Interpretation of Overtracing Freehand Sketching for Geometric Shapes

Day Chyi Ku
School of Engineering and Design
Brunel University
Uxbridge
UB8 3PH, UK
Daychyi.Ku@brunel.ac.uk

Sheng-Feng Qin
School of Engineering and Design
Brunel University
Uxbridge
UB8 3PH, UK
Sheng.Feng.Qin@brunel.ac.uk

David K. Wright
School of Engineering and Design
Brunel University
Uxbridge
UB8 3PH, UK
David.Wright@Brunel.ac.uk

ABSTRACT

This paper presents a novel method for interpreting overtracing freehand sketch. The overtracing strokes are interpreted as sketch content and are used to generate 2D geometric primitives. The approach consists of four stages: stroke classification, strokes grouping and fitting, 2D tidy-up with endpoint clustering and parallelism correction, and in-context interpretation. Strokes are first classified into lines and curves by a linearity test. It is followed by an innovative strokes grouping process that handles lines and curves separately. The grouped strokes are fitted with 2D geometry and further tidied-up with endpoint clustering and parallelism correction. Finally, the in-context interpretation is applied to detect incorrect stroke interpretation based on geometry constraints and to suggest a most plausible correction based on the overall sketch context. The interpretation ensures sketched strokes to be interpreted into meaningful output. The interface overcomes the limitation where only a single line drawing can be sketched out as in most existing sketching programs, meanwhile is more intuitive to the user.

Keywords

Freehand sketching, multistroke sketching, calligraphic interface, design

1. INTRODUCTION

Sketch interface is an important, natural application to support conceptual design. A sketch system implemented in a computer has the advantage whereby further manipulations and processing, such as 3D modelling from 2D sketch, can be made directly with minimal steps within a short time. However, this is realistic only if the system allows the flexibility of transferring ideas directly from designers through a series of freehand sketch. To support this process, the system should provide an interface that is natural to the user as sketching with pen and paper. Users may find sketching with extensive menus difficult as a result of frequent interruptions.

Conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

On the other hand, the system will also need to be able to correctly interpret the user's intent from the drawing of the sketch. A trade-off is often required in the design of the calligraphic interface between being natural, easy-to-use, and that of accuracy of the system's interpretation of user's intent.

One problem in the design of a natural but accurate calligraphic interface is that of interpreting overtracing of freehand sketch. Overtracing is frequently used to enhance and complete an edge during freehand sketching. A system that supports overtracing, i.e., accepts multiple stroke inputs, has the advantage of providing more drawing freedom to the user. However, at the same time, overtracing further increases the number of possible interpretations of a sketch, and as such, making the system more susceptible to making mistakes in interpreting the user's actual intent.

In this paper, we are concerned specifically with the interpretation of overtracing freehand sketches of geometric objects. Although there are sketch systems that support overtracing inputs, they do not actually interpret them [FIO02a, GRO00a, KAR05a], or provide limited interpretations (e.g., only supports certain sketching primitives or has a very strict

definition of how a sketch is interpreted) and as such, not applicable to general cases [MIT02a, SHE04a].

Here, we developed a system that supports and interprets overtracing freehand sketch of general geometric objects (Figure 1). The work presented in this paper is a part of our on-going project that is aimed at reconstructing 3D models from 2D sketches. The preliminary system is developed to address the problem of interpreting overtraced strokes, i.e., multiple strokes that are part of the same geometric primitives.

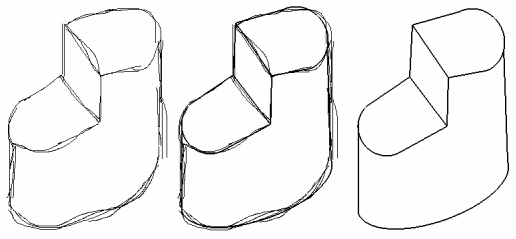


Figure 1: A freehand sketch with overtracing and its tidy-up output from our system.

There are two important differences between our system and other earlier, calligraphic interface systems such as in [MIT02a, SHE04a]. First, our system supports and interprets both straight-line and curve overtracing sketches. In [SHE04a], the system can only interpret straight-line input. Second, our system interprets overtracing strokes with various curvatures and represents them in parametric equations that best represent the strokes. In [MIT02a], the overtracing strokes are represented by polylines after grouping into core curves and only curves with low curvatures are showed in their examples.

Furthermore, some of the systems impose frequent interruptions to users during the sketching process. This is normally associated with interactive systems that prompt users for selection of choices [IGA97a]. The frequent interruptions can be a source of distractions that can impede the flow of thoughts of the designer, and as such, should be addressed in the design of a calligraphic interface.

Therefore, the interpretation process in our system is carried out automatically and designed to have minimum interruption to the user. To achieve this, we designed the system to carry out the interpretation process between sketching sessions rather than interrupting the user during sketching for further inputs to clarify ambiguous cases. This allows a user to draw in a more intuitive and natural way without diverting attention from the sketching flow.

2. RELATED WORKS

Computer calligraphic interface has been receiving more attention over recent years. However, despite the availability of hardware such as the stylus tablet, there are still many problems left to be solved in the development of a full-fledged, functional free-hand sketching calligraphic interface, such as interpretation of multiple strokes, in-context classification and clustering of strokes, inferring constraints from freehand sketches for tidy-up and beautification.

Perhaps Pavlidis [PAV85a] made the first attempt to infer constraints from initial freehand sketch to automatically tidy-up rectilinear drawings. However, the method is bound to produce unintended and undesirable results in practice and negative constraints were suggested to address the problem [PAV85a]. Similarly in Easel [JEN92a], Jenkins and Martin developed a system that can automatically analyze and tidy-up sketches. Easel introduced more constraints and processed freeform curves [JEN92a]. Pegasus [IGA98a, IGA97a] is a prototype system for beautification of freehand sketching through an interactive process. The system generates several candidate strokes based on geometry constraints for user selection. This interaction process is to ensure that the resulting sketch is as closely desired by the user, i.e., the precise and correct sketch that the user had in mind but unable to accurately draw himself. However, the process of selecting candidates, stroke-by-stroke, is somewhat distracting to the user. Consequently, the system is not suitable for conceptual design sketching, where the designer must be allowed to sketch without interruption to ensure a continuous flow of ideas.

In addition to the tidy-up process, there are other research efforts focused on sketch interpretation problems such as the strokes classification and clustering. For example, Shpitalni and Lipson [SHP97a] presented a method for classifying input strokes in an online sketching system that is based on linear least squares fitting to a conic section equation. They also introduced new endpoints clustering scheme based on adaptive tolerances [SHP97a]. Qin [QIN00a] introduced a system that classified strokes using adaptive threshold and fuzzy knowledge with respect to curve's linearity and convexity. After that, 2D primitives are identified and a 2D relationship inference engine is used to study their relationship for 3D recognition.

Although advances have been made in developing a functional calligraphic interface for freehand sketching, in general, most of the current 2D freehand sketch interpreters have the limitation of either not supporting overtracing sketches [JEN92a, IGA98a, IGA97a, PAV85a, QIN00a, QIN00b, SHP97a], or have limited support ([SHE4a], [MIT02a]). SMARTPAPER [SHE4a] groups overtracing strokes into segments. The grouping

process was carried out in two passes where the distance between end points and the slopes of the strokes are checked against each other. However, SMARTPAPER is limited to straight-line input with limited configurations only. Furthermore the system does not support curves input. There is no discussion of wrongly drawn lines correction and hence it is assumed that the system is limited to interpret sketches with correct strokes. 3D SKETCH [MIT02a] supports both overtracing and hatching sketch. Strokes are divided into core strokes (strokes that touch the characteristic curves of the object), and hatching strokes (strokes that are mapped to the faces of the object). Although the system can interpret overtracing strokes (e.g., by grouping strokes into bundles), there is no explanation of how the characteristic strokes are distinguished from the hatching strokes. Furthermore, the system used polylines to represent the core curve limited its accuracy in representing curves with higher curvature.

There are other sketch systems that support overtracing strokes but do not interpret them as part of the sketch. In [GRO00a], the system filters out overtraced lines to produce simpler, approximated drawings (e.g., filtering out elements that are smaller than a specified size) rather than performing interpretations in the context of the sketch as a whole. In [KAR05a], the sketch recognizer allows overtracing symbol recognition based on image processing techniques, e.g., it relies entirely on the symbol libraries for the recognition where the sketch is examined in pixel and hence no work is done on interpretation of individual stroke. In [FIO02a], overtracing is used to edit an existing stroke that is represented in cubic Bezier splines. The movements of oversketching is sampled and interpreted specifically as transformation attractor to the underlying curve's control points. The overtracing strokes are used as gesture input to alter the existing sketch rather than sketching information that actually add more lines to the existing sketch.

In this paper, we propose an interactive calligraphic interface that addresses the limitations discussed above. Referring to the systems discussed above, we use Qin's curve classification, Shiptalni and Lipson's conic fitting equation, and others geometry constraints as the backbone of our system to interpret and tidy-up freehand sketch. However, stroke pre-processing, i.e., segmentation, is not covered in this paper. All input strokes are taken to represent only a segment or part of a segment. However, the system here can be easily extended to include pre-processing algorithms for segmentation such as those described in [QIN00b, SEZ01a].

3. SYSTEM OVERVIEW

We propose the interpretation of overtracing strokes by carrying out a set of tests automatically to group and tidy up the sketched strokes into segments. The output of the system is edge-vertex graph with initial sketched strokes associated to the edges. The graph can be transferred into CAD or 3D modelling systems for further processing and manipulation while the edges can be reproduced in sketchy style with the sketched strokes information. Another advantage to our proposed system is that the new edges that looked similar to the original sketching style can be produced from the information obtained during the sketching process. However, for this paper, we focus only on planar objects and objects with ellipsoidal curves for inputs to reduce the system's bounds in producing unintended and undesirable result. Additional work is needed to extend the system to support the interpretation B-spline curves and freeform objects.

Interruptions such as prompts for clarifying ambiguous strokes are made at the end of the drawing session. For example, our system prompts the user to decide whether an ambiguous stroke should be deleted or kept with the existing drawing only at the end of the drawing session.

The system can take in the sketch directly from the user through a tablet with a digital pen or mouse. A stroke is a set of points that is captured during the period when the mouse button is pressed down, moved, and released. An edge refers to the intersection of two faces of a solid object, which in 2D drawing is represented by strokes. In many drawings an edge is often composed of a set of discontinuous strokes. The overtracing strokes are used to complete, correct or enhance an edge in the sketch. The strokes will be processed by the 2D sketch interpreter and automatically grouped together to represent the associated edges.

The 2D sketch interpretation is divided into five stages: grouping and fitting of strokes, end points clustering, parallelism correction, in-context interpretation and user interactive selection. First, all classified strokes (using the linearity test) will be grouped according to their positional relations. After that, the grouped strokes are fitted collectively into an appropriate parametric equation in the least square sense. A tidied-up sketch will be produced from the equations. Lastly, lines with similar gradients (within some tolerance levels) are adjusted to be parallel to each other, thus reflecting the user intention because it is not possible for the user to draw exactly parallel edges in freehand sketch. When there are strokes that cannot be interpreted correctly according to geometry constraints, the system will tag the strokes as ambiguous cases. The system only prompts the

user for further action and correction of the ambiguous cases at the end of the drawing session.

After the interpretation process, the system is then able to generate tidied-up 2D single line drawing, while at the same time, still have initial sketchy strokes linked to the appropriate tidied-up edges.

4. CLASSIFITION AND GROUPING

4.1 Stroke Classification

All drawn strokes will be classified into either one of the categories, straight-lines or curves. The linearity test is used to classify a straight-line from a curve [QIN00a]. It is simpler and faster compared to the conic curve equation used in [SHP97a]. This is because the linearity test involves only a simple calculation of two parameters, while the conic curve equation requires the calculation of five parameters in the quadratic equation.

The linearity is defined as the ratio of the distance between two endpoints of a stroke to the accumulative chord length between sequences of points captured.

The linearity value is a floating point between zero to one. A greater linearity value for a stroke indicates that the stroke is more likely to be a straight-line rather than a curve. An ideal straight-line will have unity linearity, which rarely happened in freehand sketching. Therefore, we set an arbitrary straight-line tolerance for the classification.

Each classified stroke will be fitted either into a general straight-line equation to generate a straight-line, or a general conic curve equation to generate a curve. With equations, strokes can be represented in a more general and effective way compared to using a list of points. Furthermore, the representation of input strokes with parameterised equations allow for more effective and efficient tests of input strokes during the interpretation process.

4.2 Strokes Grouping and Fitting

The strokes grouping process groups sketched strokes into the appropriate edges that they represent. However, the grouping process only tries to group strokes in the same category, i.e., lines only grouped with lines, and curves only grouped with curves. The grouped strokes are fitted with equations to obtain parameters that best represent them.

4.2.1 Line Segment Grouping and Fitting

A straight-line is approximated by fitting data points of the strokes into a line equation that give the minimum least-square error [ONE83a]. The fitted line end points are the points on the equation that have the minimum Euclidian distance from the sketched stroke end points. All straight-lines will be tested to determine whether they should be grouped together to represent the same edge. The line segments will be grouped together if they are close in position and orientation to each other.

4.2.1.1 Distance Tests

A series of distance tests is used to determine the smallest distance between two line segments. The tests are carried out in sequence as follows:

1) Get the smallest Euclidean distances between endpoints of lines *A* and *B* (Figure 2). The resulting distance is compared with a threshold value. The threshold value represents the largest tolerance of the distance between two line segments to be grouped together. The threshold value varies according to the lines' length, as in [SHP97a]. If the two line segments (*A* and *B*) pass the threshold test, go to the angular test, otherwise go to step 2.
2) Calculate the perpendicular Euclidean distances from line endpoints of one line to the other line segment, as shown in Figure 2. Eliminate the distances that corresponding projection points are out of the other line segment, e.g., (a) and (d) in Figure 2. Compare the smallest distance among the remains, (b), with a threshold value. If the distance is smaller than the threshold, go to the angular test. Otherwise the two line segments are grouped into separate edges.

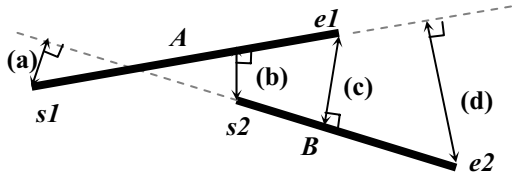


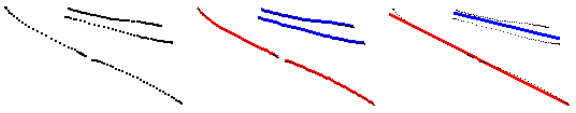
Figure 2: Distance calculated between *A* and *B*.

4.2.1.2 Angular Test

The Angular test is to ensure that strokes grouped as one edge are pointing in the same direction.

The absolute dot product value, which is the smallest angle between the lines *A* and *B*, is calculated. The two lines will be grouped into one segment if the angle is smaller than a threshold value. The threshold value is adjusted according to the longest length of the lines *A* and *B*. The value is calculated to be inversely proportional to the length of the corresponding line, e.g. the longer line will have a smaller threshold value. This is because the longer length of lines will "look" more apart than shorter line with the same angle.

The lines do not have to intersect, or overlap each other to be grouped together. The grouped strokes will undergo another round of straight-line least-square fitting to update its equation and endpoints (Figure 3).



(a) (b) (c)

Figure 3: Line Grouping: (a) Initial sketch; (b) Grouped sketch; (c) Fitted result.

We assume that earlier drawn stroke is more likely to represent an edge position and orientation, while strokes drawn later are the overtracing to enhance or complete the edge. Therefore, earlier drawn stroke is used as reference to test against candidate strokes to determine if the strokes should be grouped into the edge. If a candidate stroke fails the tests for all existing edges, then it will be grouped as a new edge and used as reference for the edge.

4.3 Curve Grouping and Conic Fitting

Strokes classified as curves by the linearity test will undergo the curve fitting process before curve grouping can be carried out. They will be fitted using a conic curve equation [SHP97a] to obtain the curve parametric equation. The fitted result will fall in one of the following category: straight-line, parabola, hyperbola, or ellipse.

However, straight-lines will not be generated due to the pre-processing of strokes through the linearity test. Hyperbola and parabola are considered as special cases by our system on the assumption that they rarely occurred in sketched geometric objects. Consequently, we only consider the elliptical curves that result from the fitting. Circle is treated as special case of ellipse where the major and minor radii are identical. After that, all sketched curves are represented by the parametric equations generated from the conic fitting procedure.

A general conic curve can be described by the following equation [BOW83a]:

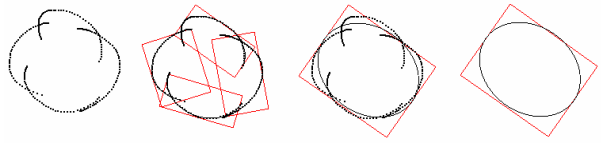
$$Q(x,y) = ax^2 + 2hxy + by^2 + 2gx + 2fy + c = 0$$

In our system, we need to find the least square fitting based on the distance between the captured sketching points and the equation. The fitting problem is then reduced to minimizing the following function:

$$E = \sum_{i=1}^{n} \left(ax_i^2 + 2hx_i y_i + by_i^2 + 2gx_i + 2fy_i + 1\right)^2$$

We obtain the coefficients (*a, h, b, g,* and *f*) by solving the partial derivatives of *E* equals to zero. The central point, major and minor radii, and the rotation angle of an ellipse can be obtained from the equation as in [QIN99a].

A bounding box, the smallest rectangle to enclose a fitted curve, is used to test for the curves adjacency for grouping them together. If the bounding boxes overlap one another, the sketched strokes associated with the curves will be grouped together. The grouped strokes will be fitted again. The overlapping test will be repeated for new fitted curve until there is no more overlapping bounding boxes in the sketch (Figure 4).



(a) (b) (c) (d)

Figure 4: Curve Grouping and Fitting: (a) Initial sketch; (b) Generate bounding box for each stroke; (c-d) Grouped strokes with fitted result.

4.3.1 Curve Range

A curve range is calculated from its starting point to the ending point, with reference to the centre point of the curve [JEN92a]. The user has the freedom in sketching a curve in any desired direction. However, our system standardized all curve range in anti-clockwise. The ending angle can be a value greater than 360 degree, as shown in Figure 5.

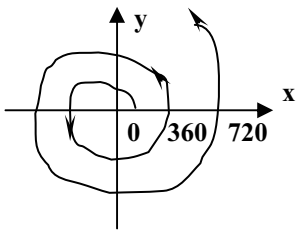


Figure 5: Curve range calculation.

Over-sketched and incomplete curves often occur in freehand sketching. Our system will tidy-up the curves into smooth ellipse or circle as shown in Figure 6. The curve range for grouped curves is updated automatically by our system, as shown in Figure 7.

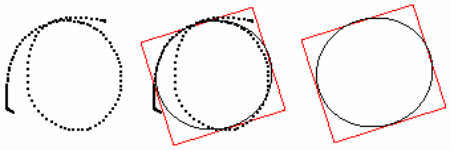


Figure 6(a): Over-sketched Curve.

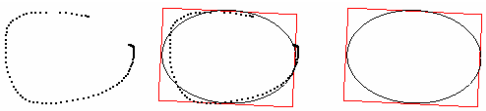


Figure 6(b): Incomplete Curve.

(a) (b) (c) (d) (e) (f)

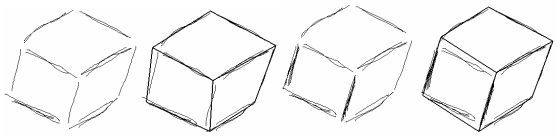
Figure 7: New curve range is determined from grouped curves: (a) Initial sketch; (b-c) Curve with initial range; (d) Additional stroke; (e) Strokes are grouped as a curve; (f) Curve with new range.

5. ENDPOINT CLUSTERING

After the grouping process, strokes representing a particular edge are approximated by a single line or ellipse parametric equation. The endpoint clustering process ensures that the corresponding edge endpoints meet together and that a close loop can be formed for edge-graph extraction.

5.1 Straight-line Junction Clustering

The straight-line junction clustering is applied for the line-to-line clustering. Before the clustering can be applied, the system first finds out endpoints of edges that are adjacent to each other and within the tolerance zone as discussed in [SHP97a]. Edges with endpoints that lie within the zone will be clustered together to form a junction. In case of a junction with more than two edges, the system will select two edges with the most number of strokes to determine the junction point, the rest of the edges will automatically snapped to the point (Figure 8).

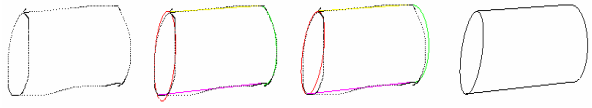


(a) (b) (c) (d)

Figure 8: Straight-line junction clustering: (a) Initial sketch; (b) Clustering result; (c) Modified sketch; (d) Updated clustering result.

5.2 Lines and Curve Junction Clustering

The curve edge will be adjusted based on the line edges endpoints position so that the curve edge endpoints (open curve) or boundaries (closed curve) will meet the line edges endpoints to form a close loop (Figure 9).

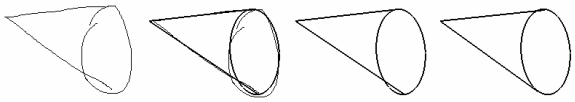


(a) (b) (c) (d)

Figure 9: Endpoint clustering for lines and curves junctions: (a) Initial sketch; (b) Fitted result; (c-d) Clustered result.

The orientation of the curve is determined to be parallel to the imaginary line segment L1 that connects the endpoints of lines that are to form junctions with the curve. After that, the curve's central point is determined to be the middle point of the line segment L1. The major or minor radius of the curve is calculated based on the length of L1, with the other radius being adjusted according to the distance from sketched curve to central point. For open curve, new curve range is determined based on the line endpoints.

For closed curve, lines are adjusted to be tangent to the curve boundaries, that is, the line is snapped to the curve at only one intersection point. Figure 10 shows an example of the adjustment made on such junction.



(a) (b) (c) (d)

Figure 10: Intersection adjustment on full ellipse for line and curve junction (a) Initial sketch; (b) Fitted result; (c) Before adjustment; (d) After adjustment.

5.3 Parallelism Correction

After the endpoints clustering process, the edges of the object can be represented by single 2D geometry, in perfect straight-lines or ellipses that are joined to each other. To allow some tolerance for freehand sketching error, straight-lines are tested if they have similar gradients or orientations. A slight change in the orientation of lines can be done to ensure parallelism in the sketch, which is often difficult to be achieved by freehand sketching.

Each straight-line edge is tested against all the other edges to obtain their similarity with each other. Edges drawn with more number of strokes are determined to be more important and such, are used as the reference. All adjustment of an edge is achieved by rotating at the middle point of the edge. Figure 11 shows an example of the parallelism correction.

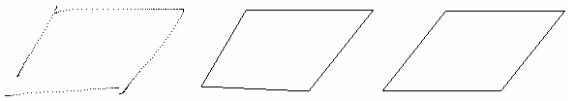


Figure 11: A rectangle before and after parallelism correction.

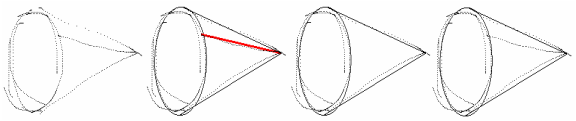
If the parallelism correction process changes line endpoints, the clustering process (as discussed in 5.1) will be carried out for the associated strokes to ensure the junction connectivity.

6. IN-CONTEXT INTERPRETATION OF STROKE

The calligraphic interface system developed here is meant as a preliminary system taking in 2D sketches, which are then interpreted so that they can later be used for 3D models reconstruction. As such, the system's interpretation is set to process sketches of 3D geometric objects drawn on 2D. We consider the isometric projection, although other projections can be used as well.

For a 2D sketch, a closed loop line or curve is used to represent a face of a 3D geometric object. A closed loop is normally interpreted as a surface, depending on the shape of the object. A complete 3D object should have no open edge in sketch. Any stroke with endpoint unconnected to a junction in the sketch will be considered as an error, and the stroke will be tagged as ambiguous. The system will highlight the strokes and prompt the user with a choice to either delete or keep them. If the user decides not to delete the strokes, the system will reinterpret the strokes based on the context so that they satisfied the geometry constraints.

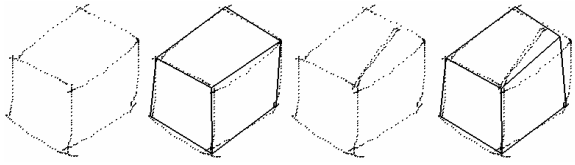
There are two sources for the ambiguous strokes. Firstly, the strokes can be caused by the misclassification of strokes at the beginning of the interpretation. For example, a short curve drawn with high sketching speed can be misinterpreted as a line by the classification method. To correct it, the system will reclassified it as curve, and continue with the grouping and fitting with other strokes in the sketch. Secondly, the stroke might be caused by poor sketching skill of the user, where overtracing strokes that meant to be grouped into an edge are sketched too far apart. The overtracing strokes failed the grouping tests and were grouped into separate edge. The both edges have the same endpoints that can be detected by our system, and grouped into one edge. Similar restriction applies to cylindrical object, where straight-lines only intersect curve at the boundaries of the curve surface (Figure 12).



(a) (b) (c) (d)

Figure 12: Lines detected as ambiguous with endpoints at an arc: (a) Input sketch; (b) Detected ambiguous stroke is in thick red color; (c) Result when user choose to delete the strokes; (d) Result when user choose to merge the strokes.

The in-context interpretation also allows the user to modify sketch with overtracing strokes (Figure 13). The system will try to connect 'open' strokes in the sketch to its nearest junctions, resulting regrouping and refitting of the strokes. The fitted result is therefore moved by the new strokes and modified the initial sketch.



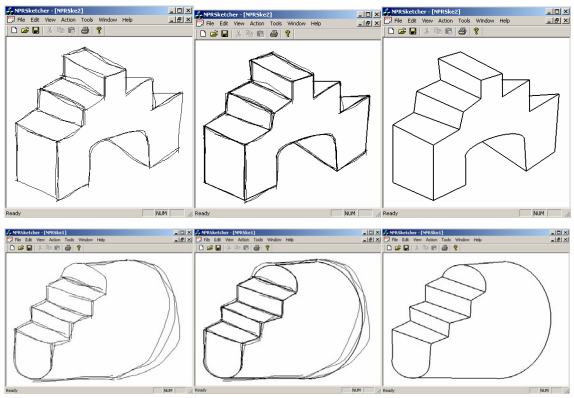
(a) (b) (c) (d)

Figure 13: Modification by overtracing; (a-b) Initial sketch with fitted result; (c-d) Sketch with 'open' strokes and new fitted result.

7. IMPLEMENTATION AND EXAMPLES

The system is implemented using Visual C++ under the Windows 2000 operating system. The input device is a traditional digitizing tablet that senses the drawing on its screen.

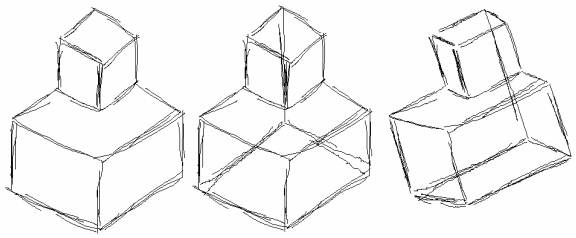
Figure 14 shows some examples of the sketch before and after tidy-up with our system. The interface allows the user to sketch freely as though using pen and paper. The system will interpret and tidy-up the sketch into a single line drawing.



(a) (b) (c)

Figure 14: (a) Input sketch; (b) Tidied-up sketch; (c) Single-line output.

Our system keeps the initial sketch information and associates them with the corresponding edges in tidy-up drawing. The information can be used to render additional new strokes in the sketch that are of similar appearance to the original sketch even though the new strokes are not actually drawn by the user. This suggests that the result from our system can be used as input for non-photorealistic rendering (NPR) system that renders 3D objects with sketchy appearance that is similar to the original sketch by the user. Figure 15 shows the result of such an implementation.



(a) **(b)** **(c)**

Figure 15: (a) Initial sketch; (b) 3D model in NPR; (c) 3D model after rotation.

8. CONCLUSION

This paper presents an interactive calligraphic interface for conceptual design, which supports multistroke sketching. It is designed to handle freehand sketches with overtracing and imperfections. User can sketch over the existing drawing to enhance, complete or correct an edge. The system provides in-context interpretation for sketch that depict 3D geometric object. The interpretation provides verifications and corrections for errors made during sketching session, thus achieving meaningful tidy-up result.

We proposed an alternative interpretation of overtracing strokes from freehand sketch input from the existing systems. We consider all strokes as part of the sketch and grouped them into edges. We introduce a new method to group multiple curves using the minimal enclosing bounding box. The method is fast and hence suitable for online sketching.

The work presented here is only the first part of our final sketched-based 3D modeling and rendering system. The freehand sketch input is processed and tidied-up for the 3D reconstruction. The reconstructed 3D model will have a photorealistic appearance by default. The future work is to reconstruct and display the 3D model with the appearance similar to the original sketch in a non-photorealistic rendering form.

9. REFERENCES

[BOW83a] Bowyer, A., and Woodwark, J., A Programmer's Geometry, Butterworths, 1983.

[FIO02a] Fiore, F.D., and Reeth, F.V., A Multi-Level Sketching Tool for Pencil-and-Paper Animation, AAAI 2002 Spring Symposium., Tech. Rep. SS-02-08, 2002.

[GRO00a] Gross, M.D., and Do, E.Y., Drawing on the Back of an Envelope: a framework for interacting with application programs by freehand drawing, Computers & Graphics, vol.24, pp.835-849, 2000.

[IGA98a] Igarashi, T., Kawachiya, S., Tanaka, H., and Matsuoka, S., Pegasus: a drawing system for rapid geometric design, ACM Press, pp. 24-25, 1998.

[IGA97a] Igarashi, T., Matsuoka, S., Kawachiya, S., and Tanaka, H., Interactive beautification: a technique for rapid geometric design, ACM Press, pp.105-114, 1997.

[JEN92a] Jenkins, D.L., and Martin, R.R., Applying constraints to enforce users' intentions in free-hand 2-D sketches, Intell.Syst.Eng., vol.1, pp.31-49, 1992.

[KAR05a] Kara, L.B., and Stahovich, T.F., An image-based, trainable symbol recognizer for hand-drawn sketches, Computers & Graphics, vol.29, pp.501-507, 2005.

[MIT02a] Mitani, J., Suzuki, H., and Kimura, F., 3D sketch: sketch-based model reconstruction and rendering, Kluwer Academic Publishers, pp.85-98, 2002.

[ONE83a] O'Neil, P.V., Advanced engineering mathematics, Wadsworth Publishing, pp.1091-1093, 1983.

[PAV85a] Pavlidis, T., and Wyk, C.J.V., An Automatic Beautifier for Drawings and Illustrations, SIGGRAPH., vol. 19, pp.225-234, 1985.

[QIN01a] Qin, S.F., Wright, D.K., and Jordanov, I.N., On-line segmentation of freehand sketches by knowledge-based nonlinear thresholding operations, Pattern Recognit, vol.34, pp.1885-1893, 2001.

[QIN00a] Qin, S.F., Investigation of Sketch Interpretation Techniques Into 2D and 3D Conceptual Design Geometry, University of Wales Institute, Cardiff, PhD Thesis, 2000.

[QIN00b] Qin, S.F., Wright, D.K., and Jordanov, I.N., From on-line sketching to 2D and 3D geometry: A system based on fuzzy knowledge, CAD Computer Aided Design, vol.32, pp.851-866, 2000.

[QIN99a] Qin, S.F., Jordanov, I.N., and Wright, D.K., Freehand drawing system using a fuzzy logic concept, CAD Computer Aided Design, vol.31, pp.359-360, 1999.

[SEZ01a] Sezgin, T.M., Stahovich, T., and Davis, R., Sketch based interfaces: early processing for sketch understanding, PUI '01: Proc. of the 2001 workshop on Percetive user interfaces, pp.1-8, 2001.

[SHE04a] Shesh, A., and Chen, B., SMARTPAPER: An Interactive and User Friendly Sketching System, Comput.Graphics Forum, vol.23, pp.301-301, 2004.

[SHP97a] Shpitalni, M., and Lipson, H., Classification of sketch strokes and corner detection using conic sections and adaptive clustering, J Mech Des, Trans ASME, vol.119, pp.131-135, 1997.