The Visual Computer manuscript No.
(will be inserted by the editor)

Richard Pusch · Faramarz Samavati · Ahmad Nasri · Brian Wyvill

Improving the Sketch-Based Interface

Forming Curves from Many Small Strokes

Abstract Sketch-based interfaces are becoming a useful methodology for interaction with a wide range of applications. Drawing is a natural and simple paradigm for designers. One of the problems in most of the current generation of such interfaces is that designers are forced to use single strokes where they may prefer to use many strokes while drawing with traditional tools such as a pencil.

In this work we have addressed this problem by analyzing multiple strokes and replacing them with a single stroke that makes a reasonable estimate of the designer's intention. Our solution recursively subdivides space stopping where either there is only a single stroke, or several strokes that have a proper ordering using Principal Component Analysis. The subspaces are then reconnected, and the orderings are joined to create the control points of a single B-Spline curve. The resulting curve is very noisy due to the multitude of strokes. A multi-resolution technique that makes use of reverse subdivision has been used to fit a smooth B-Spline curve.

Keywords Curve Fitting · Sketch-Based Interface · Small Strokes · PCA · Reverse Subdivision

R. Pusch
University of Calgary,
E-mail: rapusch@ucalgary.ca

F. Samavati
University of Calgary,
E-mail: samavati@cpsc.ucalgary.ca

A. Nasri
American University of Beirut,
E-mail: anasri@aub.edu.lb

B. Wyvill
University of Victoria,
E-mail: blob@cs.uvic.ca

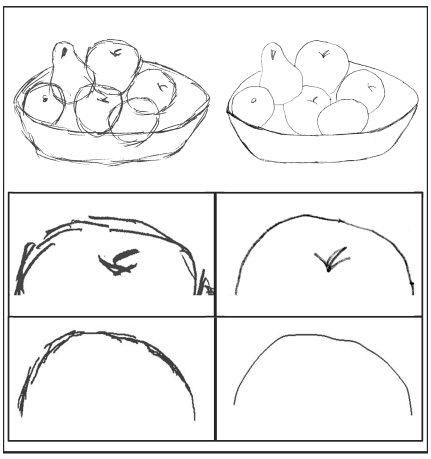


Fig. 1 A traditional hand-drawn example at two different stages. The last row shows a result from our system.

1 Introduction

Sketch-based interfaces are becoming a popular way for describing complex tasks to modeling and animation systems. At an early stage of design, artists and designers often use pencil or pen and ink sketches to make prototypes of their ideas. Sketching a curve is the basic primitive for an artist to build preliminary drawings, yet current sketch-based systems restrict the user to curves formed by a single stroke of the input device. There exists right from the start a disconnect between the sketching methodologies employed by traditional pen-and-paper artists and the sketching interfaces in sketch-based systems. It is common for a designer to use many small strokes which define the general shape of a final curve (see Fig. 1). This disconnect cripples the current interface, and initial sketches performed in this manner will often be drawn on paper, foregoing sketch-based systems entirely. We propose an alternative input method for 2D curves which uses many small strokes. Our goal is to

bridge the current gap between pen-and-paper sketching techniques and the sketch-based interface.

Using our method, designers can freely input many strokes in any order and direction to draw complex curves (i.e., with high curvature or multiple turning points). The final output of this system is a B-Spline curve which can be used by many applications. Although it is possible to extract a point cloud from all of the input strokes, we use the local ordering of points in each stroke for more efficiency and to better approximate the input strokes. Our method also has the advantage of using *reverse subdivision filters* that automatically generate a local least-squares minimizer for B-Spline subdivision curves without solving any linear system.

The rest of the paper is structured as follows: some related works are discussed in section 2. The proposed methodology, described in section 3, provides some background on principal component analysis, and describes how they are employed to solve the multiple strokes problem. Some implementation details are presented in section 4 and results are given in section 5. Finally section 6 describes some conclusions and possible future work.

2 Related Work

Igarashi [4] has implemented a system for 3D freeform design which requires nice curves input in 2D. Cherlin [3] models 3D shapes with few strokes (i.e., curves), but each curve must be drawn without lifting the input device. These systems provide evidence that there is a need for B-Spline curve input, but the difficulty with the interface limits their applicability and requests that users draw difficult and complex curves in a somewhat unnatural way.

Baudel [2], Kara [5], Zheng [12], Michalik [7] and others have discussed techniques for using strokes to modify existing strokes. For example, Zheng [12] deforms a curve, locally matching it with another curve. This becomes computationally expensive due to knot removal techniques and cannot guarantee smooth transitions. Michalik [7] starts with few control points, fits a curve, and then adds more control points locally until the deviation from the input curve is sufficiently small.

While these methods are close to our objective, there is still a requirement that a base curve be given that can be later modified. It is desirable that the system makes no assumption about the shape of the curve until all strokes have been input. Additionally, many of these systems solve an optimization problem iteratively, which is costly. Kara [5] also employs a technique that accepts many strokes but the overall shape of the curve is assumed to be simple (small curvature and no turning point). For example, Kara's method will handle the leftmost set of strokes shown in Fig. 7 but not any of the other more complex shapes shown to the right.

Each of the above methods either solves a linear system for fitting a curve, or minimizes some function with iterative optimization. This can be time consuming and difficult to implement. Schmidt et al. [11] provide a solution for connecting disjointed strokes or smoothing several minimally self-intersecting (i.e., overlapping) strokes. However, their approach uses 2D variational implicit curves which provide a framework for these tasks, but are not as applicable to a wealth of systems as B-Spline curves. Kegl et al. [6] discuss a method to skeletonize handwritten letters, a similar problem to ours, but they construct a Euclidean graph where a given vertex may have a degree of more than two, which does not translate well to B-spline curves.

Bartels and Samavati [1,9] construct multiresolution filters based on local least-squares minimization. As a result, they provide pre-calculated reverse subdivision filters that can be used for curve fitting in a very efficient manner. We use this approach for fitting the final curve in section 3.5.

3 Methodology of Approach

In this section, we outline some important notation, provide some background on PCA, and describe the various stages of our algorithm.

The scenario we consider is the designer or artist interactively sketching a set of strokes $(s_i)_{(1\leq i\leq n)}$ with a curve in mind, and we must find a B-spline curve that resembles the general shape of these strokes. Such a technique can be used as an important component for many sketch-based techniques ([4,3,11]) to build 3D models since their modeling techniques strongly rely on the structure of the input curves. Each stroke s_i includes an ordered set of points whose ordering is induced by the input device. However, the entire set of points

$$P = \bigcup_i s_i$$

do not have a global ordering since a designer does not have to sketch strokes in a specific order. Finding the global ordering is the first and major step in our work. Although P may be treated as a pure point cloud, we found the local ordering of s_i very useful.

One possibility for ordering the points is to sort them based on their x or y coordinate. This is problematic since this method not only depends to the coordinate system but can cause a many-to-one collision for the sorted value (x or y). To address these issues, we can use a linear transformation to map the points into a new normalized coordinate system whose first coordinate shows

the greatest variance of the points. Principal Component Analysis (PCA) is a simple technique for finding this transformation. To form PCA for $N_P = \{p_1, p_2, \ldots p_n\}$, a local neighborhood in P , let $p_i = (x_i, y_i)$. Then the covariance matrix can be defined by

$$C = \begin{pmatrix} \mathbf{x}^T\mathbf{x} & \mathbf{x}^T\mathbf{y} \\ \mathbf{y}^T\mathbf{x} & \mathbf{y}^T\mathbf{y} \end{pmatrix}$$

where

$$\mathbf{x} = [x_1 - \bar{x}, x_2 - \bar{x}, \ldots x_n - \bar{x})]^T$$
$$\mathbf{y} = [y_1 - \bar{y}, y_2 - \bar{y}, \ldots y_n - \bar{y})]^T .$$

In the above equation $\bar{x}$ and $\bar{y}$ show the average of x and y coordinates of N_P. The principal components can be found by determining the eigenvectors of the covariance matrix. Let $\lambda_1 \geq \lambda_2$ be the sorted eigenvalues of the covariance matrix, and let V_1, V_2 be the associated eigenvectors. Then, V_1 is the direction of greatest variance in N_p (principal component).

Finding the size of N_P is an important issue. A very small size neighborhood can capture variations of the strokes but it reduces the efficiency of the box connection algorithm and makes many small pieces of local orderings. These small pieces must be properly connected to form the global ordering (bottom-up approach). In fact, the orderings of s_i will be useless in this case. In addition, when using a bottom-up approach, it is hard to differentiate between stroke features and noise. On the other hand, a very large neighborhood generates an efficient method but it may lose some important features. For example, for a "Z"-like set of stokes, if we consider the entire neighborhood as N_P, we cannot obtain a reasonable ordering by projecting points to the x-axis, which is the principal component.

In our method, we make use of the ordering of s_i and start with a bounding box containing all the strokes. Then, we adaptively subdivide the boxes until each individual box contains either a single stroke, or several strokes that have a proper ordering using PCA.

Our goal is to ensure that each box contains a set of points simple enough to obtain a local ordering. First, we recall that each stroke s_i has an induced ordering from the input device. Therefore, if a box contains only one stroke, it needs no further subdivision as we have already obtained a suitable ordering. We may also be able to create an ordering if the PCA of the box, and each stroke, is similar enough (see Fig. 2).

However, if a box contains many strokes and the PCA determines we cannot yet find a suitable ordering, we must subdivide this box. We can choose to subdivide the box into two horizontal halves, into two vertical halves, or into four equal quadrants depending on local trends of the data. We continue to recursively and adaptively subdivide each box as necessary until all boxes are simple enough to obtain a local ordering.

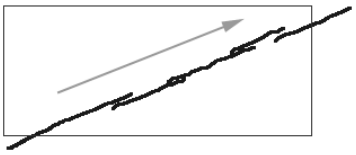


Fig. 2 Strokes contained in a simple box. Each stroke will have a similar principle eigenvector, which is also similar to the eigenvector of the box's points as a whole.

Once this has been accomplished, we must connect the boxes in a way which is both visually acceptable to the user, and which matches his original intentions for the curve. A clever subdivision of the original bounding box is important for this step, as connecting the boxes becomes difficult if there are many small boxes, each containing small, nearly straight features of the curve.

Once a global ordering on the boxes is obtained, we simply connect each local ordering to obtain a single ordering P_0 which contains every point drawn by the user. We then fit a quadratic B-Spline curve to P_0 using a reverse Chaikin subdivision technique (section 3.5).

3.1 Simplicity of a Box

A given box b_i will contain a number of strokes s_i (see Fig. 2), some of which could extend to other boxes. The PCA provides us with a metric for measuring how close the data in the box is to a straight line segment, and how close each stroke is to the overall trend in the box. If the points are roughly distributed along a line, the first eigenvalue of their covariance matrix will be relatively larger than the second eigenvalue.

We first compute the principal vector of the box b_i, then each of the included strokes s_i. If these vector directions are close enough, and the data in the box is approximately straight (determined by the ratio λ_1/λ_2), then the box is considered simple enough to obtain an ordering (see Fig. 2). This value of this ratio must be greater than a certain threshold, because only data that shows a dominant trend is considered simple enough to obtain an ordering. When the ratio is below this threshold, no dominant trend can be found and the eigenvectors of random strokes may be similar through coincidence. The situation is also considered simple if the box contains only one stroke, for reasons previously stated.

3.2 Conditions for Subdividing a Box

When choosing a subdivision method for a box, we want the sub-boxes to be as simple as possible so we can stop

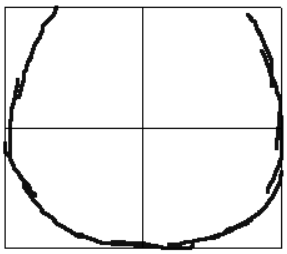


Fig. 3 The data shows no horizontal or vertical trends, so quad subdivision is best.

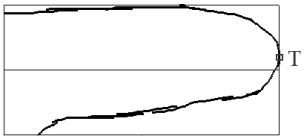


Fig. 4 The data is largely horizontal, so we choose to subdivide it into two horizontal boxes. The vertical case is analogous. This will often isolate the key components, making each sub-box a simple case. The turning point T is also shown.

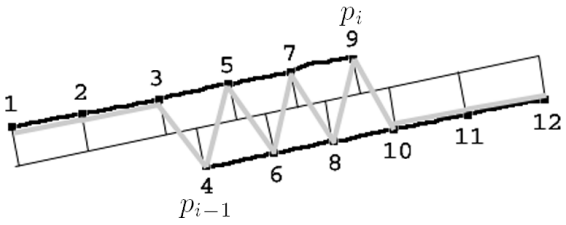


Fig. 5 Two overlapping strokes are projected onto the box's eigenvector to obtain an ordering on the points. This recreates a jaggedness in the control polygon. p_{i-1} and p_i are also shown.

subdivision early. The subdivision method is chosen according to the trend of the data in that box. That is, if the data in a box is largely horizontal, then the box is subdivided into two horizontal halves (see Fig. 4). By subdividing in this way, we will often isolate the large, simple components representing a single trend into the sub-boxes. For example if the box in Fig. 4 were cut vertically rather than horizontally as shown, we would keep the problem area, near to the turning point T, intact in a sub-box and the situation would not be simplified. When the data shows no particular trends (indicated by a low λ_1/λ_2 ratio for a box), or the trend shown is largely diagonal, we choose to subdivide into four quadrants (see Fig. 3).

Additionally, very straight strokes or strokes that contain a lot of points should be given more influence in deciding how to subdivide a box. This allows the most dominant strokes in the box to dictate how a box is subdivided, as these strokes will play the largest part in determining when we have reached a simple case. Thus, when determining the nature of the data, we perform a weighted average of the angle between the x-axis and each stroke's primary eigenvector. We weight each value with the λ_1/λ_2 ratios and the number of points on the stroke. The value of this weighted average is compared to thresholds to create one of three distinct cases for subdivision;

1. two vertical halves,
2. two horizontal halves,
3. four equal quadrants.

Lastly, very thin boxes, either vertical or horizontal, are undesirable, as the trend in such boxes always dictates we subdivide to create two even thinner boxes. This causes a case of near-infinite recursion, creating dozens of thin boxes which we are quite difficult to reconnect. Thus, when we encounter an extremely thin box, we will always choose to subdivide the box in such a way as to reduce the aspect ratio of the sub-boxes, regardless of the supposed data trends in the box.

3.3 Ordering Inside a Simple Box

Once a box has been declared "simple", we must find a local ordering on the points in this box. There are only two cases to handle. If the box contains a single stroke, we take the ordering created by the input device.

If the box contains many strokes, we simply project each point along the box's principle eigenvector and sort the points according to their projected distance. This will create a local ordered list of points within the box. When there are overlapping strokes, however, this will create a jaggedness in the local ordering which has proven difficult to denoise (see Fig. 5).

One approach to solve this issue is to replace the points responsible for jaggedness with points equally distributed along an *average curve*. We form this curve with the weighted average of the pieces of each stroke involved in the overlap (see Fig. 6). This method allows us to keep curvature information about the desired curve during stroke overlap.

To do this, we first must determine all regions of overlap. Each region contains jaggedness that will require fixing. We find each stroke's start and end point in the box, and sort them by their distance along the primary eigenvector of the box. Denote each point in the sorted list of these "event points" as p_i. Note that p_{i-1} comes before p_i in the local ordering for each i.

We then process the list left to right, and maintain a list of "active" strokes by adding a stroke when we reach its starting point and removing a stroke when we encounter its endpoint. Each time we add or remove a stroke from the active stroke list by encountering p_i (and there were at least two strokes in the list before the change), there is

a region of overlap which needs to be fixed. Specifically, each point in the ordered list between p_{i-1} and p_i is part of a jagged sequence, and each stroke in the active stroke list is contributing points to this jagged sequence.

In order to fix this jaggedness, we must remove all points between p_{i-1} and p_i in the ordered list and replace them with a non-jagged sequence. We want this sequence to contain roughly the same density of points as the surrounding strokes, and we want the sequence to represent an average of the strokes contributing to the jaggedness. We use a B-spline representation to achieve this result.

For simplicity, consider the case where there are only two overlapping strokes (see Fig. 6). We create two B-spline curves using the points of each stroke contributing to the overlap (i.e., the points on each stroke between p_{i-1} and p_i); denote them as $T(u)$ and $B(u)$. Let t and b be the number of control points on $T(u)$ and $B(u)$ respectively, and let m be the maximum of t and b. We will then sample, with equal distribution, m points along the following B-spline curve:

$$A(u) = \frac{t}{t+b}T(u) + \frac{b}{t+b}B(u)$$

We will then replace all the points in the local ordering between p_{i-1} and p_i with these m points. We use a weighted average of the two B-spline curves because it is reasonable to assume that a stroke with more points indicates a more confident stroke from the user, which should have a bigger impact on the final curve. We also only sample m points to maintain a similar density of points for the overlapping sections. This algorithm is also easily extendable to an area of overlap containing an arbitrary number of strokes.

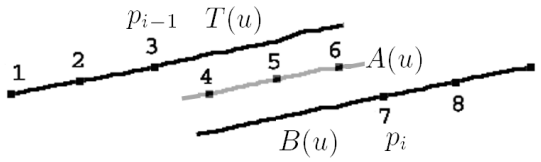


Fig. 6 The pieces of the strokes between p_{i-1} and p_i are overlapping. We replace the points in that range with points equally distributed on the "average curve" $A(u)$. This ordering has no jaggedness and is appropriately dense.

Once we have fixed all the overlapping strokes by moving the points to an appropriate "average curve", the ordering in the box is complete.

3.4 Global Ordering on the Set of Boxes

In order to connect boxes from a set $B = \{b_1, b_2, \ldots b_n\}$, we choose any box b and add it to an empty list of boxes L. We find the last point in the local ordering of b, determine the global stroke to which this point belongs, and "follow" the stroke along its induced ordering to the next point. If this point is in a new box, this becomes the next box in our ordering; add it to the end of L. We continue in this way until we are unable to follow each successive box to a new box.

We then re-examine the original box b and attempt to "follow" the *first* stroke in the box backwards in a similar way. Each new box that we find is connected by adding it to the front of L. When we follow this trail to its completion, we have connected all boxes for our global ordering as listed in L.

3.5 Fitting a Final Curve

Once we have an ordering on the boxes, we can simply connect each box's local ordering together to form a large list of points P_0, which can be interpreted as a very noisy approximation of the user's intended curve. Simply using these points as control points for a B-spline curve does not generate good results, as there are far too many points, and noise from the input device remains in the data.

One possible approach to this problem is to employ a least square B-spline fitting [8]. This approach requires finding the solution to a large linear system. As an alternative, we have chosen to use a multiresolution representation based on *reverse subdivision* [1,9]. In these works, a "local" least-squares approach (minimizing the error in a local neighborhood of the data) is used to find a set of coarse points of half size along with a set of details. Due to the regularity (subdivision and its reverse), it results in some simple decomposition filters; therefore, by using these pre-determined filters, we can find a set of coarse control points without solving any linear system. Notice that by changing the size of the local neighborhood, different filters are determined. Narrower filters are easier to implement but generate larger least-squares error.

In our implementation, for sake of simplicity, Chaikin subdivision (uniform quadratic B-spline curve) has been used. Any other uniform B-spline can be similarly used.

The reverses Chaikin subdivision can reduce the number of control points and the noise of the curve [9]. As general reverse subdivisions, this technique will decompose a set of points into a set of coarse points of half size and a set of details. Since the details are largely filled with noise in our case, they are simply discarded. The coarse points are then interpreted as control points for a quadratic B-spline curve.

We have used two different reverse Chaikin techniques. The first one is the filter with width four

$$q_j = -\frac{1}{4}p_{i-1} + \frac{3}{4}p_i + \frac{3}{4}p_{i+1} - \frac{1}{4}p_{i+2}$$

where each q_j is a coarse point and each p_i is a fine point, and the step size of i is two. All results shown were made by a filter with width eight

$$q_j = \frac{3}{40}p_{i-3} - \frac{9}{40}p_{i-2} - \frac{1}{40}p_{i-1} + \frac{27}{40}p_i + \frac{27}{40}p_{i+1} - \frac{1}{40}p_{i+2} - \frac{9}{40}p_{i+3} + \frac{3}{40}p_{i+4}$$

We have found that the wider filter creates a better curve.

4 Implementation

In this section, we discuss specific implementation details, including threshold values and details about connecting the boxes to obtain a global ordering.

4.1 Deciding Box Simplicity

As discussed in section 3.1, a box may be considered simple enough to obtain a local ordering if the strokes are "similar". When checking for simplicity in the implementation, each box is assumed to be simple, and then each stroke contained in the box is checked. Strokes containing few enough points are disregarded; our particular input device sensitivity has caused us to set this threshold to 70. We disregard these strokes entirely as short strokes do not overly affect the curve structure and therefore do not have a significant ordering. Otherwise, if we come across a stroke whose primary eigenvector forms an angle 40 degrees or more with the box's primary eigenvector, or if a stroke has a λ_1/λ_2 ratio of 10 or less, we will mark the box as not simple. The first condition shows a stroke too diverse from the rest of the box, and the second condition indicates that some stroke in the box is too curvy to obtain a meaningful ordering through projection.

4.2 Choosing a Subdivision Method

Each stroke's contribution to the weighted angle is weighted by two factors. The first factor is the number of points on the stroke. This ensures that short, divergent strokes have significantly less effect on the curve. The second factor is the ratio λ_1/λ_2 which allows straighter strokes to have more of an impact on the subdivision choice. A box is subdivided into quadrants if the average λ_1/λ_2 ratio of the strokes is less than 50, or if the weighted angle is between 30 and 60 degrees. Otherwise, the box is subdivided into vertical halves if the weighted angle is larger than 60 degrees, and into horizontal halves if the weighted angle is less than 30 degrees.

When we encounter a box with an aspect ratio of 5 or greater, we choose to subdivide in such a way as to lower the aspect ratio of the sub-boxes, independent from the data trends in the box.

4.3 Connecting Boxes

When connecting some boxes, "following" the original stroke based on the order induced by the input device brings about problems when some strokes are drawn in different orders from others. This problem is attacked in two steps in the implementation. Firstly, when attempting to follow a stroke "forward" out of a box, we normally find the last point along the local ordering, find this point's main stroke, and then look at the next point higher in the ordering along the main stroke. If there is no next point or the next point is in the same box, then we have reached a dead end. We will then try finding the previous point lower in the ordering along the main stroke. This helps us when some strokes are drawn right-to-left; the point we are looking for to continue our connection is often "backwards" to our assumption.

Secondly, it is also possible that loops can occur, so that box A leads to box B, then back to box A. This is due to the local ordering in box B; the projected points on the eigenvector are in the reverse order compared with the original data. When this happens, attempting to follow box B "forward" is confusing, because "forward" is a different direction along the strokes for the local ordering and the global ordering induced by the input device. Therefore, when such a loop is detected, the local ordering in box B will be reversed and the system will attempt to reconnect B with the new ordering.

Finally, due to floating point error and other local instabilities of the input device, it may not be possible to follow the very last or the very first point in the local ordering out of the box. When it is not possible to follow the stroke found at one of the extreme endpoints, a move is made towards the median of the local ordering and will stop when a point on another stroke is discovered. An attempt is made to continue by following the new stroke out of the box. This process is continued until the connection is established. Either we will find another box to connect from some other stroke, or else none of the strokes in the box can be followed in the attempted direction. This implies that there are no more boxes with which to connect in this direction.

5 Results

Fig. 7 illustrates results generated by our system. The strokes input by the user are shown, as well as the final curve generated. The left-most figure can be achieved through a variety of approaches, but the remaining figures have non-trivial curvature and cannot be done with current methods.

The two right-most examples show efforts to handle curves with sharp features or figures made of more than one dominant curve. In these examples, we must directly indicate the end of each set of strokes in the application (for the cup-like shape there are three independent sets of strokes, and for the letter 'P', there are two).

6 Conclusion and Future Work

In this work, we addressed the problem of analyzing multiple strokes and replacing them with a single stroke that makes a reasonable estimate of the designer's intention. We adaptively subdivide the 2D plane until each sub-box is simple enough to obtain an ordering. We then connect the subspaces and their local orderings to obtain a rough estimate of the user's curve. We fit a B-spline curve to the data using Chaikin reverse subdivision.

This method provides a bridge between current sketch-based interfaces and traditional methods employed by pen-and-paper artists for initial sketch generation. Our system can handle sketches of varying degrees of curvature, including many changes in curvature. We believe that bridging this gap is crucial for the sketch-based interface, where users of our system can effectively sketch a wide variety of interesting curves.

Our box connection algorithm follows boxes if there are strokes connecting them. We did not implement the case that has sparse strokes with gaps because, as is evident in Fig. 1, in most artistic drawings there are no gaps in the strokes. The artist usually wants complete control over his conceptualization. However, as a solution for the sparse case, it is possible to maintain a data structure which contains a list of all neighboring boxes for any given box [10]. Ray casting techniques can be implemented to determine if there are more boxes to connect in the ordering, instead of immediately stopping when we are unable to follow a stroke to a new box.

Our current implementation struggles when there are sharp features in the desired curve (that is, the desired curve is not C^1 continuous everywhere). While it is possible to simply draw each smooth section of the curve individually and then connect them as separate curves later, as our results section demonstrates, it may be beneficial to support sharp features on a single curve.

Our implementation also cannot support self-intersecting curves, since each box has only one local ordering and at most one entry and exit point from the box. These assumptions are not true of a box containing a self-intersecting piece, as the box will have two entry and exit points, and each intersecting piece will require an separate ordering. This is a desirable trait of our system, but one which would require a different paradigm to accomplish.

References

1. Bartels, R.H., Samavati, F.F.: Reversing subdivision rules: Local linear conditions and observations on inner products. Journal of Computational and Applied Mathematics **119**(1-2), 29–67 (2000)
2. Baudel, T.: A mark-based interaction paradigm for free-hand drawing. In: Proceedings of the 7th annual ACM symposium on User Interface Software and Technology, UIST '94, pp. 185–192. ACM ISBN:0-89791-657-3 (1990)
3. Cherlin, J.J., Samavati, F.F., Sousa, M.C., Jorge, J.A.: Sketch-based modeling with few strokes. In: Proc. of the 21st Spring Conference on Computer Graphics (SCCG'05) (2005)
4. Igarashi, T., Matsuoka, S., Tanaka, H.: Teddy: A sketching interface for 3d freeform design. In: Proc. of SIGGRAPH '99, pp. 409–416 (1999)
5. Kara, L.B., Shimada, K.: Construction and modification of 3d geometry using a sketch-based interface. In: EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling, pp. 59–66. Eurographics (2006)
6. Kegl, B., Krzyzak, A.: Piecewise linear skeletonization using principal curves. IEEE Transactions on Pattern Analysis and Machine Intelligence **24**(1), 59–74 (2000)
7. Michalik, P., Kim, D.H., Bruderlin, B.D.: Sketch- and constraint-based design of b-spline surfaces. In: ACM Symposium on Solid and Physical Modeling, pp. 297–304. ACM (2002)
8. Piegl, L., Tiller, W.: The NURBS Book, 2nd Ed. Springer (1997)
9. Samavati, F.F., Bartels, R.H., Olsen, L.: Local b-spline multiresolution with examples in iris synthesis and volumetric rendering. In: Series in Machine Perception and Artificial Intelligence, Vol. 67, Synthesis and Analysis in Biometrics, chapter 2 (2007)
10. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann (2005)
11. Schmidt, R., Wyvill, B., Sousa, M.C., Jorge, J.A.: Shapeshop: Sketch-based solid modeling with the blobtree. In: EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling. Eurographics (2005)
12. Zheng, J.M., Chan, K.W., Gibson, I.: A new approach for direct manipulation of free-form curve. Computer Graphics Forum **17**(3), 327–334 (1998)

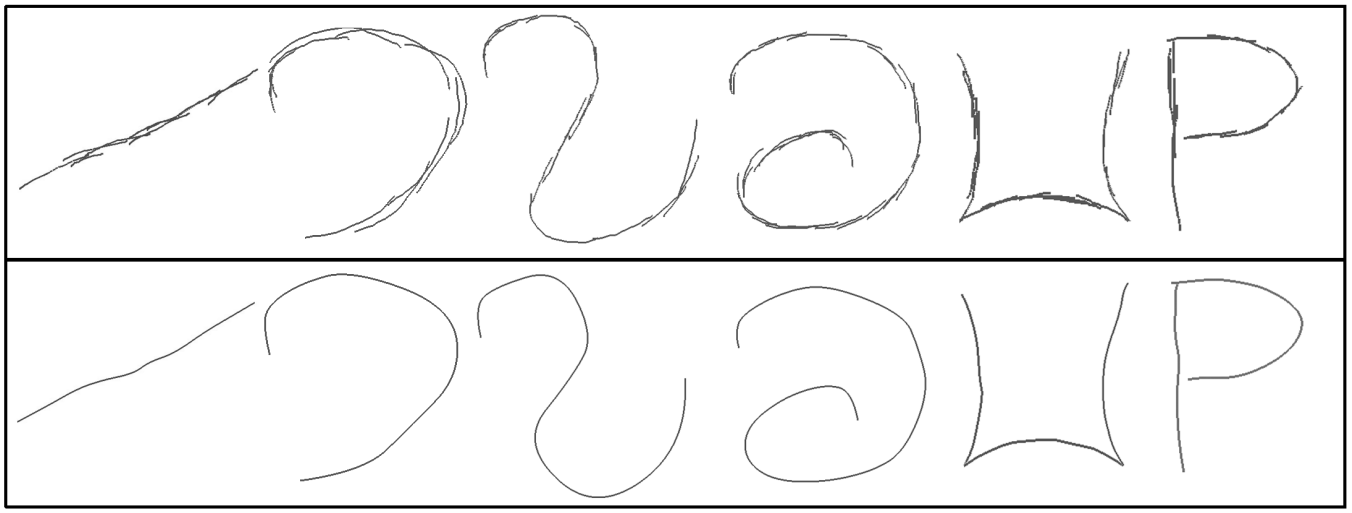
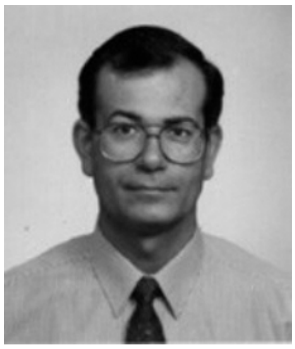


Fig. 7 Some results of our system. The top row shows stroke input, and the bottom row shows a quadratic B-spline curve which approximates the strokes. The shapes may have interesting curvature changes which are limitations of other systems.

R. Pusch is a Masters student in the Department of Computer Science, University of Calgary. Richard graduated with a B.Sc. in Computer Science, with a minor in Pure Mathematics, in December 2005, and began his postgraduate work in September 2006. His current research interests in computer graphics include Sketch-Based Modeling, Multiresolution and Wavelets, and Surface Modeling and Deformation.



A. Nasri is a professor in computer graphics at the American University of Beirut. He received a Ph.D. in Computer Graphics from the University of East Anglia in 1985. He was a research visitor at various universities such as MIT, Arizona State University, University of Calgary, Purdue University, Brigham Young University, Seoul National University, and Cambridge University. With Malcolm Sabin he co-edited a special issue of the Journal of Visual Computer on Subdivision surfaces, 2002. Since 1982 he has been involved in promoting subdivision surfaces and its use in computer graphics, geometric modeling, and animation. His research interests also include Digital Arts, and the use of Computer Graphics in Education.

F.F. Samavati is an associate professor in the Department of Computer Science, University of Calgary. He is also an adjunct associate professor in Computer Engineering Department, Technical University of Lisbon, Portugal. He received his Ph.D. degree from Sharif University of Technology in 1999. He was a research visitor at University of Waterloo in 1997. Dr. Samavati's research interests are Computer Graphics, Geometric Modeling, Visualization, and Computer Vision. He has authored more than 40 research papers in Subdivision Surfaces, Sketch-Based Modeling, Multiresolution and Wavelets, Surface Modeling and Scientific Visualization.

B. Wyvill was a professor at the University of Calgary for 25 years with research interests in implicit and sketch-based modeling and also non-photorealistic rendering. Recently he moved to the University of Victoria to take up a tier one Canada Research Chair.