EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling (2005)
Takeo Igarashi, Joaquim A. Jorge (Editors)

Sketching Free-form Surfaces Using Network of Curves

Koel Das Pablo Diaz-Gutierrez M. Gopi
Department of Computer Science,
University of California, Irvine

Abstract

This work addresses the issue of generating free-form surfaces using a 2D sketch interface. As the first step in this process, we develop a methodology to sketch 3D space curves from 2D sketches. Since the inverse projection from 2D sketches to 3D curve or surface is a one to many function, there is no unique solution. Hence we propose to interpret the given 2D curve to be the projection of the 3D curve that has minimum curvature among all the candidates in 3D. We present an algorithm to efficiently find a close approximation of this minimum curvature 3D space curve. In the second step, this network of curves along with the boundary information are given to the surface fitting method to generate free-form surfaces.

1. Introduction

Modeling by sketching has a fundamental problem of interpreting the 2D sketches of curves and surfaces as a 3D entity. Most of the real world objects have non-planar 3D curves and these curves are the fundamental component of surface generation [IMT99, ZHH96] for 3D models. Space curves are also used as input for specifying implicit surfaces and also for manipulating and modifying object deformations [SF98, IH03]. The problem of finding the 3D space curve from a 2D sketch is mathematically indeterminable, as it has many possible solutions; It is well known that the fundamental difficulty is in choosing the appropriate one. Most of the sketch-based modeling tools try to avoid addressing this fundamental problem directly due to its complexity. These systems provide a fixed canvas to the user to draw the 2D curve, thus fixing the 3D curve to be on the canvas. They provide tools to change the shape of the canvas to enable the user to draw non-planar curves [IITS04]. One of the clever uses of this canvas method was by Igarashi et al. [IMT99] where the system provides a planar canvas first and the user starts with a planar closed curve. A rotund object is generated using that planar closed curve and from then on, the generated object becomes the canvas for all future curve drawings. Often, curves that are not drawn on the canvas are interpreted as gestural inputs for viewpoint or object manipulation. In spite of all the successes that have been achieved in the field of sketch based modeling, the fundamental problem of space curve determination still remains to be a challenging one.

In our paper, we attempt to provide an intuitive and mathematically sound insight into this problem of 3D space curve determination from a 2D curve. We propose that the 3D curve corresponding to the given 2D curve to be the one that minimizes the maximum normal curvature in 3D. Cur-

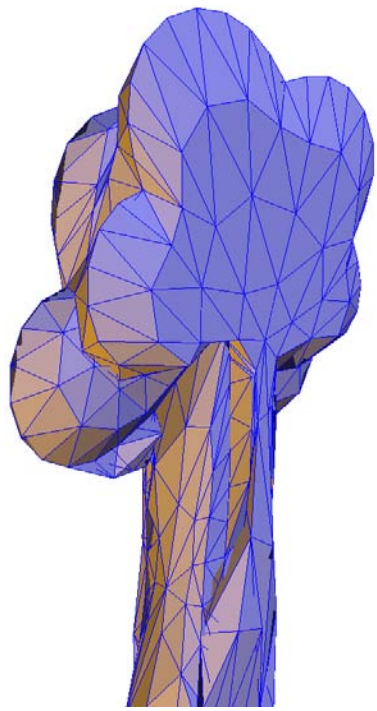

Figure 1: *A sample model generated by the system.*

vature minimizing interpolation approaches have been previously used in the study of visual perception to create seamless color smoothing for overlapping multi-projector displays [MS05], indicating a relationship between curvature minimization and perception of desirability and intuitiveness. In this paper we take a similar approach to address the following problem: given a 2D curve, the projection parameters, and the depth range, find the 3D space curve within the given depth range that has the minimum curvature and matches with the 2D curve under the given projection.

The second issue we address in this paper is the generation of surfaces using the above curvature minimizing space curves. We sketch the network of planar and non-planar input curves and specify the patch connectivity. Our system automatically generates a free form surface that fits these network of curves. Once an initial 3D model, based on the input sketch is presented, the user could modify it to get the desired shape. In our current exposition, we stress more on the proof-of-concept implementation of geometric principles and less on the user-interface design.

Specifically, following are the main contributions of this paper:

1. We prove the existence of the curvature minimizing space curve for an orthographic 2D projection of a polynomial curve.
2. Given only a sequence of 2D curve points as input, we present an efficient algorithm to best approximate the curvature minimizing 3D space curve, from that viewpoint. We use Laplacian filter effectively along with our approach to achieve the desired result of curvature minimizing depth interpolation.
3. We generate smooth free form surfaces using minimal user input using an iterative method having fast convergence. We present a proof-of-concept system that fits a smooth surface given the boundary curves that are sketched using our system.

2. Related Work

There has been a lot of ongoing research on generating 3D models using a sketching interface. The different endeavors in the field of 3D curves focus either on *curve manipulation methods* or on *2D to 3D curve generation methods*. The approach mentioned in this paper belongs to the latter category.

Existing space curves can be manipulated indirectly in many commercial packages like Maya, 3DStudioMax and most other CAD tools, by modifying the curve parameters such as spline control points or knot values. Spline curves can also be manipulated directly as described in [FB93, GA98]. Modifying existing curves by over-sketching was initially presented by Baudel [Bau94]. This approach has been adopted for curve modification and manipulation by several other researchers in this field [CMZ*99, FRSS04]. Cohen et al. [CMZ*99] used the correlation between the curve and its shadow as a visual cue to help the user compute and modify the shape of the 3D curve. Karpenko et al. [KHR04] take a multi-view sketching approach using epipolar lines as cues to deduce shape information. Their method uses perspective projection and can determine the 3D values only when drawn from two different view points. The epipolar geometry approach works for simple cases but fails to give desirable results for more complex objects due to the nature of their algorithm.

In the recent past, there has been some research devoted to extract depth information from 2D curves and strokes directly. Most of the existing literature on space curve generation relies on the domain specific knowledge. The edge-vertex graph method has been utilized to extract the depth information in systems best suited for architectural designs and CAD type applications in [LS96, SC04]. Furthermore, [LS02] use an interesting domain specific learning based approach using geometric correlation to deduce the 3D model that is again more suitable for CAD tools. The underlying assumption behind this learning technique is that the input sketch comprises of straight lines and thus does not account for free-from curves. It also follows from the paper that the human ability to perceive the 3D information from 2D sketch is based on prior knowledge about the drawn object or scene and human perception fails to reconstruct random objects. Learning based methods are also non-interactive and are computationally expensive.

One of the earliest example of constructing free-form space curves is the 3-Draw system [SRS91] that extracts the 3D coordinate information by using a tracker-based system. A curvature minimizing approach was followed by [PK89]. Ijiri et al. [IITS04] utilize the input sketch to generate a curved canvas where the drawn curve essentially becomes an extruded surface from a given viewpoint providing the user the capability of drawing non-planar curves on that surface. Tolba et al. [TDM99] use the approach of oriented projective representations of 2D points. This allows "3D-like" perspective viewing, object manipulation and rendering of the 2D points, but they do not generate actual 3D values from 2D points.

To the best of our knowledge, there is no literature on research that performs domain independent mathematical treatise to generate intuitive free form space curve. Our paper attempts to address this issue by using curvature minimizing depth interpolation. We present an interpolation method to generate an artistic and expressive 3D curve from a *single* 2D curve input. It is obvious that one cannot provide the exact curve that user has in mind, just from a single 2D sketch. Hence, any curve *manipulation* technique can be used thereafter to modify our computed 3D curve. The method discussed tries to empower the user by simplifying the user-interface, and shortening the system learning time of the user as well as the and overall time to sketch the object.

Recently, some papers have been presented on fitting sur-

faces to a network of curves. Teddy [IMT99] is one of the exemplary works in the field of free-form surface generation using polygonal representation. Michalik et al. [MKB02], on the other hand used a parametric representation of surface using a constraint based design. Volumetric representation is also used effectively by researchers [ONNI03] which has the advantage of relaxing the topological limitations on surface generation. Furthermore, recent research on surface generation techniques use implicit surface representation. Notably, Tai et al. [TZF05] obtain an analytical convolution surface from sketched silhouettes. In [SWZ04], Schaefer et al. make clever use of lofting techniques to create subdivision surfaces from curve network.

Our system generates free form surfaces from sketched network of curves by using a simple *topological triangulation* followed by surface fairing.

3. Curves and Networks of Curves

We repeat the problem statement for the sake of completion. *Given a 2D curve, the orthographic projection parameters, and the depth range, the problem is to find from all the possible 3D space curves within the given depth range and that matches with the 2D curve under the given projection, the curve that minimizes the maximum curvature.* The given 2D curve is assumed to be a polynomial curve. Since any polynomial basis can be converted to and from a Bernstein basis, we can convert the given polynomial curve into a 2D Bezier curve. We state, prove and use a theorem given in the appendix involving the control points of Bezier curves to come up with a solution for the stated problem.

Theorem 1 Given a 2D Bezier curve with a known depth range, the depth of the control points of curvature minimizing 3D Bezier curve is monotonically and equally spaced within the given depth range.

Using this theorem to construct the 3D curve from the 2D sketch has seemingly a drawback: It can produce only monotonic, low curvature curves. As we go from 2D to 3D, we must constrain the problem to be able to choose one solution from an infinite number of possibilities. Monotonicity from a given view point is a required constraint for finding a curvature minimizing curve. At the same time, the generated curve can be non-monotonic in every other viewing direction and hence this is not a limitation. Further, curvature is minimized within certain constraints - the given depth range and the 2D projection of the curve. Smaller depth ranges provide less space to reduce the curvature. Hence arbitrarily high curvature curves can be created by using appropriately small depth ranges (see Figure 3, top right image).

3.1. From 2D curves to 3D curves

The 2D curves drawn in sketching applications are not usually Bezier curves, but a sequence of edge connected points. Hence, we have to adapt our stated theorem for the depth extraction for a generic 2D curve. One way to adapt our result on curvature minimizing depth interpolation is to fit a Bezier curve to the input point sequence and interpolate the depth of this 2D Bezier curve. This is a computationally expensive operation and most importantly, the error in curve fitting will provide a distracting and unexpected feedback to the user.

In this section we provide an alternative method for finding the depth of the sketched 2D curve points that approximates the minimum curvature 3D curve. We use the following fundamental observation of the 3D Bezier curve chosen in the previous section (but not true in a generic Bezier curve): since the depth values of the control points are equally spaced, the closest point on the curve from any control point has the same depth value as the control point. Hence if we identify these *key points* on the curve that are closest to the control points and space their depth values equally, we will have the first approximation of the space curve. Since we do not have these 3D Bezier curve control points and hence of its 2D projection, identifying even the correct *number of key points* is difficult, let alone the actual key points. Even though we do not have an explicit Bezier representation of the 2D sketched curve, the number of critical points in the curve is the lower bound on the number of control points of the hypothetical Bezier representation of the same curve. The critical point need not be the closest curve point (key point) to the corresponding control point. The fundamental source of *approximation* in our algorithm for curvature minimizing depth interpolation comes from the fact that we assume that these critical points are key points.

The second source of approximation comes from the fact that, there need not be a critical point corresponding to every control point. Hence all the required control points cannot be found by just one sequence of critical points. We solve this issue by subdividing the original 2D curve at the initial critical point (*key point*) locations. If we perform a de-Casteljau subdivision of the Bezier curve at the *key points*, the above theorem still holds good for each of the subdivided curve segments. In this process, we increase the total number of control points (sum of the control points of the subdivided curve segments) and hence the possibility of more critical points. Using this observation, we repeat the process of finding the critical points recursively within each 2D segment between the initial sequence of critical points up till a threshold in terms of linearity of the curve segment. Since we assume that all these critical points are representatives of *key points*, we assign equally spaced depth values within the range to the critical points. The depth of all other points on the 2D curve is computed by linearly interpolating the depth between the critical points based on the curve length parameterization.

The resulting 3D space curve, though a very close approximation of the curvature minimizing curve, has non-smooth, high frequency transitions of the depth values across the

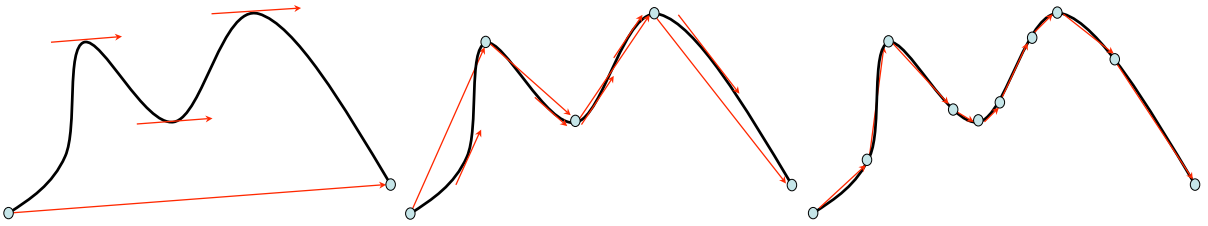

Figure 2: *Illustration of our approximation algorithm to generate curvature minimizing space curve. At every iteration the critical points of the distance function from the line joining the first and last points of the curve segment is chosen for further curve subdivision. The recursion ends when the curve segment can be approximated well by the line joining its end points.*

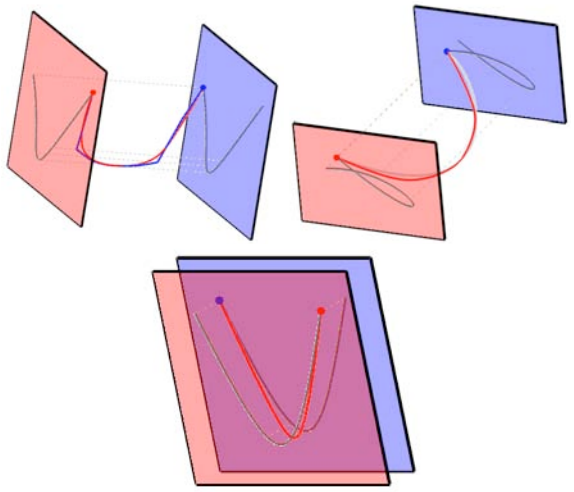

Figure 3: ***(Top left)*** *Effect of filtering. Blue curve shows the interpolation before filtering and the red curve shows the curve after filtering.* ***(Top right)*** *Reconstruction of space curves using our approximate method for different Bezier curves. The actual curvature minimizing Bezier curve as suggested in Theorem 1 is shown in gray. Our approximate curve follows the actual curve closely. Notice in that even from a viewpoint where projections are self intersecting in 2D, the reconstructed curvature minimizing curve follows the actual Bezier curve.* ***(Bottom)*** *High curvature curve obtained with our system.*

critical points. We use the Laplacian filter to obtain a final smooth space curve. This filter is extremely fast and simple, as required by interactive sketching applications. But it is a known fact that it produces a shrinkage effect for geometric primitives. In our algorithm, the shrinking of curves is prevented by keeping the depth at the curve end points constant. We apply a few iterations of this smoothening filter *only* to the interpolated depth values of all the interior points on the curve, as $z_i\prime = z_i + \lambda(\Delta z_i)$ where $\Delta z_i = ((z_{i-1} + z_{i+1})/2) - z_i$ and $\lambda < 1.0$ for a low-pass filter, . This produces an excellent approximation of the curvature minimizing curve Figure 3.

The summary of the algorithm for finding the curvature minimizing curve is given as pseudo-codes in Algorithms 1 and 2. An illustration of this algorithm is shown in Figure 2.

Algorithm 1 ReturnCriticalPoints(Curve C)

{Function: Find the Critical Points of the 2D curve C.}
{Given: Sample points of a 2D curve C}
L = Line between the first and the last sample points of C.
CS = Sequence of all the critical points of the distance function of C from L.
R = Subset of CS, that is within a user-defined threshold distance from L.
$CS = CS - R$.
if $CS == \phi$ **then**
  return; {Terminating condition.}
end if
$CriticalList = CS$
$CheckList$ = {First point of C} $\cup$ CS $\cup$ {Last point of C}
for every curve segment C_i between consecutive points in $CheckList$ **do**
  CriticalList = CriticalList $\cup$ ReturnCriticalPoints(C_i) {
  Insert into CriticalList, the critical points of C_i in order of their appearance in C.}
end for
return CriticalList;

3.2. Generation of Curve Networks

Our system can be used to generate network of curves comprising of both planar and non-planar space curves, using the curvature minimizing curve interpolation as explained in the previous section. Let the end points of a curve be called the reference points. Two important interface/capabilities of our system are key to enable users to draw a network. First is the ability to delete a curve and the second is the fact that any point on a (already drawn) curve or a plane (normal to the viewing direction) can be marked as a reference point to draw other curves. Thus any curve drawn can be part of the network and/or specify the reference points for other curves to be drawn. Curves that are not part of the network are later deleted. We briefly discuss the procedure to use these two features to draw a network of curves.

Algorithm 2 CurvatureMinimizingCurve(Curve C, Depth z0, Depth z1)

{Function: Curvature Minimizing Interpolation of Depth}
{Given: Sample points of a 2D curve C; depth values of the first and the last points of C, z0 and z1. }
CriticalList = {First point of C} ∪ ReturnCritical-Points(C) ∪ {Last point of C}
Let DepthInterpC be the points in C with unset depth values.
Interpolate the depth value linearly between z0 and z1 among all points of *CriticalList* and set their values in DepthInterpC.
for every curve segment C_i between consecutive points in *CriticalList* **do**
  Interpolate the depth value based on curve length parameterization and set their values in DepthInterpC.
end for
CurvMinC = LaplacianFilter(DepthInterpC)
return CurvMinC

As the first step, the user draws a curve on the plane perpendicular to the viewing direction. This curve can be rotated so that every point on the curve has different depths from the new viewpoint. Reference points can be specified on this rotated curve and a new space curve can be drawn connecting these reference points from any viewpoint. Since the end points and hence their depths from the given view point are known, these curves which are drawn in 2D, are interpolated in 3D using our curvature minimizing curve interpolation. The new curve can be part of the network and/or can be further used to specify more reference points. Later, all the curves that were used only to specify the reference points are deleted.

Thus the user interface is designed to accentuate the simplicity of the sketching environment where the user keeps drawing, rotates the plane and continues drawing till the desired set of curves are sketched. We still have not included the sanity check of the requirement of closed models. This will be integrated with the automated patch identification as a future work. The resulting curve network is the input to the surface generation technique described in the following section.

4. Surface Generation

Our system generates free form surfaces from a network of input curves. In our current implementation, once the user sketches the desired shape of the object in the form of outline curves, the connectivity of the lines that form surface patches must be given. Although this is currently done manually, we intend to automate the process of patch identification. Once the patch boundary connectivity is known, our system then generates a smooth free-form surface based on the given boundary curve network. In order to maximize the number of regular vertices (here degree 5) we first find the additional vertices in the interior of the patch and their connectivity and then calculate the geometric coordinates for these vertices. As an outline, our surface generation method consists of three steps: normal computation and interpolation, topological triangulation, in which we find the connectivity of the triangles in the surface patches, and finally surface fairing, where we compute the position of the introduced vertices. We briefly describe each of these steps below.

Normal Computation and Interpolation: Every corner vertex in the input closed boundary curve sequence has at least degree three. Hence a normal vector with a local coordinate system can be computed at these corner vertices. Then these coordinate systems are interpolated using quaternions at intermediate vertices along the curve. We use spherical linear interpolation (*SLERP* [Sho85]) to smoothly interpolate these local coordinate systems between the curve endpoints. In order to reduce the noise in the normals, the input segments which suffer from sharp transitions are smoothened.

Topological Triangulation: The input to this step is the boundary of a future surface patch, identified as a closed sequence of connected curves. Curves are represented as piecewise linear segments, or polylines. The output is a topology of the triangulation that consists of input vertices on the curve and the newly introduced vertices in the interior of the patch. Since only the connectivity is important here, the actual position of the vertices is not relevant; Hence the name *topological triangulation*. The iterative algorithm, illustrated in Figure 4, considers the sequence of curves as one single piecewise linear closed (boundary) curve. The goal at each step is to create a new piecewise linear closed curve inside the patch, such that this new curve has half the number of vertices than the previous one. Then the space between the previous curve and the new curve is triangulated. We repeat this process, and stop when the number of vertices in the inner curve reduces to three or fewer vertices. Then a triangle fan fills the remaining hole and finishes the procedure. The number of new interior closed curves will be in the order of the logarithm of the patch boundary size.

One iteration of the process is as follows: the given curve is projected onto a plane that best approximates the curve points. A new vertex is introduced for every alternate vertex on the boundary, and the new vertex is connected to its corresponding boundary vertex and its two neighbors. Two new vertices are connected to each other, if they have a common boundary neighbor. The sequence of new vertices form the new boundary curve. The above mentioned connectivity triangulates the space between the previous and the new curves. Although not relevant, the initial coordinate of the new vertex is computed as the centroid of four points: the three implied boundary vertices and the centroid P of the points on the boundary of the original input curve. The point P is fixed and does not change over iterations. This approach

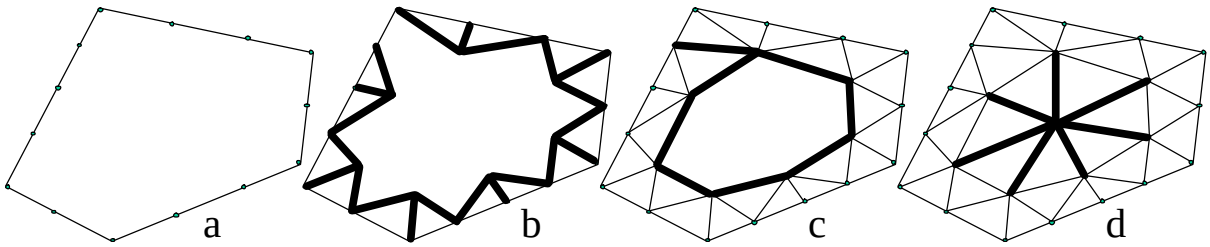

Figure 4: *Illustration of the topological triangulation procedure. Changes in each step are drawn thick:* ***(a)*** *Original patch to be triangulated. Notice the vertices in the corners and along the lines that form the patch.* ***(b)*** *A layer of triangle pairs is laid adjacent to pairs of segments. Since there is an odd number of segments, one segment remains without a triangle.* ***(c)*** *The new vertices are connected with a polyline.* ***(d)*** *Each iteration of steps* b *and* c *reduces the number of boundary segments by a half. Ween this number is small enough (our limit was 10 segments), a triangle fan closes the hole.*

to compute coordinates can be improved by taking into account the number of closed inner curves that will be computed.

Surface Fairing: Once topological structure for patch is generated, final geometric positions of the new vertices are computed to make the limit surface fair. For every internal vertex, in the order in which they were found in the topological triangulation step, we use its initially assigned coordinates and the tangent planes at its neighbors in the boundary, to find the final position of this new vertex. We take the line defined by the position of the internal vertex and its normal, and find the intersection of this line with each neighboring tangent plane. The median position of these intersection points is the new position of the vertex. Similar to the topological triangulation step, the surface fairing step is iterated to find the final positions of all the newly introduced vertices. Further iterations of this algorithm reduce the noise in the result.

Our approach to surface fitting to the network of curves provides a large number of internal vertices having degree five, except for those in the last layer of the triangulation. It is simple, has fast convergence, and runs at interactive speed for the tested models. It can handle models with any arbitrary genus and works for convex patches and those with soft concavities.

5. Implementation and Results

We have tested our approach generating a number of simple models (see Figure 5), with typically six to ten curves, producing a proportional number of patches. It took an average of about one minute for the users to sketch simple models using our system, and the automatic generation was executed at interactive rates on a 2.4 Ghz Pentium 4 processor with 512 MB RAM. Our system provides a simple interface (see Figure 6) where the user can draw lines, rotate the view and continue drawing until the desired shape is achieved. Basic utilities like picking, copying and moving curves, or indicating corner points, are part of the interface.The identification of surface patches is currently manual but will be automated in future. In order to reduce the effect of the possible outliers in the surface fairing procedure, we damp the migration of vertices, only allowing displacements proportional to the length of surrounding edges. This way the relative movement of vertices is controlled by a user-defined parameter. The surface generated gives a coarse representation of the object and can be further refined using subdivision surfaces or any other 3D modelling tool.

6. Conclusion and Future Work

In this paper we have addressed the problem of generating a free form surface from a network of curves which are in turn generated using 2D sketches. The 3D space curve generation technique uses a single view information about a 2D curve to find the curvature minimizing 3D curve. We believe that the mathematical rigor used to explain the intuitive space curve generation would find its use in perceptual studies. The 3D surface generation algorithm using the network of curves is simple and efficient that is required for interactive applications like sketch based modeling. The generated surface might not be the final surface that the user intends to keep. Hence, once the initial surface is generated, users can modify the shape of the model interactively thus making the system analogous to a virtual sculpting tool.

Planned future work in our sketch-based free-form modeling system includes automatic patch identification for surface generation, reduction of identified numerical stability issues, generation of models with boundary, experimentation with non-iterative vertex location procedures for the surface fairing, and improvement of the basic sketch interface to provide more utilities to the user, thereby making a simple, easy-to-use sketch-based free-form modeling system for generation of intuitive free-form objects.

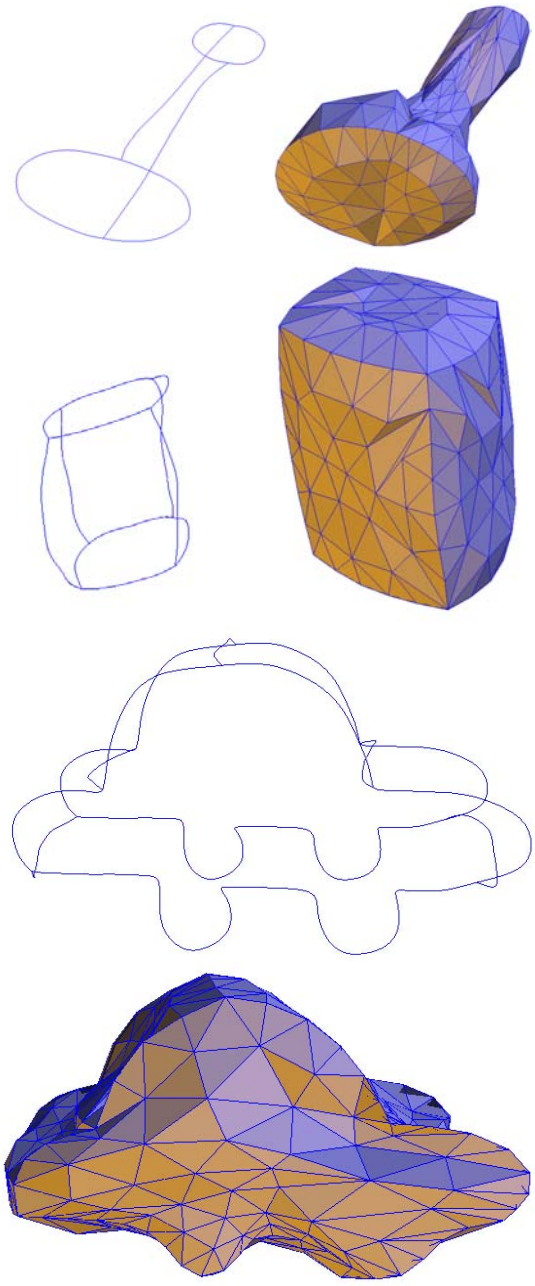

Figure 5: *Simple models generated by our system along with the input sketch*

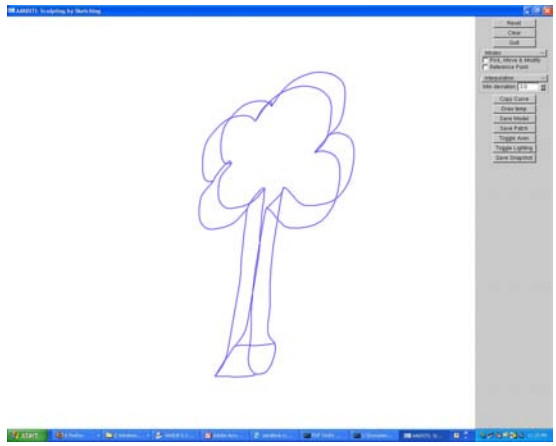

Figure 6: *User interface showing one of the sketch inputs for generated models.*

References

[Bau94] BAUDEL T.: A mark-based interaction paradigm for free-hand drawing. In *Proceedings of 7th annual ACM Symposium on User Interface Software and Technology* (1994), ACM Press, pp. 185–192.

[CMZ*99] COHEN J. M., MARKOSIAN L., ZELEZNIK R. C., HUGHES J. F., BARZEL R.: An interface for sketching 3d curves. In *Proceedings of the Symposium on Interactive 3D graphics* (1999), ACM Press, pp. 17–21.

[FB93] FOWLER B., BARTELS R.: Constraint-based curve manipulation. *IEEE Computer Graphics and Applications 13*, 5 (1993), 43–49.

[FRSS04] FLEISCH T., RECHEL F., SANTOS P., STORK A.: Constraint stroke-based oversketching for 3d curves. In *Proceedings of the Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBM-04)* (August 2004), Eurographics Association, pp. 161–166.

[GA98] GRIMM C., AYERS M.: A framework for synchronized editing of multiple curve representations. In *Proceedings of the EUROGRAPHICS* (1998), vol. 17, pp. C31–C40.

[IH03] IGARASHI T., HUGHES J. F.: Smooth meshes for sketch-based freeform modeling. In *Proceedings of the Symposium on Interactive 3D graphics* (2003), ACM Press, pp. 139–142.

[IITS04] IJIRI T., IGARASHI T., TAKAHASHI S., SHIBAYAMA E.: Sketch interface for 3d modeling of flowers. Technical Sketch at ACM SIGGRAPH, August 2004.

[IMT99] IGARASHI T., MATSUOKA S., TANAKA H.: Teddy: A sketching interface for 3d freeform design. In *Proceedings of ACM SIGGRAPH* (August 1999), ACM Press, pp. 409–416.

[KHR04] KARPENKO O., HUGHES J. F., RASKAR R.: Epipolar methods for multi-view sketching. In *Proceedings of the Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBM-04)* (August 2004), Eurographics Association, pp. 167–174.

[LS96] LIPSON H., SHPITALNI M.: Identification of faces

in a 2d line drawing projection of a wireframe object. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 18*, 10 (1996), 1000–1012.

[LS02] LIPSON H., SHPITALNI M.: Correlation-based reconstruction of a 3d object from a single freehand sketch. In *AAAI Spring Symposium on Sketch Understanding* (2002), pp. 99–104.

[MKB02] MICHALIK P., KIM D. H., BRUDERLIN B. D.: Sketch- and constraint-based design of b-spline surfaces. In *SMA '02: Proceedings of the seventh ACM symposium on Solid modeling and applications* (New York, NY, USA, 2002), ACM Press, pp. 297–304.

[MS05] MAJUMDER A., STEVENS R.: Perceptual photometric seamlessness in projection-based tiled displays. *ACM Transactions on Graphics 24*, 1 (2005), 118–139.

[ONNI03] OWADA S., NIELSEN F., NAKAZAWA K., IGARASHI T.: A sketching interface for modeling the internal structures of 3d shapes. 49–57.

[PK89] PENTLAND A., KUO J.: *The Artist at the Interface. Vision and Modeling*. Tech. Rep. 114, MIT media lab, 1989.

[SC04] SHESH A., CHEN B.: Smartpaper–an interactive and easy-to-use sketching system. In *Proceedings of Eurographics* (August 30-September 2 2004), pp. 409–416.

[SF98] SINGH K., FIUME E.: Wires: a geometric deformation technique. In *Proceedings of ACM SIGGRAPH* (1998), ACM Press, pp. 405–414.

[Sho85] SHOEMAKER K.: Animating rotation with quaternion curves. In *Proceedings of ACM SIGGRAPH* (July 1985), vol. 19, ACM Press, pp. 245–254.

[SRS91] SACHS E., ROBERTS A., STOOPS D.: 3-draw: A tool for designing 3d shapes. *IEEE Computer Graphics and Applications 11*, 6 (1991), 18–26.

[SWZ04] SCHAEFER S., WARREN J., ZORIN D.: Lofting curve networks using subdivision surfaces. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (New York, NY, USA, 2004), ACM Press, pp. 103–114.

[TDM99] TOLBA O., DORSEY J., MCMILLAN L.: Sketching with projective 2d strokes. In *Proceedings of the 12th annual ACM symposium on User interface software and technology* (1999), ACM Press, pp. 149–157.

[TZF05] TAI C.-L., ZHANG H., FONG J. C.-K.: Prototype modeling from sketched silhouettes based on convolution surfaces. 071–183.

[ZHH96] ZELEZNIK R. C., HERNDON K. P., HUGHES J. F.: Sketch: An interface for sketching 3d scenes. In *Proceedings of ACM SIGGRAPH* (August 1996), ACM Press, pp. 163–170.

Appendix: Proof of Theorem

Theorem 1: Given a 2D Bezier curve with a known depth range, the depth of the control points of curvature minimizing 3D Bezier curve is monotonically and equally spaced within the given depth range.

Proof (Sketch) For the sake of simplicity of mathematical proof, we assume that the given plane is parallel to the XY plane and the viewing direction is the Z axis. Let us also assume that the range denotes the depth value of the first and the last control point of the Bezier curve.
Let the control points of the 2D Bezier curve be denoted by $P_i = (x_i, y_i)$, $0 \leq i \leq n$, where n is the order of the curve. The Bezier curves are orthographic projection invariant. Hence the 3D curve will have the same number of control points as its 2D projection, and the control points of the 2D curve are actually the projection of the 3D curve control points. Since it is an orthographic projection, the control points of all the candidate 3D space curves will have the same x and y coordinates. Let the 3D control points be $Q_i = (x_i, y_i, z_i)$. The curve is,

$$Q(u) = \Sigma_{i=0}^{n} \frac{n!}{i!(n-i)!} Q_i u^i (1-u)^{n-i}$$

Differentiating, we get

$$Q'(u) = n\Sigma_{i=0}^{n-1} \frac{(n-1)!}{i!(n-1-i)!} [Q_{i+1} - Q_i] u^i (1-u)^{(n-1-i)}$$

and the second derivative is given by,

$$Q''(u) = (n)(n-1)\Sigma_{i=0}^{n-2} \frac{(n-2)!}{i!(n-2-i)!} [Q_{i+2} - 2Q_{i+1} + Q_i] u^i (1-u)^{(n-2-i)}$$

The normal curvature κ is given by $\frac{|Q' \times Q''|}{|Q'|^3}$. Intuitively, to minimize the maximum curvature, we need to minimize the maximum value of Q'' and maximize the minimum value of Q'.
With respect to Q', larger the differences in depth values of the control points, larger the denominator and hence the minimum value has no maxima. Hence, minimizing curvature depends only on the magnitude of Q''. This can also be seen from the fact that since curvature is an implicit characteristic of a curve, for an equivalent arc-length parameterization, the curvature is controlled by its second derivative. Based on the above argument, our goal is to minimize the magnitude of the 3D vectors $(Q_{i+2} - 2Q_{i+1} + Q_i)$ for all i. Since the x and y coordinates of Q_i are fixed by the input 2D Bezier control points, and the z coordinates of the first and last control points are fixed by the given depth range, the only free parameters are the z coordinates of the intermediate control points Q_i, $1 \leq i \leq (n-1)$. Minimizing $|z_{i+2} - 2z_{i+1} + z_i|$ would minimize the magnitude of the required vector. Since the z_0 and z_n are known, setting $z_{i+2} - 2z_{i+1} + z_i = 0$ for all i provides a set of recurrence equations. The solution for this set of equations is

$$z_i = z_0 + \frac{z_n - z_0}{n} i.$$

□