

# Construction and Modification of 3D Geometry Using a Sketch-based Interface

Levent Burak Kara<sup>†1</sup> and Kenji Shimada<sup>‡1</sup>

<sup>1</sup>Mechanical Engineering Department, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213

---

## Abstract

*We present an interactive pen-based computer program for designing 3D objects through direct sketching. The proposed techniques are tailored toward the creation of free-form curves and surfaces, and are therefore particularly useful for styling purposes. In our approach, the design process consists of two main steps. In the first step, the user designs a wireframe model by sketching its constituent curves in 3D. Using purely sketch-based operations, the initial curves can then be modified as desired. In the second step, the user constructs interpolating surfaces on the wireframe to obtain a solid model. Again, through sketch-based operations, the user can modify the initial surfaces, and specify the boundary conditions if necessary. In addition to the main modeling operations, a gesture-based command interface allows many of the frequently used commands to be input through pen strokes. The utility of our system is demonstrated with various examples.*

Categories and Subject Descriptors (according to ACM CCS): H.5.2 [User Interfaces]: Graphical User Interfaces (GUI) Pen-based interaction; I.3.5 [Computational Geometry and Object Modeling]: Curve, surface, solid, and object representations, Physically based modeling.

---

## 1. Introduction

We describe a sketch-based design tool for the construction and modification of 3D geometry. Users of our system can design a variety of objects consisting of relatively complex edge and surface geometries. A key advantage of our system is that resulting geometry is directly dictated by input strokes thus making our system suitable for product styling design. This is in contrast to systems that use input strokes as gestures to modify primitives in certain directions, or those that use indirect manipulation methods based on handles or control lattices.

Our system supports a variety of pen-based operations both for modeling and command inputting. In a typical scenario, the user begins by constructing the wireframe of the design object. For this, the user simply sketches the constituent curves. Input strokes are first beautified into smooth curves in the image plane, and are then projected into 3D

to form the wireframe. Initially created curves can be later modified by simply sketching their new shapes. For curve modification, our program uses a physically-based deformation technique that modifies the original curve until it best conforms to the new shape dictated by input strokes. If desired, connectivity constraints between different curves can be set by simple pen operations. After the desired wireframe is obtained, the user constructs interpolating surfaces that cover the wireframe. Guided by sketch input, initial surfaces can then be smoothly modified to give them new shapes. Boundary conditions across different surfaces can also be specified through direct sketching of tangent planes. At any point during the design cycle, a trainable, gesture-based command interface allows frequently used commands to be specified with simple pen gestures.

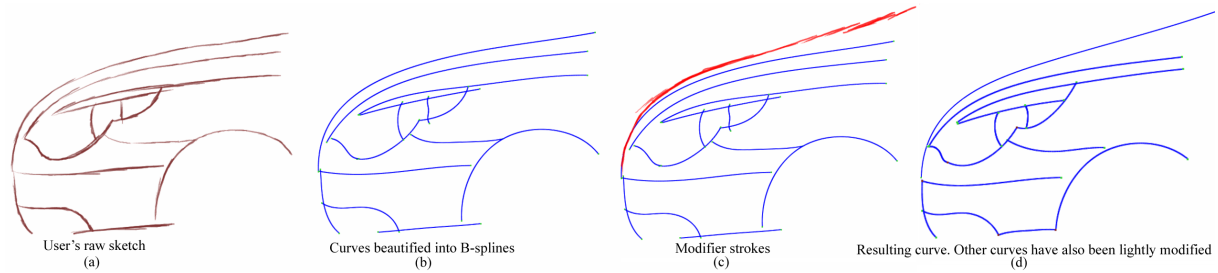
## 2. Related Work

While most 3D modeling software traditionally evolved around a windows-mouse-menu-based interaction paradigm, recent advances in stylus-enabled tablet technology has made sketch-based interaction an appealing alternative. To

---

<sup>†</sup> e-mail: lkara@andrew.cmu.edu

<sup>‡</sup> e-mail: shimada@cmu.edu



**Figure 1:** Illustration of curve creation and modification in 2D. Curves are initially created by beautifying input strokes into B-splines. Once created, a curve can be modified by simply sketching its new shape near the original curve.

date, researchers have developed a variety of sketch-based 3D modeling systems in various domains [ZHH96, IMT99, KHR02, TBSR04, BCCD04, DDGG05, MKL05]. In gesture-based approaches such as [ZHH96, EBE95, HQ03, DE03], designers' strokes are used primarily for geometric operations such as extrusion, bending, and primitive modification. Silhouette-based approaches [IMT99, KHR02, BCCD04, SWSJ05, CSSJ05] enable free-form surface generation. In these methods, users' strokes are used to form a 2D silhouette representing an outline or a cross-section, which is then extruded, inflated or swept to give 3D form. Systems such as [KG05, CCP\*04, NSACO05] allow users to directly operate on existing surfaces to deform or add features lines using a digital pen. The key difference of these systems compared to gesture-based interfaces is that users' strokes are directly replicated in the resulting shape. However, these systems are most useful during later design stages where the main geometry is already available. Optimization-based algorithms such as [MKL05] produce the most plausible 3D shape from a 2D sketch of its wireframe. Line-labeling techniques have also been explored for 3D shape construction from 2D input. While many previous systems were limited to straight-edge models with planar surfaces, recent systems such as [PYJH04] have begun to extend these techniques to curved edges. Template based methods such as [MSK00, KDS06] allow the desired 3D form to be obtained by deforming an underlying 3D template.

### 3. Wireframe Creation and Modification

In the first step of the design process, the user creates a wireframe model by sketching the constituent curves in 3D. Users are allowed to sketch each curve using an arbitrary number of strokes, drawn in arbitrary directions and order. During curve construction, input strokes are first beautified into B-splines in the image plane using a curve fitting algorithm. The curves obtained in the image plane are then projected back into 3D to yield the final wireframe model. Once the initial curves are laid out, the user may modify each curve using a physically-based deformation algorithm. If desired, the user can specify connectivity constraints between

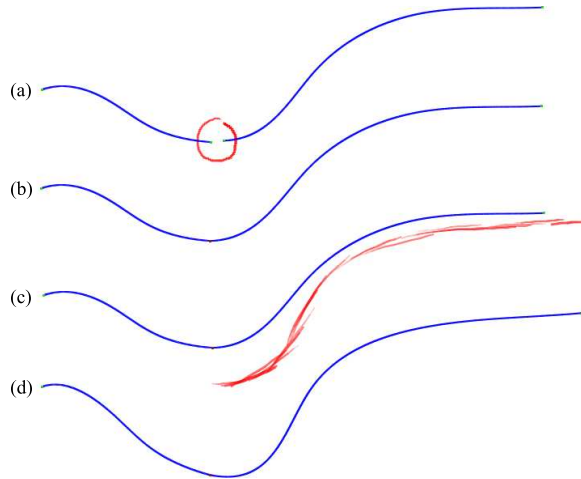
individual curves. To facilitate discussion, we first describe our techniques in a 2D environment. We then describe how we extend the fundamental principles to 3D.

#### 3.1. An Illustrative Example in 2D

Our curve creation and modification techniques are both purely sketch-based. Figure 1 illustrates the main steps involved in a typical design scenario. To begin, the user first sketches a rough outline of the design object. When creating a curve, users are free to use an arbitrary number of strokes, drawn in arbitrary directions and order. However, the user is asked to indicate the separation between the stroke groups that make up different curves. In Figure 1a, for instance, the user may draw the front hood using as many strokes as desired, but must inform the program (currently by tapping a button) when moving onto, say the wheel well.

Our program beautifies each stroke group into a cubic B-spline using a minimum least-squares curve fitting algorithm described in [PT97]. As shown in Figure 1b, resulting curves closely approximate the input strokes. After creating the initial curves, the user can set the system in 'modification' mode and modify each curve by simply sketching the curve's new shape. Figure 1c shows an example where the user has sketched several strokes above the hood. We call these strokes as *modifiers* as their purpose is to modify existing curves rather than to create new ones. After drawing the modifiers, the user invokes the 'process' command by gesturing a checkmark<sup>†</sup>. With this, our program first determines the curve that the user is intending to modify. This is done by identifying the curve that is closest to the modifiers, e.g., the hood in Figure 1. Next, our program uses an energy minimization algorithm based on active contours [KWT88] to deform the original curve until it best conforms to the modifiers. In this formulation, the original curve is treated as a physical spline that works to minimize both an internal energy term arising from stretching and bending, and an

<sup>†</sup> A detailed description of our gesture interface is given in Section 5.



**Figure 2:** *Joining curves. (a-b) A counter clockwise ‘o’ gesture joins curves whose ends lie inside the gesture. (c-d) Once connected, subsequent modifications to one curve induce complying modifications in others in the group to preserve connectedness.*

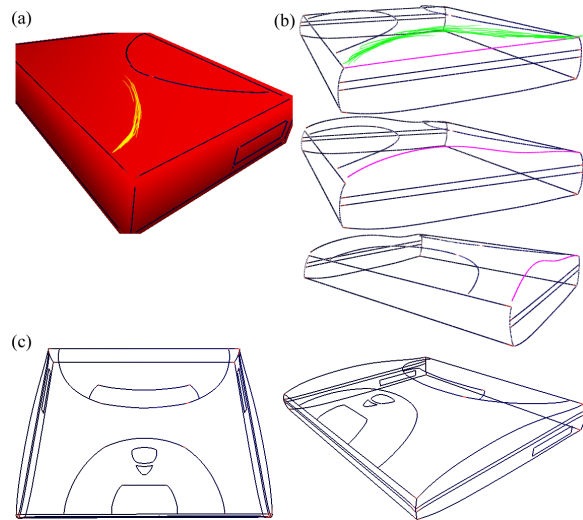
external energy term arising from the modifiers’ presence. The internal energy forces the curve to evolve as a smooth curve. The external energy can be thought of a driving potential which decreases as the original curve approaches the modifiers. The balance between smoothness versus compliance of the final curve can be conveniently controlled by adjusting the corresponding weights in the energy functional. Figure 1d shows the hood modified in this way.

When desired, two or more curves can be joined at their ends with a counter clockwise ‘o’ gesture as shown in Figure 2. This operation applies a set of rotations and scalings to the curves until their ends meet at the same point. It also establishes the connectivity between the curves in that subsequent modifications to one curve induces modifications on the other curves to keep the group connected. When necessary the curves can be detached by a clockwise ‘o’ gesture.

### 3.2. Wireframe Creation in 3D

We use the same principles described above for 3D modeling. Users begin the design by sketching the curves of the wireframe. The main difference, however, is that we facilitate 3D modeling through the use of an underlying template model. This template acts as a platform that helps anchor users’ initial strokes in 3D space, and is typically a very simplified model of the design object in question. For instance a thin rectangular prism serves as a suitable template for the design of a laptop computer as shown in Figure 3a.

With the presence of the template, the curves originally constructed in the image plane using B-spline fitting are uniquely projected into 3D using a standard ray intersection



**Figure 3:** *Curve creation and modification in 3D. (a) An initial 3D template helps anchor input strokes in 3D. (b) A curve can be modified by simply sketching its new shape. The optimal 3D configuration is determined by minimizing the deviation from the original curve in 3D, while closely approximating input strokes in 2D. (c) Final wireframe model.*

algorithm. At the end, a set of 3D curves is obtained whose projections to the image plane match the input strokes. 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 greatly lessens the work involved in the subsequent step of wireframe modification. Additionally the use of a template helps circumvent the well-known challenge of one-to-many mapping in 3D interpretation from 2D input.

### 3.3. Modification in 3D

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. To modify a curve, the user simply sketches the modifier strokes that specify the new shape of the curve as it would occur from the current viewpoint. With this, our system modifies the curve in three steps. In the first step, the target curve is identified by projecting the existing 3D curves to the image plane, and determining the curve that lies spatially nearest to the modifiers. In the second step, our system uses the active-contour-based energy minimization algorithm described earlier to deform the projected curve in the image plane until it conforms to the modifiers. Finally, the newly obtained 2D curve is projected back into 3D resulting in the new 3D curve. Figure 3b shows an example. A key challenge here is that there are infinitely many such

back-projections into 3D. We must therefore determine the best 3D configuration by constraining the problem. In our approach we use the following constraints:

- The 3D curve should appear right under the modifiers.
- If the modifier strokes appear precisely over the original target curve, i.e., 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. By “reasonable,” we mean 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 3D configuration as the one that minimizes the spatial deviation from the original 3D 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. For this, a surface that originates from the current eye position, passes through the modifiers, and extends into the page, is first computed. Theoretically, all candidate solutions lie on this surface. The optimal 3D curve is then found by computing the minimum distance projection of the original curve onto this surface. Further details of our algorithm can be found in [KDS06].

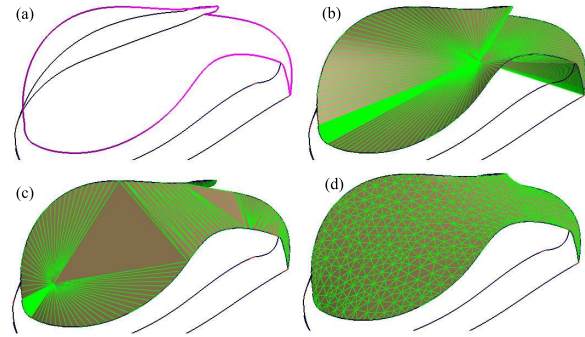
By remaining proximate to the original curve, the new curve can be thought to be “least surprising” when viewed from a different viewpoint. One advantage of this is that curves can be modified incrementally, with predictable outcomes in each step. That is, as the curve desirably conforms to the input 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.

#### 4. Surface Creation and Modification

Once a wireframe model is obtained, the user constructs interpolating surfaces to obtain a solid model. Initially created surfaces can later be modified to the desired shape. The following paragraphs detail these processes.

##### 4.1. Initial Surface Creation

Given the wireframe model, the goal in surfacing is to construct a surface geometry for each of the closed face loops of the wireframe. Figure 4 illustrates the process. For each surface to be created, the user first identifies the associated face loop by highlighting the constituent wireframe curves involved in that face loop (Figure 4a). With this, our system creates an interpolating surface in three steps. First, a vertex is created at the centroid of the boundary vertices. A set of initial triangles are then created that use the new vertex as the common apex, and have their bases at the boundary (Figure 4b). Finally, a series of edge swapping, face subdivision



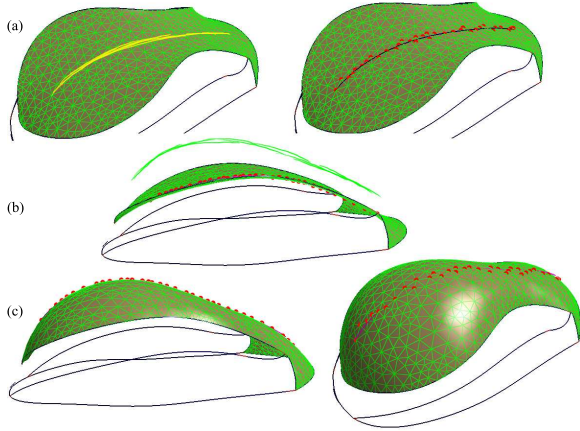
**Figure 4:** Surface creation on a face loop. (a) Highlighted boundary curves, (b) Centroid vertex and associated initial triangulation, (c) An intermediate state during edge swapping, (d) Resulting surface after edge swapping, face subdivision and Laplacian smoothing.

and Laplacian smoothing is applied until a sufficient number of uniformly distributed triangles are obtained (Figure 4d). The result is a smooth polygonal surface consisting of purely triangular elements. For a given face loop geometry, the resulting surface has the unique property of having the minimum surface area due to the nature of Laplacian smoothing.

##### 4.2. Specifying Surface Deformations

Initially created surfaces can be modified to give them the desired shape. To deform a surface, the user first sketches a *reference curve* on it as shown in Figure 5a. This curve defines a region of interest in which the surface vertices closest to the reference curve are selected for subsequent deformation. Next, the user sketches a new curve that specifies the desired shape of the reference curve in 3D (Figure 5b). The 3D configuration of this new curve is computed by modifying the reference curve using the methods described in Section 3.3. Given the initially selected surface vertices and the new curve, our system invokes an optimization algorithm that deforms the surface until the selected surface vertices lie close to the target shape (Figure 5c). During deformation, the objective function we minimize is the Hausdorff distance [Ruc96] between the surface vertices, and the points comprising the target curve. The Hausdorff distance provides a convenient metric that reveals the spatial proximity between two point sets in the form of an upper bound: If the Hausdorff distance is  $d$ , all points in the first point set are at most distance  $d$  away from the other point set (and vice versa).

At the heart of our deformation mechanism is an intuitive method that simulates the effect of a pressure force on a thin membrane (see [KDS06] for details). This tool allows surfaces to be inflated or flattened in a predictable way. A key advantage of this technique is that surfaces can be deformed wholistically in a smooth way without generating



**Figure 5:** Surface deformation. (a) A sketched reference curve and selected surface vertices, (b) The user sketches the new desired shape, (c) The surface is inflated using a pressure force until the surface closely approximates the new shape.

unintended creases. The extent of the deformation depends on the magnitude of the pressure, whose optimal value is determined by our optimization algorithm. In other words, our algorithm seeks for the optimal pressure value which produces a deformation that minimizes the Hausdorff distance mentioned above. Figure 5c shows the result of a surface deformed using this technique.

### 4.3. Specifying Boundary Conditions

If desired, the user can also specify the boundary conditions along surface edges. Normally, it is assumed that the user wishes the surface boundaries to interpolate the wireframe edges, and hence position constraints are already implicit in the wireframe model. Since created surfaces automatically interpolate the boundary curves, position constraints are readily satisfied. The user, however, can specify tangent directions along various parts of the boundary by sketching the silhouettes of the tangent planes at desired points. Figure 6 shows an example. For this, the user first marks a boundary vertex, across which the tangent will be specified (Figure 6a). Next, after transforming to a suitable viewpoint, the user sketches the tangent plane as it would be seen from the side (Figure 6b). This defines a plane in 3D that passes through the input strokes, and extends into the page along the current viewpoint. The normal vector of this virtual plane is then set as the normal of the vertex under consideration (Figure 6c). Note that the normal of a vertex provides precisely the same information as the tangent plane through the vertex.

We only require the user to specify the normal directions (in the form of tangent planes) at a handful of discrete vertices. The normal directions at intermediate vertices along

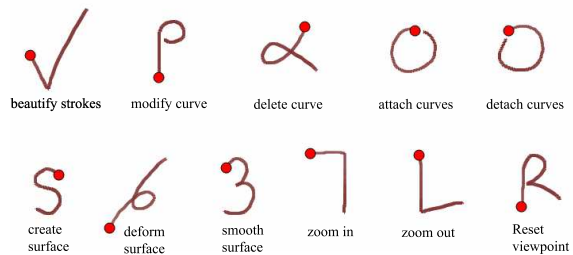
the boundary are identified using a weighted linear averaging function. For instance, the normal vector at an intermediate vertex  $q$  is computed as the weighted average of two surrounding vertices  $p$  and  $r$  as:

$$\mathbf{n}_q = (w)\mathbf{n}_p + (1 - w)\mathbf{n}_r,$$

where  $w$  is distance (along the boundary) between vertices  $q$  and  $r$  divided by the distance (along the boundary) between vertices  $p$  and  $r$ . With this formulation, normals at intermediate vertices will change smoothly between the normals of  $p$  and  $q$ . Figure 6d shows an example. If necessary, more control on the normal directions can be achieved by increasing the number of vertices for which the normals are explicitly specified. Once specified, these constraints are taken into account by both the surface creation and deformation tools described above, resulting in surfaces that conform to the constraints. Note that specification of the boundary normals naturally results in our surfaces to be  $G^1$  continuous across the boundary edges.

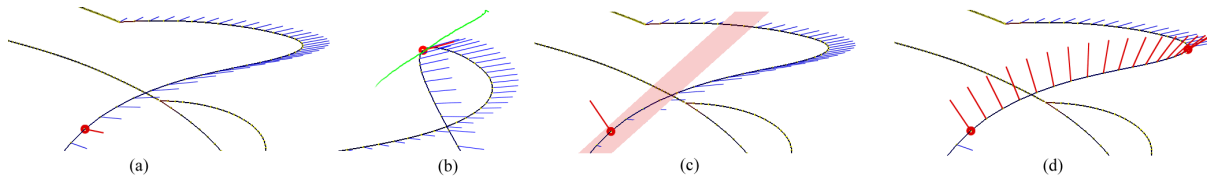
## 5. Command Gestures

Besides the main modeling operations described so far, our system also provides a gesture-based command interface for inputting frequently used commands. The command interface is extensible and customizable in that new gestures can be trained, or existing gestures can be linked to different commands. Figure 7 shows the currently used gestures and their associated commands. Each gesture is a single stroke entity that is sensitive to orientation and drawing direction. Due to variation in drawing speeds, input raw strokes frequently consist of data points spaced non-uniformly along the stroke's trajectory (low pen speeds cause dense point clouds while high pen speeds cause large gaps between points), which adversely affect their recognition. To alleviate this difficulty, input strokes are first resampled using a linear interpolation function to obtain data points equally spaced along the stroke's trajectory.



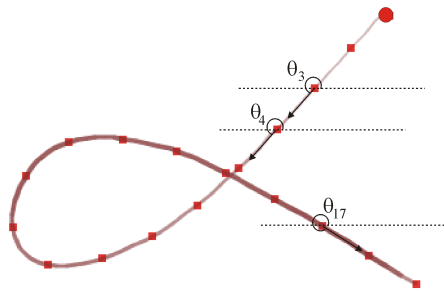
**Figure 7:** Currently used gestures and their associated commands. Red dots indicate the starting points of the gestures.

Our gesture recognizer employs neural networks to learn and classify gestures. For each gesture, a separate network is constructed. The networks in our approach use the angles



**Figure 6:** Specifying boundary normals. (a) A selected boundary vertex and its default normal. (b) User sketches the tangent plane across this vertex from a suitable viewpoint. (c) New vertex normal. Notice that the new normal is the normal of the virtual plane generated in b. (d) The user has specified the normals of two vertices. The normals of intermediate vertices are interpolated smoothly between the two vertex normals.

formed by the horizontal line, and the line segments connecting consecutive points as the input feature vector. Figure 8 shows these angles. The entries in this vector suitably range between  $[-\pi, +\pi]$ , thereby providing a congruent range to the input layer of the networks in all cases. The nature of this feature vector makes gesture recognition sensitive to orientation and drawing direction, but insensitive to scale. While additional geometric features could be considered to characterize a stroke more specifically, we have found the proposed angle information to be sufficiently discriminatory for our purposes.



**Figure 8:** The angles between the line segments in a gesture and the horizontal line form the feature vector for the neural network during gesture recognition.

Each network in our command interface is designed as a fully-connected, feed-forward back propagation neural network with 18 inputs, 16 neurons in the first hidden layer, 6 neurons in the second hidden layer, and 1 output neuron. The inputs to the network are the entries of the feature vector described above<sup>‡</sup>. In each neuron, a tangent sigmoid ('tansig') function is used as the activation function. The output of the network is a single neuron which outputs a real number in the range  $[0,1]$ . For each gesture, the network is trained using around 200 positive and 200 negative examples. During training, a target value of 1.0 and 0.0 is assigned to positive

<sup>‡</sup> Input strokes are thus resampled to 19 data points, which results in 18 angles that form the input vector.

and negative examples respectively. During recognition, an unknown stroke is evaluated by each of the neural networks. The stroke is classified as the gesture whose network produces the highest real value at the output neuron. In actual use, the classification of a stroke by the ensemble of networks is almost instantaneous.

A key issue in gesture recognition however is that, prior to deciding *which* gesture a stroke represents, it is necessary to decide *if* the stroke is a gesture in the first place. This issue is discussed in detail in [SL03] on modeless input. The free-form sketching nature of our modeling operations makes this initial distinction a much challenging task, as the strokes used during modeling operations can be easily mistaken for a gesture and vice-versa. To avert this difficulty, we ask the user to hold a modifier button on the tablet or the keyboard when gesturing.

## 6. Examples

Figure 9 shows several models designed using our system. In each case, the user starts the design on a simple template. In the case of the laptop computer and the shaver, starting templates are simply rectangular prisms. For the mouse, it is a half-egg shape. Normally our system is only loosely dependent on the underlying template. Indeed, the template is only necessary to lay out a rough wireframe model containing a handful of characteristic curves. Once such a wireframe is obtained, its curves can be quickly modified resulting a simple but geometrically accurate wireframe. Next, this wireframe can be surfaced to obtain a set of initial surfaces that define the main style of the design object. Further curves and surfaces can thus be added later using this initial model as a convenient base platform. This is especially useful for adding small details such as buttons or impressions. In the examples shown above this strategy has been used extensively, and has proven to be quite effective. To further facilitate design, our program provides the option to preserve symmetry across one of the three principal Cartesian planes. This way, work performed on one side of the symmetry plane is automatically duplicated on the other side. The above examples take advantage of this feature.

All processes described in earlier sections are performed

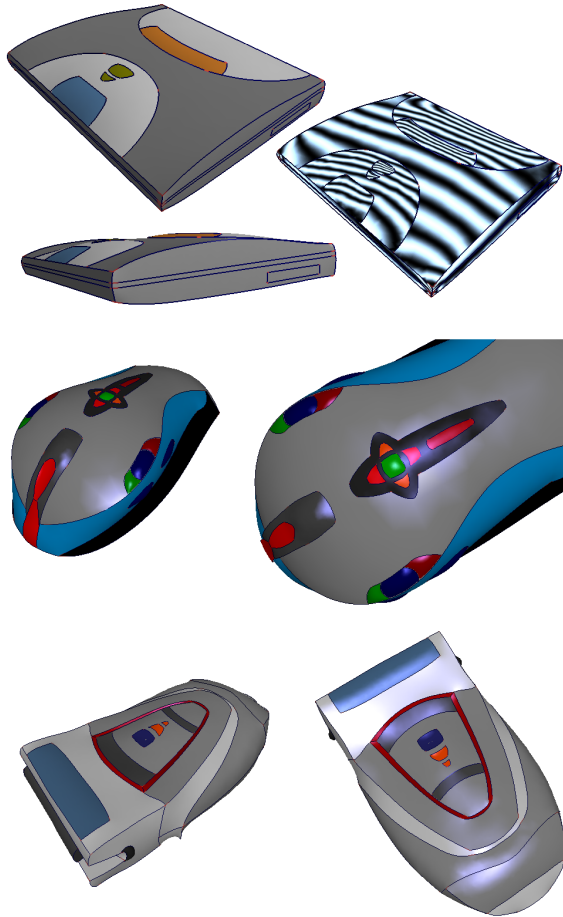


Figure 9: Example models created by our system.

at interactive speeds on a consumer level PC (2.0GHz with 1MB of RAM), except for the optimization-based surface modification process. In this case, our experiments have shown that it takes anywhere between 5 to 30 seconds to determine the optimal surface deformation.

## 7. Conclusions and Future Work

We have described a new approach to computer aided design of 3D geometry using a sketch-based interface. Our techniques are particularly useful for *styling design*, where a fully sketch-based interaction is an appealing natural alternative to traditional menu-and-mouse-based interaction. Our system supports a wide range of sketch-based 3D modeling operations including wireframe construction through curve creation, wireframe modification, surface creation, surface deformation, and boundary condition specification. Besides these main modeling operations, a trainable gesture-based command interface allows many of the frequently used commands to be invoked via a set of stroke gestures. With the

proposed techniques, users of our system can design a wide variety of 3D models exclusively through the use of a digital pen. We believe the proposed techniques demonstrate that a sketch-based interaction is a viable alternative to the traditional interaction mechanisms found in existing software. Moreover, it is particularly advantageous for styling design purposes where typical tasks would be tedious, if not complicated, using conventional interaction techniques.

While our current system is an effective tool, there are several directions for future improvements. Currently our curve modification algorithm requires curves to be modified individually, and each curve must be modified in its entirety. This means local modifications to a curve are currently not permitted. We plan to extend our curve modification techniques to allow local curve modifications. Also, our current surface deformation tool uses a uniform pressure field to deform a surface. We are currently exploring other means, such as non-uniform pressure fields, for a more flexible control of the deformed surfaces. Also, while our techniques are tailored toward the creation of *curvy* edges and surfaces, we plan to incorporate options for creating more standard geometry such as straight lines, arcs, fillets etc. Finally, we are planning to conduct field studies with real industrial designers to better assess our system, and to see how it can be improved.

## References

- [BCCD04] BOURGUIGNON D., CHAINE R., CANI M.-P., DRETTAKIS G.: Relief: A modeling by drawing tool. In *EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling* (2004).
- [CCP\*04] CHEUTET V., CATALANO C., PERNOT J., FALCIDIENO B., GIANNINI F.: 3d sketching with fully free form deformation features (d-f4) for aesthetic design. In *EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling* (2004).
- [CSSJ05] CHERLIN J. J., SAMAVATI F., SOUSA M. C., JORGE J. A.: Sketch-based modeling with few strokes. In *SCCG '05: Proceedings of the 21st spring conference on Computer graphics* (2005), ACM Press, pp. 137–145.
- [DDGG05] DAS K., DIAZ-GUTIERREZ P., GOPI M.: Sketching free-form surfaces using network of curves. In *EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling* (2005).
- [DE03] DRAPER G., EGBERT P.: A gestural interface to free-form deformation. In *Graphics Interface 2003* (2003), pp. 113–120.
- [EBE95] EGGI L., BRUDERLIN B. D., ELBER G.: Sketching as a solid modeling tool. In *SMA '95: Proceedings of the third ACM symposium on Solid modeling and applications* (1995), ACM Press, pp. 313–322.
- [HQ03] HUA J., QIN H.: Free-form deformations via

- sketching and manipulating scalar fields. In *SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications* (2003), ACM Press, pp. 328–333.
- [IMT99] IGARASHI T., MATSUOKA S., TANAKA H.: Teddy: a sketching interface for 3d freeform design. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (1999), pp. 409–416.
- [KDS06] KARA L. B., D'ERAMO C., SHIMADA K.: Pen-based styling design of 3d geometry using concept sketches and template models. In *ACM Solid and Physical Modeling Conference* (2006).
- [KG05] KHO Y., GARLAND M.: Sketching mesh deformations. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games* (2005), ACM Press, pp. 147–154.
- [KHR02] KARPENKO O., HUGHES J. F., RASKAR R.: Free-form sketching with variational implicit surfaces. In *Eurographics* (2002).
- [KWT88] KAAS M., WITKINS A., TERZOPOLUS D.: Snakes: active contour models. *International Journal of Computer Vision* 1, 4 (1988), 312–330.
- [MKL05] MASRY M., KANG D. J., LIPSON H.: A free-hand sketching interface for progressive construction of 3d objects. *Computers and Graphics* 29, 4 (2005), 563–575.
- [MSK00] MITANI J., SUZUKI H., KIMURA F.: 3d sketch: Sketch-based model reconstruction and rendering. In *Workshop on Geometric Modeling 2000* (2000), pp. 85–98.
- [NSAC005] NEALEN A., SORKINE O., ALEXA M., COHEN-OR D.: A sketch-based interface for detail-preserving mesh editing. *ACM Transactions on Graphics* 24, 3 (2005), 1142–1147.
- [PT97] PIEGL L., TILLER W.: *The NURBS Book*. 1997.
- [PYJH04] P.A.C.VARLEY, Y.TAKAHASHI, J.MITANI, H.SUZUKI: A two-stage approach for interpreting line drawings of curved objects. In *EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling* (2004).
- [Ruc96] RUCKLIDGE W. J.: *Efficient Visual Recognition Using the Hausdorff Distance*. Number 1173 Lecture Notes in computer Science., Springer-Verlag, Berlin, 1996.
- [SL03] SAUND E., LANK E.: Stylus input and editing without prior selection of mode. In *UIST '03: Proceedings of the 16th annual ACM symposium on User interface software* (2003), ACM Press, pp. 213–216.
- [SWSJ05] SCHMIDT R., WYVILL B., SOUSA M. C., JORGE J. A.: Shapeshop: Sketch-based solid modeling with blobtrees. In *EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling* (2005).
- [TBSR04] TSANG S., BALAKRISHNAN R., SINGH K., RANJAN A.: A suggestive interface for image guided 3d sketching. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems* (2004), pp. 591–598.
- [ZHH96] ZELEZNIK R. C., HERNDON K. P., HUGHES J. F.: Sketch: an interface for sketching 3d scenes. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (1996), pp. 163–170.