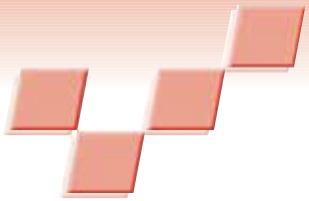
Sketch-Based 3D-Shape Creation for Industrial Styling Design



Levent Burak Kara and Kenji Shimada
Carnegie Mellon University

The authors describe a pen-based modeling system for the styling design of 3D objects. Their system is tailored toward the rapid and intuitive design of styling features such as free-form curves and surfaces. Basic wireframe and surfaces are constructed and modified using the strategy of curve creation, curve modification, surface creation, and finally surface modification.

While recent decades have seen significant progress in CAD software, current state of the art still appears insufficient when it comes to the styling design of products. This is evidenced by the fact that a significant portion of early design activities such as concept development and style generation occurs almost exclusively in 2D environments—be it the traditional pen-andF-paper environment or its digital equivalents. While part of this bias toward 2D tools in the early design stages comes from the undeniable convenience and familiarity of such media, we believe the lack of suitable software and interaction techniques to support 3D styling design has a significant role in the current bias.

In this article, we propose a pen-based modeling system for 3D-object styling design. Our system lets users create and edit 3D geometry through direct sketching on a pen-enabled tablet computer. A distinguishing feature of our system is that we tailored it toward the rapid and intuitive design of styling features—such as free-form curves and surfaces—which is often a tedious and complicated task using conventional software.

A key commonality among the product types we consider is that their aesthetic appeal is a central consideration. Additionally, given that the final aesthetic form usually evolves in time rather than simply occurring, it's important that users of our system can accurately reproduce their ideas, while having the ability to quickly explore alternatives. From a geometric standpoint, this often translates into users creating and frequently modifying free-form curves and surfaces. Our system's main utility lies precisely at this point in that it supports the direct creation and editing of such entities through an intuitive, pen-based interface. We intend that a wide variety of designers will use our system—ranging from those who prefer the traditional pen-and-paper interface to those who are accustomed to, and frequently use, existing CAD tools.

In a typical scenario using our system, the user begins by constructing the base wireframe model of the design object. For this, the user sketches the initial feature curves on a rough and simplified 3D template model. This template acts as a platform that helps anchor users' initial strokes in 3D space. Once the initial curves comprising the wireframe are constructed, the base 3D template is removed, leaving the user with a set of 3D curves. Next, through direct sketching, the user modifies the initially created curves to give them the precise desired shape. After the generating the wireframe, the user constructs interpolating surfaces that cover the wireframe. Finally, using two physically based deformation tools, the user modifies the newly created surfaces to the desired shapes. Once the basic wireframe and surfaces are created, the user can add further details using the same strategy of curve creation, curve modification, surface creation, and finally surface modification.

Our approach

In our previous work,[1] we advocate the use of deformable wireframe models as a base to facilitate styling design. In that work, users' input strokes help manipulate existing edges of a wireframe model. While the work presented here shares several similarities with our previous work, this article presents extensions to our previous work; here users can create an initial topology in the form of a network of curves on a simplistic surface model of the design object. With our proposed system, the network of curves the user designs can be arbitrarily complex with no restrictions on the number of faces. Moreover, our subsequent curve modification techniques allow drastic modifications to the initially created wireframe, thus making the selection of the initial template surface model a relatively noncritical issue.

This work also introduces several new techniques for 3D curve creation and modification. For curve creation, in particular, we present a new and simple technique for creating free-form 3D curves with varying depth coordinates, so long as the two ends of the curve can be anchored in 3D using existing primitives in the scene.

 Published by the IEEE Computer Society

Related Work

Computer modeling of 3D geometry using alternative input devices and interaction techniques has received considerable attention in recent years. While a number of techniques involving 3D input devices, haptic devices, and VR systems have been proposed, our focus is on those in which the primary interaction is purely sketch based—that is, a stylus on a 2D medium.

The key difficulty of interpreting 3D information from 2D input has forced researchers to devise a variety of different techniques. Some works focus on generating 3D geometry by inflating 2D silhouettes.[1,2] Various modeling operations such as extrusion, sweep, cut, and bend help users modify the initial geometry. The emphasis in such systems is to quickly generate a reasonable 3D shape rather than a precise modeling of the object. In gesture-based techniques, designers' strokes are used primarily for editing an existing primitive object into the desired shape.[3] While such approaches allow a fast construction of the geometry, they are most useful for constructing rectilinear models with minimal curved edges and surfaces. Optimization-based algorithms[4] and techniques based on line-labeling[5] produce the most plausible 3D shape from a 2D sketch of its wireframe. While researchers have recently begun to extend these techniques to curved edges, their use in aesthetic shape design is currently still limited, making the techniques more suitable for engineering-type geometries. In an interesting alternative method, Cohen et al. exploit shadows to facilitate 3D interpretation.[6] In this system, a space curve drawn in a 2D interface is complemented with a sketch of the same curve's shadow sketched on a plane. However, this approach relies on the user's ability to accurately visualize and depict a curve's shadow.

A number of template-based methods have also been proposed.[7,8] In these systems, the desired 3D form is obtained by deforming an underlying 3D template. For instance, Mitani et al. use a six-faced topological template for interpretation.[7] The nature of the template, however, limits the method's scope to objects topologically equivalent to a cube. Das et al. describe an approach for free-form surface creation from a network of curves.[9] Their solution to 3D interpretation from 2D input seeks to produce 3D curves with minimum curvature. This choice is justified on the grounds that the resulting 3D curve will be least surprising when viewed from a different viewpoint. Our formulation of the best 3D interpretation during curve modification is based on a similar rationale, except we minimize the deviation from an existing 3D curve as opposed to curvature. A curve modification technique most similar to ours[10] describes a multiview sketching system that lets users first create a curve and then modify it from different views. This system proposes epipolar constraints as a way to facilitate a reliable registration of the 3D curve with the users' input strokes, when the viewing point is altered. This method is similar to our single-view curve modification method except they project the sketched stroke onto the epipolar lines of the existing 3D curve, while we project the 3D curve onto the epipolar lines obtained through the sketched stroke. Additionally, we introduce the concept of two-view curve modification that seeks to generate curves that lie at the intersection of two epipolar surfaces obtained from two different views.

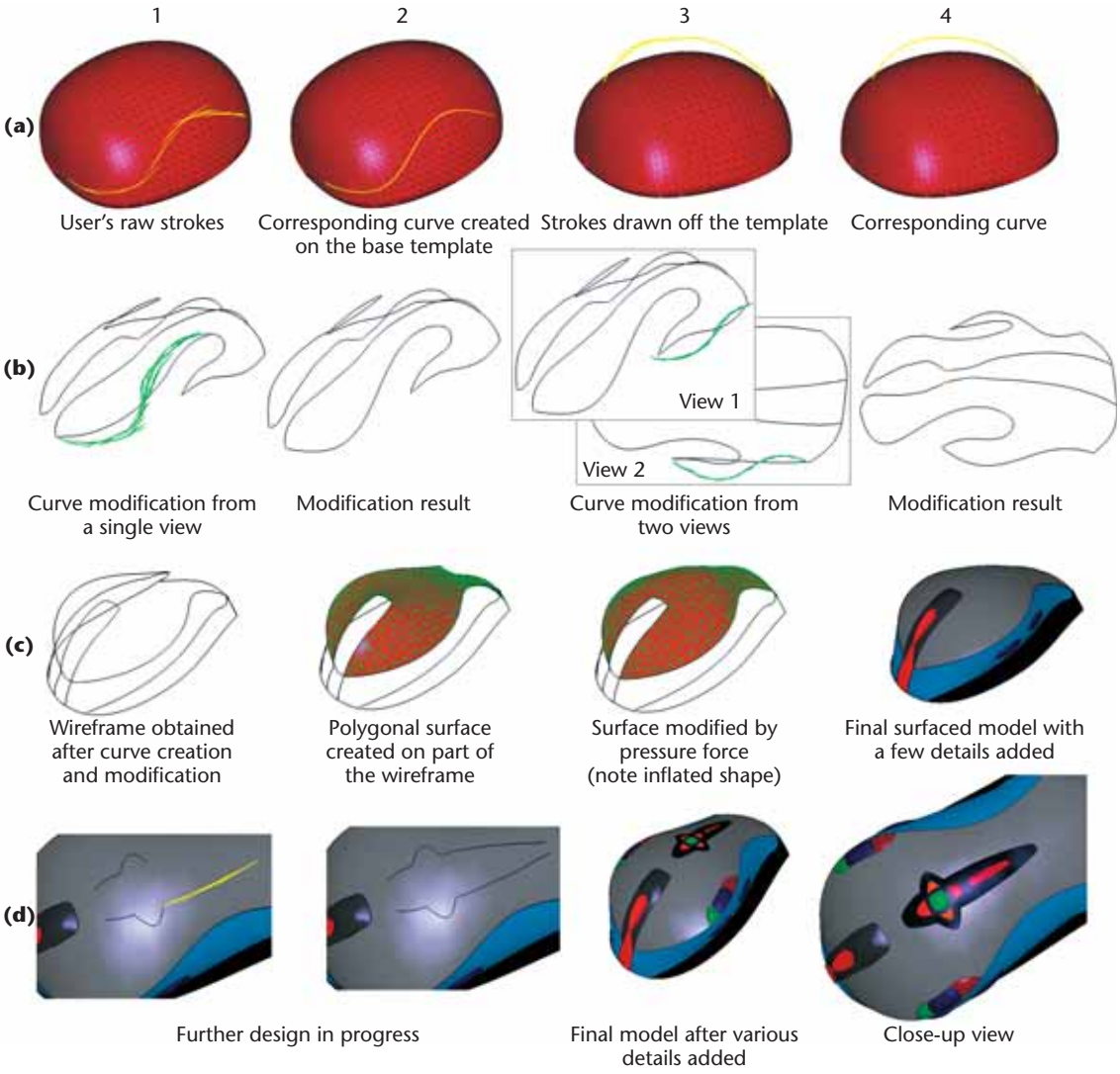
A large body of work has been devoted to designing and fairing free-form surfaces.[11-13] For surface fairing, most approaches minimize the integral of squared principal curvatures (or variations of it), and present descretized versions of the resulting optimization function that is applicable to polygonal surfaces. From a geometric standpoint, our surface modification algorithm based on a network of springs is analogous to these methods, except it's formulated based on a purely physical behavior. Our current implementation, however, does not consider some of the issues addressed in others, such as the ability to specify constrained curves embedded in the surface, or the ability to dynamically update the mesh connectivity to obtain regular mesh distributions.

References

1. T. Igarashi, S. Matsuoka, and H. Tanaka, "Teddy: A Sketching Interface for 3D Freeform Design," *Proc. Siggraph*, ACM Press, 999, pp. 409-416.
2. O. Karpenko, J.F. Hughes, and R. Raskar, "Free-Form Sketching with Variational Implicit Surfaces, *Eurographics, Computer Graphics Forum*, vol. 21, no. 3, 2002, pp. 585-594.
3. R.C. Zeleznik, H.P. Herndon, and J.F. Hughes, "Sketch: An Interface for Sketching 3D Scenes," *Proc. Siggraph*, ACM Press, 1996, pp. 163-170.
4. M. Masry, D.J. Kang, and H. Lipson, "A Freehand Sketching Interface for Progressive Construction of 3D Objects," *Computers and Graphics*, vol. 29, no. 4, 2005, pp. 563-575.
5. P.A.C. Varley et al., "A Two-Stage Approach for Interpreting Line Drawings of Curved Objects," *Proc. Eurographics Workshop Sketch-Based Interfaces and Modeling*, Eurographics, 2004.
6. J.M. Cohen et al., "An Interface for Sketching 3D Curves," *Proc. Symp. Interactive 3D Graphics*, ACM Press, 1999, pp. 17-21.
7. J. Mitani, H. Suzuki, and F. Kimura, "3D Sketch: Sketch-Based Model Reconstruction and Rendering," *Proc. Workshop Geometric Modeling*, IFIP,2000, pp. 85-98.
8. L.B. Kara, C. D'Eramo, and K. Shimada, "Pen-Based Styling Design of 3D Geometry Using Concept Sketches and Template Models," *Proc. ACM Solid and Physical Modeling Conference*, ACM Press, 2006, pp. 149-160.
9. K. Das, P. Diaz-Gutierrez, and M. Gopi, "Sketching Free-Form Surfaces Using Network of Curves,"

For curve modification, we introduce the notions of single- and two-view modifications as two alternatives to modifying 3D curves. Both our curve creation and modification methods enjoy the use of active contours,[2] which has proven highly suitable for free-form curve design. A key feature of our techniques is that they allow both curve creation and modification from totally arbitrary viewing points, thus providing the user with broad flexibility during curve design.

Product design in our target domain is usually different than the design of products that are inherently mechanical or functional in nature. Unlike the target



1 Modeling operations supported by our system. (a) Curve creation on and off the base template. (b) Single, and two-view curve modification illustrated on arbitrary curves. In the single-view modification mode the user modifies the curve using strokes drawn from one particular view. In the two-view modification mode, the system combines the stroke information gathered from the two views to produce the final modification. (c) Illustration of surface creation and modification on a user-designed wireframe. (d) Further design is performed in a similar way. Users draw, modify, and finally surface curve features.

products considered here, the majority of mechanical components exhibit more structured features such as sharp and straight edges, flat surfaces, simple geometry artifacts such as fillets and holes, and repetitive patterns. The design of such components is greatly facilitated by the use of conventional solid modeling techniques such as Boolean operations on primitive geometry, extrusion, cut, loft, revolve, and sweep, all of which existing CAD software readily support. However, the same set of tools and techniques often becomes less effective for styling design purposes. The work presented in this article is one attempt toward alleviating this shortcoming.

User interface and design overview

Our system's main input device is a pressure sensitive LCD digitizing tablet with a cordless stylus. Users' strokes are collected as time sequenced (x, y) coordinates sampled along the stylus' trajectory. Similar to the left and right buttons on a mouse, the stylus contains two buttons along its barrel, which we have customized for the camera rotation and translation operations. The user interface consists of a main drawing region, and a side toolbar for accessing commonly used commands and settings.

We break the design process into four main steps. The first is the initial layout step in which the user constructs the design object's wireframe model through direct sketching on a base template. As shown in Figure 1a, column 1, the user can lay down a curve on the template using multiple strokes, drawn in any direction and order, thus accommodating casual drawing styles such as overstroking. After drawing the strokes, the user invokes a command that processes and beautifies the collection of raw strokes into a smooth curve lying on the base template.

The base template is typically a simplified solid model of the design object in question. For instance, as shown in Figure 1a, column 2, a simple half-egg shape serves as a suitable template for the computer mouse's design. Taking advantage of the graphics engine's depth buffer, particularly its fast ray intersection capability, this template provides a platform on which users' pen strokes can be conveniently captured in 3D space. Since the curves obtained this way lie directly on the template, the initial wireframe constructed at the end of this step will usually possess a roughly correct geometry and relative proportions. This, in turn, will greatly lessen the work involved in the subsequent step of wireframe modification. Besides these practical advantages, the use of a template also helps avoid the well-known challenge of one-to-many mapping in 3D interpretation from 2D input.

In addition to creating curves that lie entirely on the template, users can also create curves that reside largely off the template, provided that their two ends lie on the template, see Figure 1a, columns 3 and 4. In this case, the system determines the best 3D configuration of the curve based on an energy minimization algorithm. As explained later, this algorithm forces the 3D curve to lie right under the input strokes (as expected), while minimizing its geometric complexity. In both curve creation methods discussed previously, the user can specify a symmetry plane, which is typically one of the three principal Cartesian planes. When the user activates a symmetry plane, the system replicates symmetrically the work on one side of the symmetry plane onto the other side, thus expediting the design process.

The curves obtained in the first step make up the design object's initial wireframe. In the second stage, the user modifies the wireframe's curves to the desired shapes using a sketch-based modification technique. During this step, the user can safely remove the base template used in the first step, as this template is no longer required. To modify a curve, the user sketches the curve's new shape directly on the computer screen. Based on the spatial proximity in the image plane—that is, the display screen on which the user draws and views the model—our program first identifies the curve that the user intends to edit. Next, using an energy minimization algorithm similar to the one mentioned previously, the program modifies the curve in 3D (see Figure1b, columns 1 and 2). The resulting curve, which we call the *minimum surprise curve*, closely approximates the new shape dictated by the input strokes, while minimizing the deviation from the original 3D curve. As before, symmetry can be preserved across a user-specified plane if necessary.

Besides providing the ability to modify a curve from a single viewpoint, our system also lets a user modify a curve from two different and arbitrary viewpoints (see Figure 1b, columns 3 and 4). In this case, the user sketches separately a curve's new shape in the two views. As explained later, the program combines the information gathered from these two views to produce the curve's new 3D shape. This feature is advantageous especially when the desired curve possesses 3D geometry that is hard to depict from a single viewpoint.

At the end of the first two steps, the system provides the user with a wireframe model where each of the constituent curves has been accurately designed. In the third step, the user constructs surfaces on the desired loops of the wireframe to obtain a solid model, see Figure 1c, columns 1 and 2. The initial surfaces laid on the wireframe have a common property of being minimum-area surfaces, which is analogous to the way a soap film would stretch across a wire loop. In the fourth step of the design, each surface patch is modified and refined using two physically based deformation tools. The first tool, inspired by the deformation of a thin membrane under a pressure force, allows the user to inflate or flatten a surface by a controllable amount (see Figure 1c, column 3). The user can control the amount of pressure applied to a surface through a simple slider bar located in the user interface. The intuitive nature of this deformation tool lets the user quickly and straightforwardly explore different surface shapes. The user can also modify surfaces using a method inspired by mechanical springs. This method works to minimize the variation of mean curvature, producing fair and aesthetically pleasing surfaces. Using these surface creation and modification techniques, the user obtains a final surface model (see Figure 1c, column 4).

While the four design steps nominally take place sequentially, at any time the user can go back to an earlier step to add new curves or modify existing ones. This flexibility allows users to later add details to the model (see Figure 1d, columns 1 and 2). In this case, the surfaces created by the user can be used as suitable platforms to create new curves, thus eliminating the need for the base template used in the first step. Newly added curves can then be modified and later surfaced using the same techniques (see Figure 1d, columns 3 and 4).

In this article, we focus primarily on the wireframe model's construction and modification. A more thorough discussion of our surfacing techniques can be found in our previous work.[1]

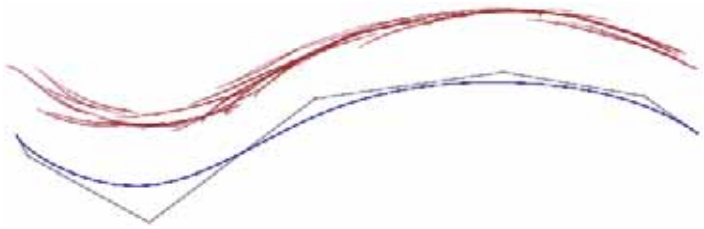
Constructing the 3D wireframe

As mentioned previously, in the first step of the design, the user constructs the wireframe by sketching its constituent 3D curves. Unlike many of the previous approaches, our system lets users create curves with an arbitrary number of strokes, drawn in any direction and order. With this in hand, our system offers two methods for creating curves based on whether the curves are instantiated entirely on the base template, or otherwise.

Creating curves on the template

In our method, users create curves directly on the base template. The process consists of two main steps. The first is a beautification step in which our program computes a smooth B-spline that closely approximates the input strokes in the image plane. In the second step, the 2D curve obtained in the image plane is projected onto the template resulting in a 3D curve.

Given the input strokes in the image plane, our program first fit a B-spline to the collection of strokes using a minimum least-squares criterion described elsewhere.[3]



2 B-spline fitting to raw strokes: input strokes (top) and the resulting B-spline and its control polygon (bottom).

Figure 2 shows an example. By default, we use cubic B-splines with seven control points. While we determined empirically these choices to best suit our purposes, we can adjust them to obtain the desired balance between computation speed, curve smoothness, and accuracy approximation. Nevertheless, details of the curve fitting process and the resulting auxiliary features, such as the curve's control polygon, are hidden from the user. The user is only presented with the resulting curve.

Normally, the data points used for curve fitting would be those sampled along the stylus' trajectory. However, fluctuations in the drawing speed often cause consecutive data points to occur either too close to, or too far away from one another. This phenomenon—as evidenced by dense point clouds near the stroke ends (where drawing speed tends to be low) and large gaps in the middle of the stroke (where speed is high)—often adversely affects curve fitting. Hence, before curve fitting is applied, we resample each stroke to obtain data points equally spaced along the stroke's trajectory.

The main challenge in curve fitting, however, arises from the fact that a curve can be constructed using multiple strokes, drawn in arbitrary directions and orders. This arbitrariness often causes spatially adjacent data points to have markedly different indices in the vector storing the data points. An accurate organization of the data points based on spatial proximity, however, is a strict requirement of the curve-fitting algorithm. Hence, prior to curve fitting, input points must be reorganized to convert the cloud of unordered points into an organized set of points. This reorganization would only affect the points' indices in the storage vector, not their geometric locations.

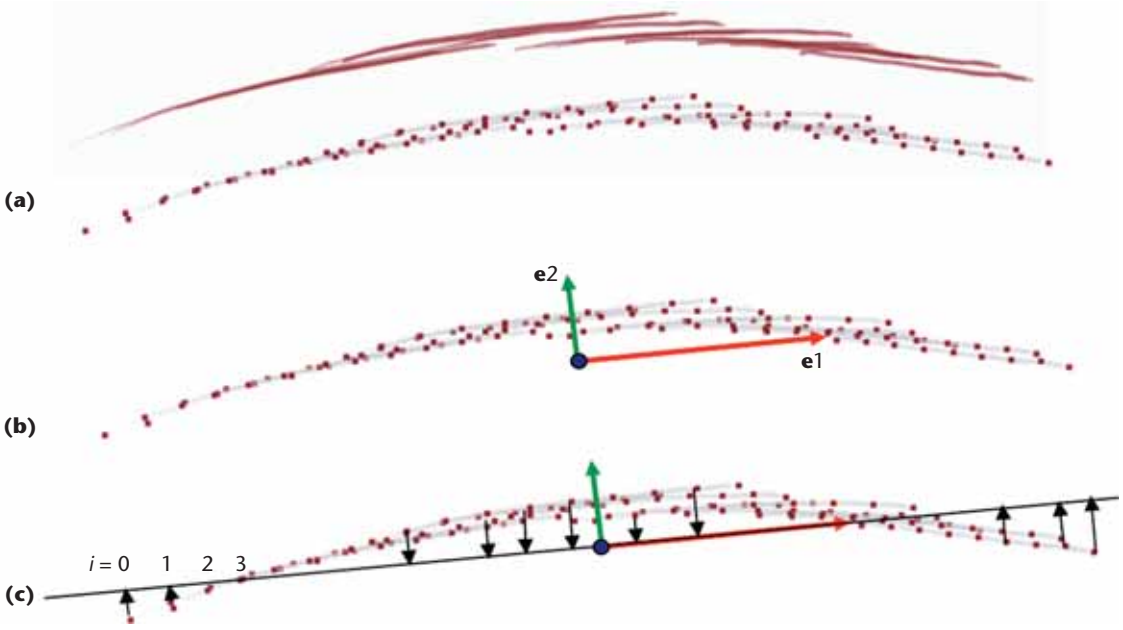
To obtain this goal, we use a principal component analysis, as Figure 3 shows. The main idea is that, by identifying the direction of maximum spread of the data points, we can obtain a straight-line approximation to the points. Next, by projecting the original points onto this line and sorting the projected points, we can obtain a suitable ordering of the original points.

Given a set of 2D points, the two principal directions can be determined as the eigenvectors of the 2×2-covariance matrix Σ given as

$$\Sigma = \frac{1}{n}\sum_{k=1}^{n}\left(\mathbf{x_k} - \mu\right)\left(\mathbf{x_k} - \mu\right)^T$$

Here, $\mathbf{x_k}$ represents the column vector containing the (x, y) coordinates of the k-th data point, and μ is the centroid of the data points.

The principal direction we seek is the one that corresponds to maximum spread, and is the eigenvector associated with the larger eigenvalue. After identifying the principal direction, we form a straight line passing through the centroid of the data points and project each of the original points onto the principal line. Next, we sort the projected points according to their positions on the principal line. The resulting ordering is then used as the ordering of the original points. One advantageous



3 Point ordering using principal component analysis. (a) Input strokes and extracted data points. (b) The program computes the two principal component directions as the eigenvectors of the data points' covariance matrix. The two direction vectors are positioned at the data points' centroid. (c) Data points are projected onto the first principal direction e1, and sorted.

by-product of this method is that it reveals the curve's extremal points (that is, its two ends), which would otherwise be difficult to identify.

In practice, we have found this approach to work well especially because the curves created with our system often stretch along a unique direction. Hence, the projections' ordering along the principal direction often matches well with the expected ordering of the original data points. However, this method falls short when users' curves exhibit hooks at the ends, or contain nearly or fully closed loops. This is because these artifacts will cause folding or overlapping of the projected points along the principal direction, thus preventing a reliable sorting. To circumvent such peculiarities, we ask users to construct such curves in pieces consisting of simpler curves; our program can later stitch together separate curves using a trim function.

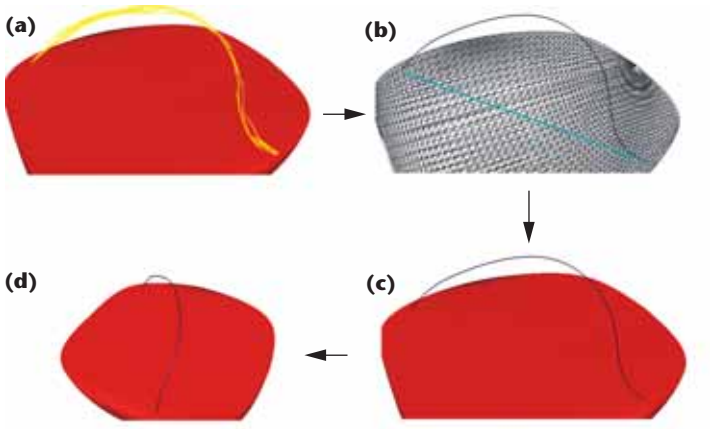
Once the raw strokes are beautified into a B-spline, the resulting curve is projected onto the base template. This is trivially accomplished using the graphics engine's depth buffer. At the end, a 3D curve is obtained that lies on the template, whose projection to the image plane matches with the user's strokes (see Figure 1a, column 2).

Creating curves in space

Users can also create curves that lie mostly off the template, provided that their two end points lie on the template (see Figure 4). As before, a user can construct a curve with multiple strokes drawn in arbitrary directions and order. In this case, the intersection of the curve ends and the template helps anchor the curve ends in 3D. The main challenge, however, is to identify the best 3D configuration of the curve's body, which likely lies off the template. This is because, unlike in the previous case, there is no reference object in 3D that can help capture the curve parts that lie off the template. Hence, there are infinitely many curves whose projections would match the user's strokes in the given view. As a result, the curve's best 3D configuration must be chosen based on certain constraints.

Trivially, we would expect the entirety of the 3D curve to lie right under the input strokes. Additionally, out of infinitely many possible such curves, we elect to favor curves with reasonably simple geometry. Under these premises, our solution is based on an energy minimization algorithm that has active contours at its heart.[2]

Imagine the user wants to modify an existing 3D curve by sketching its new shape on the display. Our system provides a way to accomplish this. First, the existing 3D curve is projected to the image plane producing a 2D curve. This projection is precisely the curve that the user normally sees on the display screen. Next, using an energy minimization technique, the projected curve is forced to change its shape until it closely conforms to the strokes sketched by the user. The result is that the projected 2D curve now takes on the new shape specified by the user's strokes. Finally, the new 2D curve is projected back into 3D, resulting in the new 3D curve. As noted previously, there are infinitely many such inverse projections. The key here is that we perform projection back into 3D in a way that produces a curve that deviates minimally from the original 3D curve. We have found this choice to produce convincing results in the majority of cases.



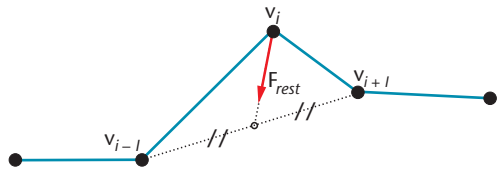
4 Curve construction off the template. (a) Input strokes. (b) The initial 3D curve (blue) instantiated as a straight line between the extremal points, and the final 3D curve (black) obtained after modification of the initial curve using snake-based energy minimization. (c-d) Two views of the resulting curve.

While this idea is used primarily to modify existing curves, we also use it to create curves in 3D, see Figure 4a. In this case, we use the two points obtained by intersecting the curve ends with the base template to instantiate a straight line between the two points as shown in Figure 4b. This process provides a suitable initial 3D curve, thereby converting the task of curve creation into a task of curve modification. The user's strokes describing the desired curve shape are then simply treated as modifiers that change this initial curve. Figures 4c and 4d show two views of the resulting curve. Unlike other approaches that assume a work plane perpendicular to the viewing direction—that is, those that resort to symmetry constraints, or those that require curves to be drawn from multiple views—our approach allows users to create curves from a single view, directly in 3D, with varying depth coordinates. Moreover, the resulting curves conform precisely to the shape dictated by the user, while exhibiting convincing forms when viewed from different viewpoints. Although the latter is a subjective assessment, it is based on the premise that visually simpler shapes are preferable over those with complex artifacts, for example, bends along the viewing direction. Our system achieves simplicity during curve creation by minimizing the deviation from a straight line.

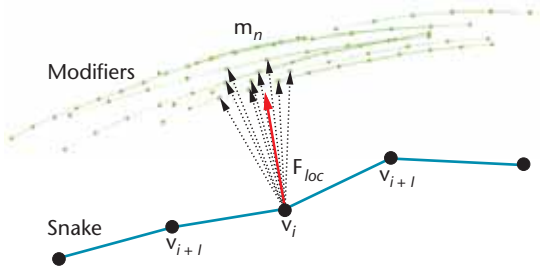
Modifying the wireframe

After creating the initial wireframe, the user begins to modify its constituent curves to give them the precise desired shape. During this step, the base template can be removed, leaving the user with a set of 3D curves.

Two methods can be used to modify a curve. The first is by sketching the new curve in a single view. In the second method, the user sketches the curve from two arbitrary views. This second method is advantageous when it's difficult to depict the curve's desired 3D shape using a single view. In both methods, we use an energy minimization algorithm to obtain the best modification of the curve in question.



5 Internal energy due to stretching and bending is minimized approximately by moving each snake node to the barycenter of its neighbors similar to Laplacian smoothing.



6 Location force on a node.

To make matters simple, we designed our approach so that the wireframe's curves are modified one at a time, with freedom to return to an earlier curve. At any point, the curve that the user intends to modify is determined automatically, thus allowing the user to modify edges in an arbitrary order. After each set of strokes, the user presses a button that processes accumulated strokes, and modifies the appropriate curve. In this article, we call users' input strokes *modifiers*, and the curve modified by those modifiers the *target curve*.

Single-view modification

Wireframe modification takes place in three steps. In the first step, curves of the wireframe are projected to the image plane resulting in a set of 2D curves. The curve that the user intends to modify is computed automatically by identifying the curve whose projection in the image plane lies closest to the modifier strokes. Sampling a set of points from the curve and the modifiers, and calculating the aggregate minimum distance between the two point sets compute the proximity between a projected curve and the modifiers. In the second step, the target curve is deformed in the image plane until it matches well with the modifiers. In the third step, the modified target curve is projected back into 3D space.

Curve modification in the image plane

We deform a projected target curve in the image plane using an energy-minimizing algorithm based on active contour models.[2] Active contours (also known as snakes) have long been used in image processing applications such as segmentation, tracking, and registration. The principal idea is that a snake moves and conforms to certain features in an image, such as intensity gradient, while minimizing its internal energy due to bending and stretching. This approach allows us to extract or track an object in the form of a continuous spline.

We adopt the previously discussed idea for curve manipulation. Here, we model the 2D target curve as a snake, whose nodes are sampled directly from the target curve. The snake's nodes are connected to one another with line segments making the snake geometrically equivalent to a polyline. The set of modifier strokes, on the other hand, is modeled as an unordered set of points (point cloud) extracted from the input strokes. As before, this allows for an arbitrary number of modifiers, drawn in arbitrary directions and order. With this formulation, the snake deforms and conforms to the modifiers, but locally resists excessive bending and stretching to maintain smoothness. Mathematically, this can be expressed as an energy functional to be minimized:

$$E_{snake} = \sum_i E_{int}\left(\mathbf{v}_i\right) + E_{ext}\left(\mathbf{v}_i\right)$$

where $\mathbf{v}_i = (x_i, y_i)$ is the i-th node coordinate of the snake. E_{int} is the internal energy arising from the stretching and bending of the snake. Our solution of minimizing this term involves applying a restitutive force $\mathbf{F}_{rest}$ that simply moves each snake node toward the barycenter of its neighboring two nodes (see Figure 5).

External energy E_{ext} describes the snake's potential energy due to external attractors, which arise in the presence of modifiers. The modifiers' influence on the snake consists of two components: location forces and pressure forces. The first component moves the snake toward the data points sampled from the modifiers. For each snake node $\mathbf{v}_i$, a force $\mathbf{F}_{loc}(\mathbf{v}_i)$ is computed corresponding to the influence of the location forces on $\mathbf{v}_i$:

$$\mathbf{F}_{loc}\left(\mathbf{v}_i\right) = \sum_{n \in k\_neigh} \frac{\mathbf{m}_n - \mathbf{v}_i}{\left\|\mathbf{m}_n - \mathbf{v}_i\right\|} \cdot w\left(n\right)$$

where $\mathbf{m}_n$ is one of the k closest neighbors of $\mathbf{v}_i$ in the modifiers (see Figure 6). $w(n)$ is a weighting factor inversely proportional to the distance between $\mathbf{m}_n$ and $\mathbf{v}_i$. In other words, at any instant, a snake node $\mathbf{v}_i$ is pulled by k nearest modifier points. The force from each modifier point $\mathbf{m}_n$ is inversely proportional to its distance to $\mathbf{v}_i$, and points along the vector $\mathbf{m}_n - \mathbf{v}_i$.

The second component of E_{ext} is related to pressure with which strokes are drawn. The force created due to this energy pulls the snake toward sections of high pressure. The rationale behind considering the pressure effect is based on the observation that users typically press the pen harder to emphasize critical sections while sketching. The pressure term exploits this phenomenon by forcing the snake to favor sections drawn more emphatically. For each snake node $\mathbf{v}_i$, a force $\mathbf{F}_{pres}(\mathbf{v}_i)$ is computed as

$$\mathbf{F}_{pres}\left(\mathbf{v}_i\right) = \sum_{n \in k\_neigh} \frac{\mathbf{m}_n - \mathbf{v}_i}{\left\|\mathbf{m}_n - \mathbf{v}_i\right\|} \cdot p\left(n\right)$$

where $p(n)$ is a weight factor proportional to the pen pressure recorded at point $\mathbf{m}_n$.

During modification, the snake moves under the influence of the two external forces while minimizing its internal energy through the restitutive force. In each iteration, the new position of $\mathbf{v}_i$ is determined by the vector sum of $\mathbf{F}_{rest}$, $\mathbf{F}_{loc}$, and $\mathbf{F}_{pres}$, whose relative weights can be adjusted to emphasize different components. For

example, increasing the weight of $\mathbf{F}_{rest}$ will result in smoother curves with less bends. On the other hand, emphasizing $\mathbf{F}_{pres}$ will increase the sensitivity to pressure differences with the resulting curve favoring high-pressure regions. Default weights are currently 30 percent for $\mathbf{F}_{rest}$, 40 percent for $\mathbf{F}_{loc}$, and 30 percent for $\mathbf{F}_{pres}$. We have determined these weights empirically so that we can obtain subjectively the best outcomes in our test cases.

Unprojection to 3D

In this step, the newly designed 2D curve is projected back into 3D space. As mentioned previously, there is no unique solution because there are infinitely many 3D curves whose projections match the 2D curve. We therefore choose the best 3D configuration based on the following constraints:

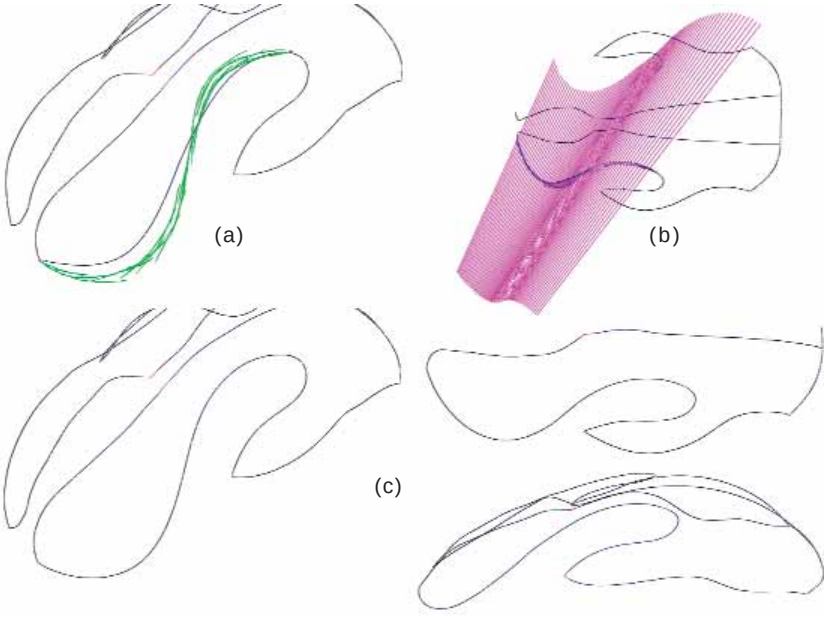
- The 3D curve should appear right under the modifier strokes.
- If the modifier strokes appear precisely over the original target curve, that is, the strokes do not alter the curve's 2D projection, the target curve should preserve its original 3D shape.
- If the curve is to change shape, it must maintain a reasonable 3D form; that is, a solution that the designer would accept in many cases, while anticipating it in the worst case.

Based on these premises, we choose the optimal configuration as the one that minimizes the spatial deviation from the original target curve. That is, among the 3D curves whose projections match the newly designed 2D curve, we choose the one that lies nearest to the original target curve. This can be formulated as follows: Let C be a curve in $\Re^3$ constrained on a surface S. (S is the surface subtended by the rays emanating from the user's viewpoint and passing through the newly designed 2D curve. This surface extends into 3D space and is not visible from the original viewpoint.) Let C^{orig} be the original target curve in $\Re^3$ that the user is modifying. The new 3D configuration C^* of the modified curve is computed as

$$C^* = \underset{C}{argmin} \sum_i \left\| C_i - C_i^{orig} \right\|$$

where C_i denotes the i–th vertex of C. With this criterion, C^* is found by computing the minimum-distance projection points of C_i^{orig} onto S (see Figure 7.)

The rationale behind this choice is that, by remaining proximate to the original curve, we can think of the new curve as least surprising when viewed from a different viewpoint. One advantage of this is that curves can be modified incrementally, with predictable outcomes in each step. That is, as the curve desirably conforms to the user's strokes in the current view, it still preserves most of its shape established in earlier steps as it deviates minimally from its previous configuration. This allows geometrically complex curves to be obtained by only a few successive modifications from different viewpoints.



7 Curve modification from a single view. (a) User's strokes. (b) Surface S created by the rays emanating from the user's eyes and passing through the strokes, and the minimum-distance lines from the original curve. (c) Resulting modification from three different views.

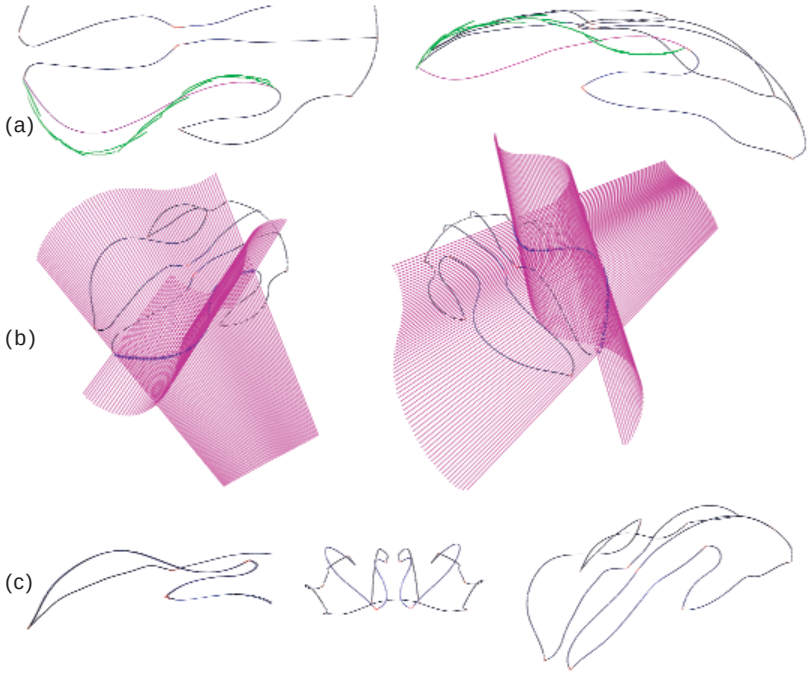
Two-view modification

As mentioned, we can use successive applications of the previous single-view modification to modify a curve into any desired shape. The two-view modification method provides yet a faster alternative to the single-view modification, especially when the curve in question cannot be easily depicted from a single view. In this setting, the user sketches the new shape of the curve from two arbitrary (but distinct) views. Similar to the single-view method, the modifiers drawn in each view define a surface that emanates from the current viewpoint, and extends into 3D passing through the modifiers. A shown in Figure 8 (on the next page), the key here is that the intersection of these two surfaces defines a unique curve in 3D. This curve is the theoretical common solution of the curves drawn in the two views.

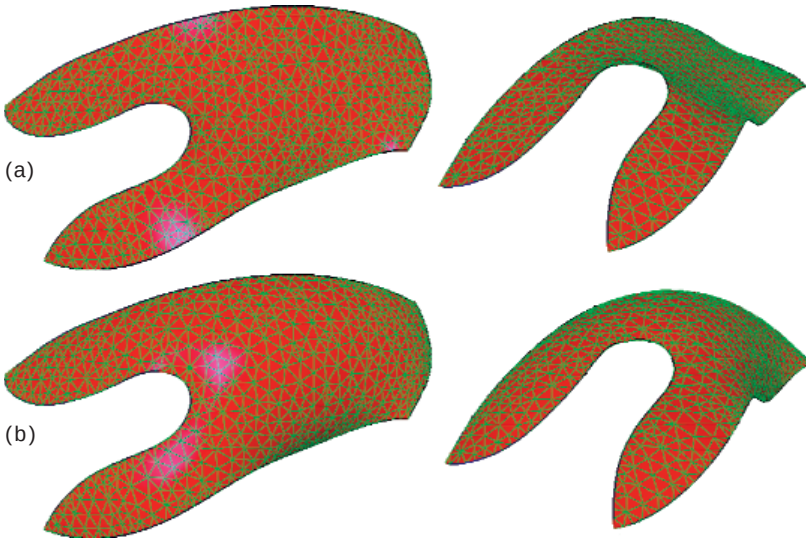
We compute the intersection curve between the two surfaces in the form of a polyline—that is, a set of points connected by line segments. The points comprising the polyline are the intersections between the longitudinal triangles obtained by triangulating the rays emanating from the eye.

With this newly obtained 3D curve in hand, we once again modify the original curve using a snake-based energy minimization method. The difference, however, is that the snake algorithm is now applied directly in 3D, rather than in 2D. In this case, a 3D snake is instantiated from the original curve. The snake then conforms to the curve defined by the intersection of two surfaces.

Although the intersection curve can be readily taken as the new modified curve, we still choose to create the modified shape through energy minimization. This is



8 Curve modification from two views. (a) The user draws the desired curve shape from two distinct views. (b) The user's input defines two surfaces that extend into 3D. The intersection curve of these two surfaces dictates the curve's new shape. (c) Resulting modification from different views.



9 Surface creation and modification via pressure force illustrated on an arbitrary loop. (a) Two views of the initial polygonal surface. (b) The same surface modified by a pressure force.

because the weights in our energy minimization algorithm can be conveniently adjusted to balance curve smoothness versus compliance. This flexibility helps suppress undesirable artifacts such as sharp bends and kinks that occasionally occur when computing the surface intersection curve.

We have found that the two-view modification method is most effective when the two viewing directions are nearly orthogonal to each other. This is because the two views in this case provide mutually exclusive information about the desired curve shape, thus maximizing the technique's utility. When the two viewing directions are the same or nearly parallel, the information obtained from one of the views becomes redundant. A more critical issue, however, is that similar viewing directions might lead to difficult to resolve situations. This happens especially when the user sketches markedly different curves from two similar viewpoints. In such cases, the two surfaces created through the modifiers might not intersect at all, or produce unexpected intersection curves. It's thus conducive to avoid similar viewing directions when using this method.

During wireframe creation and modification, the user operates on the constituent curves one at a time, without regard to their connectivity. Hence, the curves in the resulting wireframe will likely be disconnected. To prepare the wireframe for surfacing, the user might invoke a trim command that merges curve ends that lie sufficiently close to one another. This command applies an appropriate set of translations, rotations, and scalings to the entirety of a disconnected curve so that its ends meet with other curves. By transforming a curve as a whole, rather than simply extending its ends, the shape established by the user is better preserved while eliminating the kinks that could otherwise occur at the curve ends. At the end, we get a well-connected wireframe that we can subsequently surface.

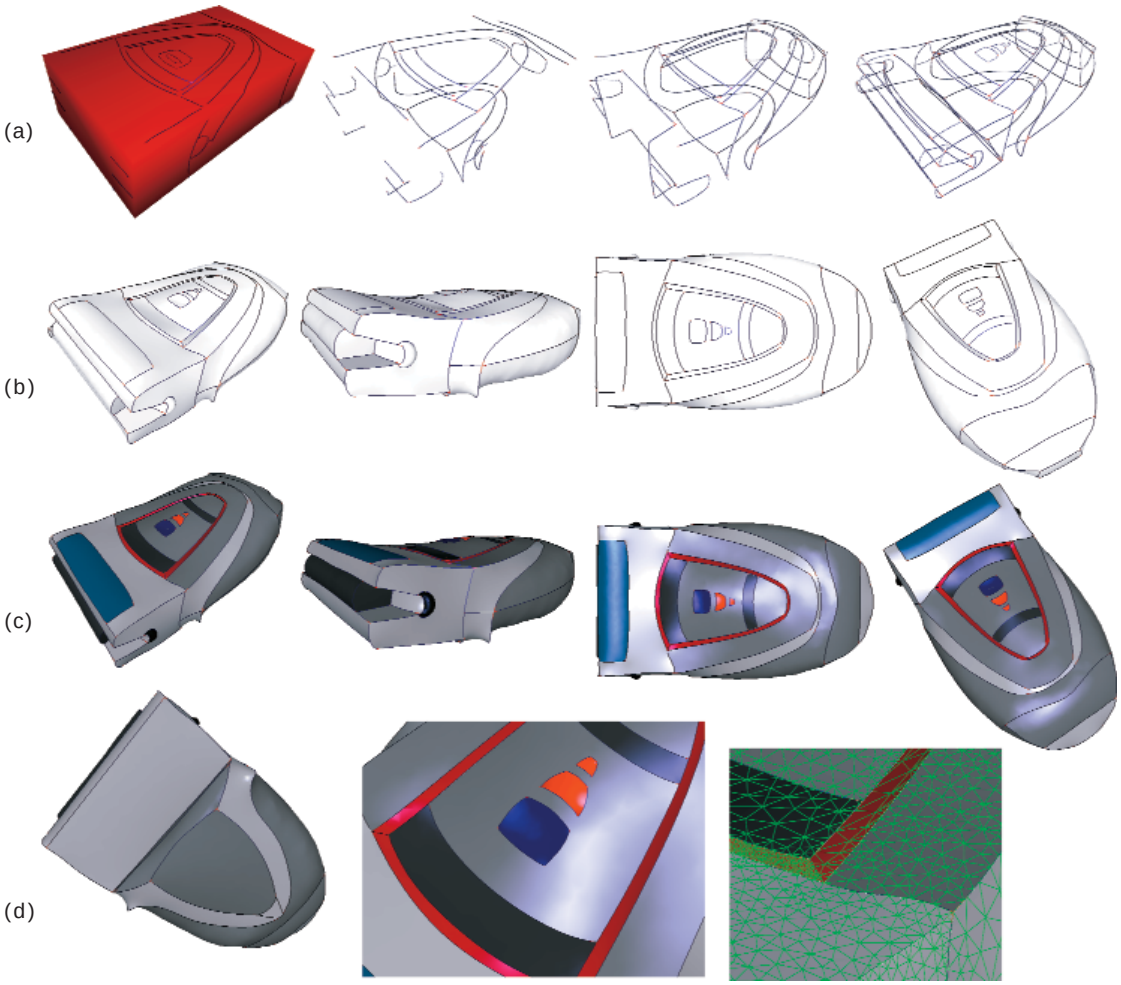
Surface construction and modification

In the last step, the newly designed wireframe is surfaced to obtain a solid model. Once the initial surfaces are obtained, the user can modify them using simple deformation tools. This section gives a brief overview of our surfacing operations. A detailed exposition of these techniques can be found in our previous work.[1]

Initial surface creation

Given the wireframe model, this step creates a surface geometry for each of the face loops in the wireframe. A face loop is defined as a closed boundary formed by a set of wireframe curves. While the face loops could be computed automatically using the techniques presented elsewhere,[4] in our current system we ask the user to assist this process by manually marking the curves involved in a particular face loop. Each face loop can consist of an arbitrary number of curves. The surfaces laid across the face loops are all polygonal surfaces consisting of purely triangular elements.

We create a surface geometry across a face loop as follows. In the first step, our program creates a vertex at the centroid of the boundary vertices. The program then creates initial triangles that use the new vertex as the apex, and have their bases at the boundary. The resulting crude triangulation serves as an initial base surface. Next, we perform a series of edge swapping, triangle



10 Design of an electric shaver. (a) Wireframe creation. (b) Surfaced model rendered in white. (c) Surfaced model painted. (d) Bottom and top faces, and a detail view of the polygonal surfaces.

subdivision, and Laplacian smoothing operations until we obtain an adequate number of regularly distributed triangular elements. The number of triangles obtained this way is dictated by a user-controlled threshold that specifies the maximum edge length as a percentage (currently 20 percent) of the average edge length occurring during the initial crude triangulation. In other words, triangle subdivision is iteratively carried out until the length of the longest edge in the entire set of triangular faces is shorter than 20 percent of the average edge length appearing in the initial triangulation.

Out of infinitely many surfaces that could be generated across the face loop, the surface obtained with this method has the unique property of having the minimum surface area, thanks to the application of the Laplacian smoothing between each iteration. Figure 9a shows an example of a surface obtained in this way.

Surface modification

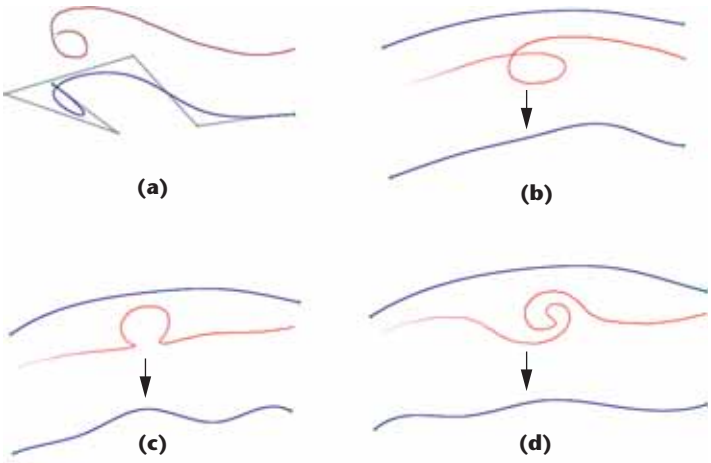
Our system offers two physically based deformation tools to modify the initial surfaces. In both methods, we apply deformation to the interior of the surface while keeping the boundaries fixed. This approach preserves the underlying wireframe geometry, with no alterations to the designed curves.

The first method simulates the effect of a pressure force on a thin membrane to deform a surface. With this tool, users can modify the initial surfaces to create round and inflated surfaces exhibiting more volume. The deformation's extent depends on the pressure magnitude, which is controlled by the user through a slider bar. Users can specify different pressure values for individual surfaces, thus giving the user a better control on the final shape of the solid model. Figure 9b shows a surface deformed in this way.

Inspired by the physical behavior of a network of mechanical springs, the second method produces surfaces in which the variation of curvature is minimized. In this method, a spring is attached to each surface vertex. Each spring is initially oriented parallel to the normal direction of its corresponding vertex. The spring length approximately represents the local curvature. During modification, the springs work to keep their lengths equal, which is equal to minimizing the variation of curvature. Each vertex thus moves under the influence of its neighbors until the vertices locally lie on a sphere. This scheme produces fair surfaces that vary smoothly, which we consider an important factor for aesthetic design purposes. Additionally, when applied to a group of adjacent surfaces, it reduces sharp edges by smoothing the transition across the boundary curves.

Example and discussions

Figure 10 shows snapshots of our system in the design of an electric shaver. The base template used for this model is a $6 \times 3 \times 10$ rectangular prism, as Figure 10a



11 **Current limitations of our curve design methods. (a) B-spline fitting with seven control points is not sufficient to reliably reproduce the sketched curve. (b) The original curve (blue) is being modified by a modifier with self-intersection (red), producing erroneous results. (c-d) Similar failures for modifiers with tight bends and hooks.**

shows. The design begins by laying down several curves on this prism. Next, the initial curves are modified to give them the appropriate 3D shape. During the construction of the wireframe, users might sometimes delete curves if they are deemed premature at a given time. For instance, initially we drew the part of the curves making up the three buttons on the top surface on the template prism, but later removed them in the early stages of curve modification. We then introduced them toward the end of the design process.

The rightmost image of Figure 10a shows the final wireframe designed via our system. Figure 10b shows different views of the solid model obtained after surfacing. Finally, Figure 10c and d show different views of the model in which individual surfaces have been painted.

The final wireframe consists of 107 individual curves. Out of these 107 curves, 37 had symmetrical pairs. Therefore, the user actually designed only 70 curves. The surfaced model consists of 50 individual surfaces, most of which were modified using the surface deformation tools described earlier. This model took one of the authors about 2 hours to design, excluding the time expended on surface painting. Our subjective assessment indicates that the majority of the design time is taken by curve modification and detailing, while initial curve creation and surfacing are relatively rapid processes.

Computation times

We obtained the following results on a Windows-based 2-GHz machine with 1 Gbyte of RAM during actual design sessions. Our curve creation techniques (both on and off the surface template) are sufficiently fast with no apparent lag during live sessions. Likewise, our single- and two-view modifications are sufficiently fast, yielding results with no delay even when the target curve to be modified has not been specified—that is, our program needs to identify the target curve by projecting all wireframe curves to the image plane and finding the nearest one to the input strokes. However, while not verified experimentally, we suspect there might be a slight delay up to a few seconds while identifying the target curve, as the number of wireframe curves markedly increases.

Given a boundary loop, creating the initial surface geometry with triangular faces is computationally more expensive and may take up to a few seconds. More particularly, it took 1.1 seconds to generate a surface with 423 vertices and 694 triangular faces, while it took 7 seconds to generate a surface with 1,298 vertices and 2,194 faces. Once created, surface modification via pressure force is relatively rapid with 0.4 and 1.7 seconds for the previous two surfaces respectively. Our surface fairing using the curvature minimization approach is slightly more expensive with 0.6 and 3.1 seconds for the two surfaces respectively.

Implementation choices and limitations

Our curve creation and modification methods are most suitable for designing globally smooth curves where small and complex artifacts along the curve don't exist. We based this choice on the observation that for the styling design of the types of objects we consider, the characteristic curves that form the perceived style often are devoid of such artifacts. The number of control vertices of the B-splines we use during curve creation and the various weights that control the curve smoothness in our snake-based curve modification method are thus set to reflect these choices.

Based on these choices, Figure 11 shows cases that our current approach does not handle well. In all cases, the current methodology results in a loss of the intended features. Some of the discrepancy between the intended and resulting curves can be attributed to the choices mentioned previously, such as the number of control points during B-spline fitting (see Figure 11a). More critically, however, the nature of our snake-based modification scheme makes it difficult to deal with curves with self-intersections (see Figure 11b), or those with tight bends and hooks (see Figures 11c and 11d). One implication of this issue during design sessions is that it precludes us from drawing a 3D curve whose projection to the current image plane is self-intersecting—for instance, a helix. Nevertheless, other than this latter case, we suspect users would encounter the cases shown in Figure 11 rather infrequently during actual styling design scenarios.

Currently, the snake-based curve modification algorithm assumes that a target curve is modified in its entirety. That is, local modifications to a curve are not permitted. Likewise, the current approach does not allow two or more curves connected in series to be modified by a single set of modifiers. We plan to alleviate this difficulty by extending our snake-based modification algorithm to enable local modifications to a single curve, and global modifications to two or more curves.

Similarly, the user currently creates and modifies each surface patch independently from other surface patches. While this provides substantial flexibility to users during surfacing, it also has the downside that users do not have control over the surface continuity across boundary curves (except for the trivial G^0 continuity).

Our future plans involve providing a mechanism to the user to specify continuity constraints across different surface patches.

Conclusions

To date, we have tested our techniques by designing a variety of different models with favorable outcomes. All of our techniques perform at interactive rates, though some processes (especially those involving surface design) might take up to a few seconds. While our preliminary results have shown our system to greatly facilitate 3D shape modeling, we plan to conduct a more thorough assessment of our techniques with real designers, which is the subject of our future work. ■

References

1. L.B. Kara, C. D'Eramo, and K. Shimada, "Pen-Based Styling Design of 3D Geometry Using Concept Sketches and Template Models," *Proc. ACM Solid and Physical Modeling Conference*, ACM Press, 2006, pp. 149-160.
2. M. Kaas, A. Witkins, and D. Terzopolus, "Snakes: Active Contour Models," *Int'l J. Computer Vision*, vol. 1, no. 4, 1988, pp. 312-330.
3. L. Piegl and W. Tiller, *The NURBS Book*, 2nd ed., Springer-Verlag, 1997.
4. K. Inoue, *Reconstruction of Two-Manifold Geometry from Wireframe CAD Models,* doctoral dissertation, Univ. of Tokyo, 2003.

Levent Burak Kara *is a postdoctoral research associate in the Department of Mechanical Engineering at Carnegie Mellon University. His research interests include pen computing, geometric modeling, human–computer interaction, and artificial intelligence. Kara has a BS in mechanical engineering from the Middle East Technical University, and an MS and PhD in mechanical engineering from Carnegie Mellon University. Contact him at lkara@andrew.cmu.edu.*

Kenji Shimada *is a professor of mechanical engineering, biomedical engineering, and robotics at Carnegie Mellon University. His research interests include geometric modeling, mesh processing, computer graphics, and medical robotics. Shimada has a BS and MS from the University of Tokyo, and a PhD in mechanical engineering from the Massachusetts Institute of Technology He is a member of the IEEE, ACM, American Society of Mechanical Engineers, Japan Society for Industrial and Applied Mathematics, Society of Automotive Engineers, and Society for Industrial and Applied Mathematics.*