graphics 29*, 4 (2005), pp. 547-562.

[SD05] SEZGIN T. M., DAVIS R.: HMM-based efficient sketch recognition. In *Proc. 10th International Conference on Intelligent User Interfaces* (2005), pp.281-283.

[HN04] HSE H., NEWTON A. R.: Sketched symbol recognition using Zernike moments. In *Proc. 17th International Conference on Pattern Recognition* (2004), pp.367-370.

[KS04] KARA L. B., STAHOVICH T. F.: An image-based trainable symbol recognizer for sketch-based interfaces. In *Proc. AAAI Fall Symposium Series 2004: Making Pen-Based Interaction Intelligent and Natural* (2004), pp.99-105.

[AD04] ALVARADO C., DAVIS R.: SketchREAD: a multi-domain sketch recognition engine. In *Proc. 17th Annual ACM Symposium on User Interface Software and Technology* (2004), pp.23-32.

[KB90] KALRA D., BARR A. H.: A constraint-based figure-maker. In *Proc. European Computer Graphics Conference and Exhibition* (1990), pp. 413-424.

[Aldefeld98] ALDEFELD B.: Variation of geometries based on a geometric reasoning method. *Computer Aided Design 20*, 3 (1998), pp. 117-126.

[BFH*95] BOUMA W., FUDOS I., HOFFMANN C. M., CAI J., PAIGE R.: A geometric constraint solver. *Computer Aided Design 27*, 6 (1995), pp. 487-501.

[FH97] FUDOS I., HOFFMANN C. M.: A graph-constructive approach to solving systems of geometric constraints. *ACM Trans. on Graphics 16* (1997), pp. 179-216.

[BD86] BORNING A., DUISBERG R.: Constraint-based tools for building user interfaces. *ACM Transactions on Graphics 5,* 4 (1986), pp. 345-374.

[WSH05] WALLNER J., SCHROCKER H. P., HU S. M.: Tolerances in geometric constraint problems. *Reliable Computing 11*, 3 (2005), pp. 235-251.

[BS86] BIER E. A., STONE M. C.: Snap-Dragging. In *Proc. ACM SIGGRAPH Computer Graphics 20*, 4 (1986), pp. 233-240.

[KWL93] KARSENTY S., WEIKART C., LANDAY J. A.: Inferring graphical constraint with Rockit. In *Proc. Conference on People and Computer VII* (1993), pp. 137-153.

[OFC*02] OSADA R., FUNKHOUSER T., CHAZELLE B., DOBKIN D.: Shape distribution. *ACM Trans. on Graphics 21*, 4(2002), pp. 807-832.

[BMP02] BELONGIES S., MALIK J., PUZICHA J.: Shape matching and object recognition using shape context. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24*, 24 (2002), pp. 509-522.

[SK83] SANKOFF D., KRUSKAL J. B.: Time warps, string edits, and macromolecules. *The Theory and Practice of Sequence Comparison* (1983), Addison-Wesley, Reading, MA.