Teddy: A Sketching Interface for 3D Freeform Design

Takeo Igarashi[†], Satoshi Matsuoka[‡], Hidehiko Tanaka[†]

[†] University of Tokyo, [‡] Tokyo Institute of Technology

Abstract

We present a sketching interface for quickly and easily designing freeform models such as stuffed animals and other rotund objects. The user draws several 2D freeform strokes interactively on the screen and the system automatically constructs plausible 3D polygonal surfaces. Our system supports several modeling operations, including the operation to construct a 3D polygonal surface from a 2D silhouette drawn by the user: it inflates the region surrounded by the silhouette making wide areas fat, and narrow areas thin. Teddy, our prototype system, is implemented as a Java™ program, and the mesh construction is done in real-time on a standard PC. Our informal user study showed that a first-time user typically masters the operations within 10 minutes, and can construct interesting 3D models within minutes.

CR Categories and Subject Descriptions: I.3.6 [Computer Graphics]: Methodology and Techniques – Interaction Techniques; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling – Geometric algorithms

Additional Keywords: 3D modeling, sketching, pen-based systems, gestures, design, chordal axes, inflation

1 INTRODUCTION

Although much progress has been made over the years on 3D modeling systems, they are still difficult and tedious to use when creating freeform surfaces. Their emphasis has been the precise modeling of objects motivated by CAD and similar domains. Recently SKETCH [29] introduced a gesture-based interface for the rapid modeling of CSG-like models consisting of simple primitives.

This paper extends these ideas to create a sketching interface for designing 3D freeform objects. The essential idea is the use of freeform strokes as an expressive design tool. The user draws 2D freeform strokes *interactively* specifying the silhouette of an object, and the system automatically constructs a 3D polygonal surface model based on the strokes. The user does not have to manipulate control points or combine complicated editing operations. Using our technique, even first-time users can create simple, yet expressive 3D models within minutes. In addition, the resulting models have a hand-crafted feel (such as sculptures and stuffed animals) which is difficult to accomplish with most conventional modelers. Examples are shown in Figure 2.

Figure1: Teddy in use on a display-integrated tablet.

Figure 2: Painted models created using Teddy and painted using a commercial texture-map editor.

We describe here the sketching interface and the algorithms for constructing 3D shapes from 2D strokes. We also discuss the implementation of our prototype system, Teddy. The geometric representation we use is a standard polygonal mesh to allow the use of numerous software resources for post-manipulation and rendering. However, the interface itself can be used to create other representations such as volumes [25] or metaballs [17].

Like SKETCH [29], Teddy is designed for the rapid construction of approximate models, not for the careful editing of precise models. To emphasize this design goal and encourage creative exploration, we use the real-time pen-and-ink rendering described in [16], as shown in Figure 1. This also allows real-time interactive rendering using Java on mid-range PCs without

{takeo, tanaka}@mtl.t.u-tokyo.ac.jp, matsu@is.titech.ac.jp
http://mtl.t.u-tokyo.ac.jp/~takeo

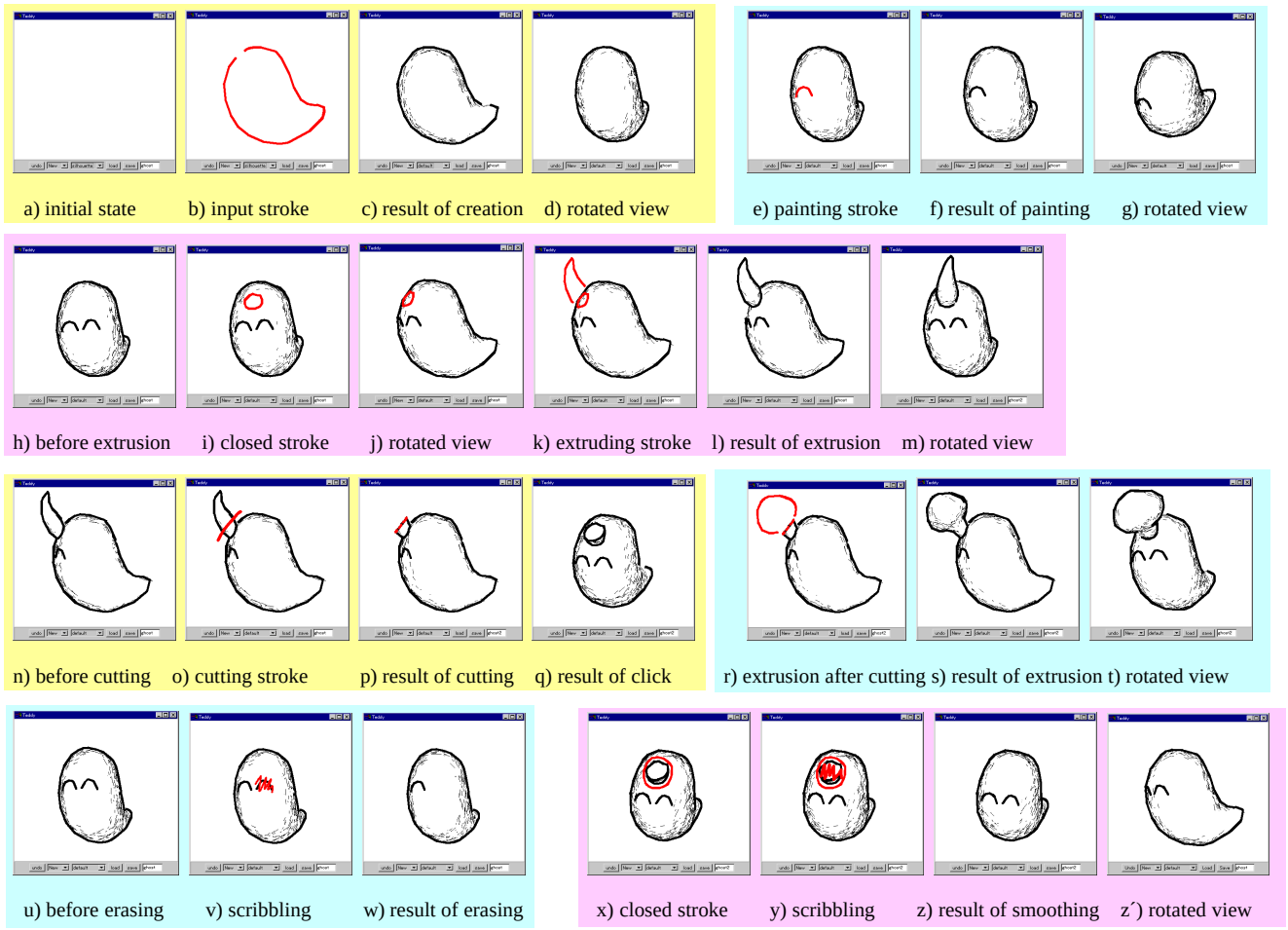


Figure 3: Overview of the modeling operations.

dedicated 3D rendering hardware.

An obvious application of Teddy is the design of 3D models for character animation. However, in addition to augmenting traditional 3D modelers, Teddy's ease of use has the potential to open up new application areas for 3D modeling. Possibilities include rapid prototyping in the early stages of design, educational/recreational use for non-professionals and children, and real-time communication assistance on pen-based systems.

The accompanying videotape demonstrates Teddy's user interface. Teddy is available as a Java applet at the following web site. http://www.mtl.t.u-tokyo.ac.jp/~takeo/teddy/teddy.htm

2 RELATED WORK

A typical procedure for geometric modeling is to start with a simple primitive such as a cube or a sphere, and gradually construct a more complex model through successive transformations or a combination of multiple primitives. Various deformation techniques [15,23] and other shape-manipulation tools [8] are examples of transformation techniques that let the user create a wide variety of precise, smooth shapes by interactively manipulating control points or 3D widgets.

Another approach to geometric modeling is the use of implicit surfaces [3,18]. The user specifies the skeleton of the intended model and the system constructs smooth, natural-looking surfaces around it. The surface inflation technique [17] extrudes the polygonal mesh from the skeleton outwards. In contrast, our approach lets the user specify the silhouette of the intended shape directly instead of by specifying its skeleton.

Some modeling systems achieve intuitive, efficient operation using 3D input/output devices [6]. 3D devices can simplify the operations that require multiple operations when using 2D devices.

Our sketching interface is inspired by previous sketch-based modeling systems [7,29] that interpret the user's freeform strokes and interactively construct 3D rectilinear models. Our goal is to develop a similar interface for designing rounded freeform models.

Inflation of a 2D drawing is introduced in [27], and 3D surface editing based on a 2D painting technique is discussed in [28]. Their target is basically a 2D array with associated height values, rather than a 3D polygonal model.

The use of freeform strokes for 2D applications has recently become popular. Some systems [10,14] use strokes to specify gestural commands and others [2] use freeform strokes for specifying 2D curves. These systems find the best matching arcs or splines automatically, freeing the users from explicit control of underlying parameters.

We use a polygonal mesh representation, but some systems use a volumetric representation [9,25], which is useful for designing topologically complicated shapes. Our mesh-construction algorithm is based on a variety of work on polygonal mesh manipulation, such as mesh optimization [12], shape design [26], and surface fairing [24], which allows polygonal meshes to be widely used as a fundamental representation for geometric

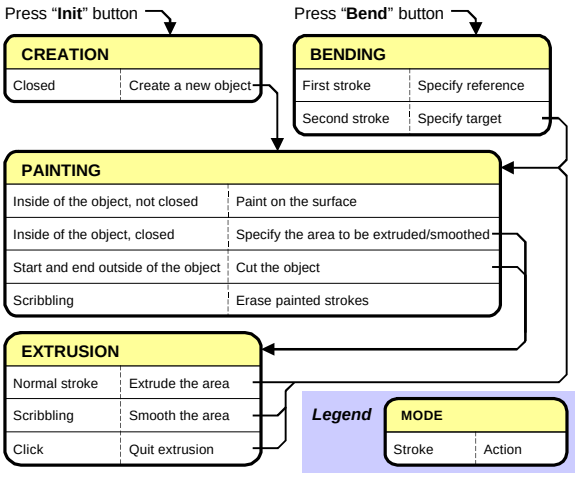


Figure 4: Summary of the gestural operations.

modeling and computer graphics in general.

3 USER INTERFACE

Teddy's physical user interface is based upon traditional 2D input devices such as a standard mouse or tablet. We use a two-button mouse with no modifier keys. Unlike traditional modeling systems, Teddy does not use WIMP-style direct manipulation techniques or standard interface widgets such as buttons and menus for modeling operations. Instead, the user specifies his or her desired operation using freeform strokes on the screen, and the system infers the user's intent and executes the appropriate editing operations. Our videotape shows how a small number of simple operations let the users create very rich models.

In addition to gestures, Teddy supports direct camera manipulation using the secondary mouse button based on a virtual trackball model [13]. We also use a few button widgets for auxiliary operations, such as save and load, and for initiating bending operations.

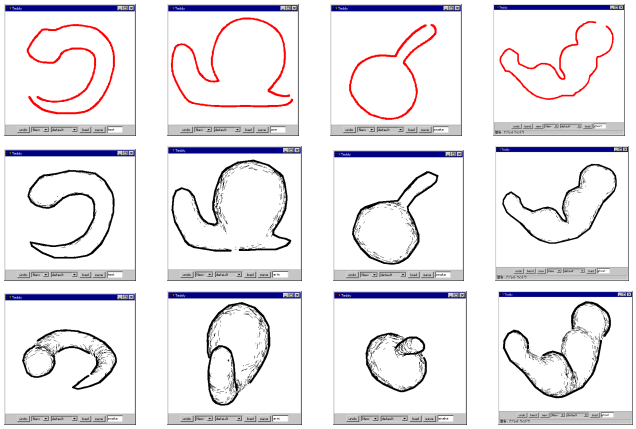
4 MODELING OPERATIONS

This section describes Teddy's modeling operations from the user's point of view; details of the algorithms are left to the next section. Some operations are executed immediately after the user completes a stroke, while some require multiple strokes. The current system supports neither the creation of multiple objects at once, nor operations to combine single objects. Additionally, models must have a spherical topology; e.g., the user cannot create a torus. An overview of the model construction process is given first, and then each operation is described in detail.

The modeling operations are carefully designed to allow incremental learning by novice users. Users can create a variety of models by learning only the first operation (creation), and can incrementally expand their vocabulary by learning other operations as necessary. We have found it helpful to restrict first-time users to the first three basic operations (creation, painting, and extrusion), and then to introduce other advanced operations after these basic operations are mastered.

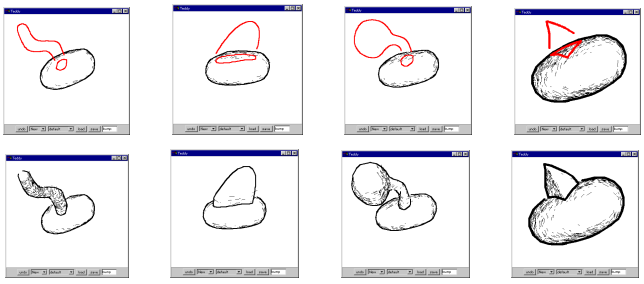
4.1 Overview

Figure 3 introduces Teddy's general model construction process. The user begins by drawing a single freeform stroke on a blank canvas (Figures 3a-b). As soon as the user finishes drawing the stroke, the system automatically constructs a corresponding 3D



a) snake b) snail c) cherry d) muscular arm

Figure 5: Examples of creation operation (top: input stroke, middle: result of creation, bottom: rotated view).



a) long b) thin c) fat d) sharp

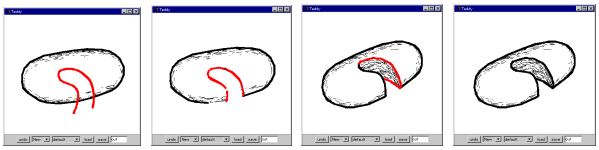
Figure 6: Examples of extrusion (top: extruding stroke, bottom: result of extrusion).

a) digging stroke b) result c) rotated d) closed stroke e) after click

Figure 7: More extrusion operations: digging a cavity (a-c) and turning the closed stroke into a surface drawing (d-e).

shape (c). The user can now view the model from a different direction (d). Once a model is created, it may be modified using various operations. The user can draw a line on the surface (e-g) by drawing a stroke within the model silhouette. If the stroke is closed, the resulting surface line turns red and the system enters "extrusion mode" (h-i). Then the user rotates the model (j) and draws the second stroke specifying the silhouette of the extruded surface (k-m). A stroke that crosses the silhouette cuts the model (n-o) and turns the cut section red (p). The user either clicks to complete the operation (q) or draws a silhouette to extrude the section (r-t). Scribbling on the surface erases the line segments on the surface (u-w). If the user scribbles during the extrusion mode (x-y), the system smoothes the area surrounded by the closed red line (z-z´).

Figure 4 summarizes the modeling operations available on the current implementation. Note that the appropriate action is chosen based on the stroke's position and shape, as well as the current



a) biting stroke b) result c) rotated view d) after click

Figure 8: Cutting operation.

a) cutting stroke b) result c) rotated d) extruding stroke e) result

Figure 9: Extrusion after cutting.

mode of the system.

4.2 Creating a New Object

Starting with a blank canvas, the user creates a new object by drawing its silhouette as a closed freeform stroke. The system automatically constructs a 3D shape based on the 2D silhouette. Figure 5 shows examples of input strokes and the corresponding 3D models. The start point and end point of the stroke are automatically connected, and the operation fails if the stroke is self-intersecting. The algorithm to calculate the 3D shape is described in detail in section 5. Briefly, the system inflates the closed region in both directions with the amount depending on the width of the region: that is, wide areas become fat, and narrow areas become thin. Our experience so far shows that this algorithm generates a reasonable-looking freeform shape. In addition to the creation operation, the user can begin model construction by loading a simple primitive. The current implementation provides a cube and a sphere, but adding more shapes is straightforward.

4.3 Painting and Erasing on the Surface

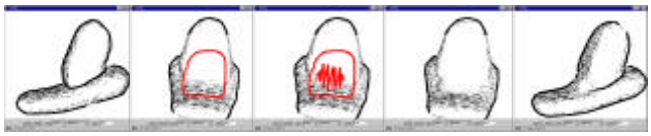
The object surface is painted by drawing a freeform stroke within the object's silhouette on the canvas (the stroke must not cross the silhouette) [11]. The 2D stroke is projected onto the object surface as 3D line segments, called surface lines (Figure 3e-g). The user can erase these surface lines by drawing a scribbling stroke[1] (Figure 3u-w). This painting operation does not modify the 3D geometry of the model, but lets the user express ideas quickly and conveniently when using Teddy as a communication medium or design tool.

4.4 Extrusion

Extrusion is a two-stroke operation: a closed stroke on the surface and a stroke depicting the silhouette of the extruded surface. When the user draws a closed stroke on the object surface, the system highlights the corresponding surface line in red, indicating the initiation of "extrusion mode" (Figure 3i). The user then rotates the model to bring the red surface line sideways (Figure 3j) and draws a silhouette line to extrude the surface (Figure 3k). This is basically a sweep operation that constructs the 3D shape by moving the closed surface line along the skeleton of the silhouette

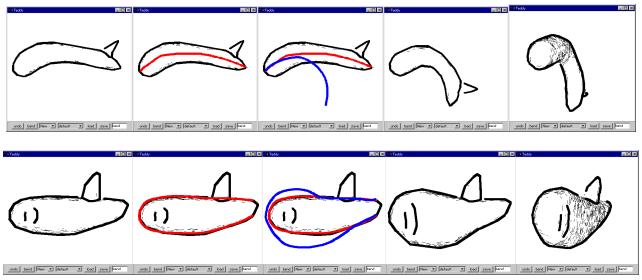
[1] A stroke is recognized as scribbling when `sl/pl > 1.5`, where `sl` is the length of the stroke and `pl` is the perimeter of its convex hull.

a) cleaning a cavity



b) smoothing a sharp edge

Figure 10: Smoothing operation.



a) original b) reference stroke c) target stroke d) result e) rotated

Figure 11: Examples of transformation (top: bending, bottom: distortion).

(Figure 3l-m). The direction of extrusion is always perpendicular to the object surface, not parallel to the screen. Users can create a wide variety of shapes using this operation, as shown in Figure 6. They can also make a cavity on the surface by drawing an inward silhouette (Figure 7a-c). The current implementation does not support holes that completely extend to the other side of the object. If the user decides not to extrude, a single click turns the red stroke into an ordinary painted stroke (Figure 7d-e).

4.5 Cutting

A cutting operation starts when the user draws a stroke that runs across the object, starting and terminating outside its silhouette (Figure 3o). The stroke divides the object into two pieces at the plane defined by the camera position and the stroke. What is on the screen to the left of the stroke is then removed entirely (Figure 3p) (as when a carpenter saws off a piece of wood). The cutting operation finishes with a click of the mouse (Figure 3q). The user can also `bite' the object using the same operation (Figure 8).

The cutting stroke turns the section edges red, indicating that the system is in "extrusion mode". The user can draw a stroke to extrude the section instead of a click (Figure3r-t, Figure 9). This "extrusion after cutting" operation is useful to modify the shape without causing creases at the root of the extrusion.

4.6 Smoothing

One often *smoothes* the surface of clay models to eliminate bumps and creases. Teddy lets the user smooth the surface by drawing a scribble during "extrusion mode." Unlike erasing, this operation modifies the actual geometry: it first removes all the polygons surrounded by the closed red surface line and then creates an

entirely new surface that covers the region smoothly. This operation is useful to remove unwanted bumps and cavities (Figure 3x-z', Figure 10a), or to smooth the creases caused by earlier extrusion operations (Figure 10b).

4.7 Transformation

We are currently experimenting with an additional "transformation" editing operation that distorts the model while preserving the polygonal mesh topology. Although it functions properly, the interface itself is not fully gestural because the modal transition into the bending mode requires a button push.

This operation starts when the user presses the "bend" button and uses two freeform strokes called the *reference stroke* and the *target stroke* to modify the model. The system moves vertices of the polygonal model so that the spatial relation between the original position and the target stroke is identical to the relation between the resulting position and the reference stroke. This movement is parallel to the screen, and the vertices do not move perpendicular to the screen. This operation is described in [5] as *warp*; we do not discuss the algorithm further.

Transformation can be used to bend, elongate, and distort the shape (Figure 11). We plan to make the system infer the reference stroke automatically from the object's structure in order to simplify the operation, in a manner similar to the mark-based interaction technique of [2].

5 ALGORITHM

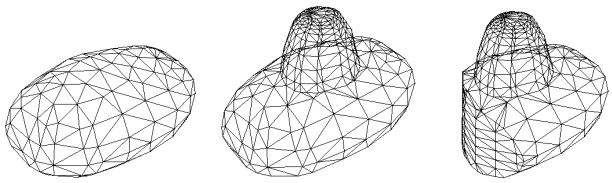
We next describe how the system constructs a 3D polygonal mesh from the user's freeform strokes. Internally, a model is represented as a polygonal mesh. Each editing operation modifies the mesh to conform to the shape specified by the user's input strokes (Figure 12). The resulting model is always topologically equivalent to a sphere. We developed the current implementation as a prototype for designing the interface; the algorithms are subject to further refinement and they fail for some illegal strokes (in that case, the system indicates the problem and requests an alternative stroke). However, these exceptional cases are fairly rare, and the algorithm works well for a wide variety of shapes.

Our algorithms for creation and extrusion are closely related to those for freeform surface construction based on skeletons [3,18], which create a surface around user-defined skeletons using implicit surface techniques. While our current implementation does not use implicit surfaces, they could be used in an alternative implementation.

In order to remove noise in the handwriting input stroke and to construct a regular polygonal mesh, every input stroke is re-sampled to form a smooth polyline with uniform edge length before further processing [4].

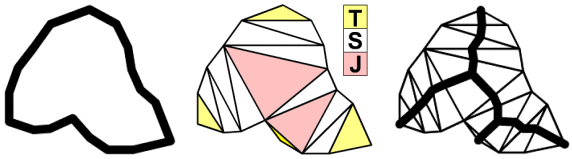
5.1 Creating a New Object

Our algorithm creates a new closed polygonal mesh model from the initial stroke. The overall procedure is this: we first create a closed planar polygon by connecting the start-point and end-point of the stroke, and determine the spine or axes of the polygon using

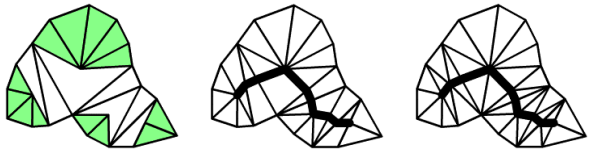


a) after creation b) after extrusion c) after cutting

Figure 12: Internal representation.



a) initial 2D polygon b) result of CDT c) chordal axis



d) fan triangles e) resulting spine f) final triangulation

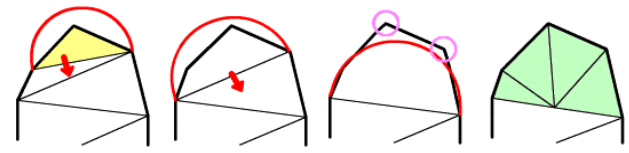
Figure 13: Finding the spine.

the *chordal axis* introduced in [21]. We then elevate the vertices of the spine by an amount proportional to their distance from the polygon. Finally, we construct a polygonal mesh wrapping the spine and the polygon in such a way that sections form ovals.

When constructing the initial closed planar polygon, the system makes all edges a predefined unit length (see Figure 13a). If the polygon is self-intersecting, the algorithm stops and the system requests an alternative stroke. The edges of this initial polygon are called *external edges*, while edges added in the following triangulation are called *internal edges*.

The system then performs constrained Delaunay triangulation of the polygon (Figure 13b). We then divide the triangles into three categories: triangles with two external edges (terminal triangle), triangles with one external edge (sleeve triangle), and triangles without external edges (junction triangle). The chordal axis is obtained by connecting the midpoints of the internal edges (Figure 13c), but our inflation algorithm first requires the *pruning* of insignificant branches and the retriangulation of the mesh. This pruning algorithm is also introduced in [21].

To prune insignificant branches, we examine each terminal triangle in turn, expanding it into progressively larger regions by merging it with adjacent triangles (Figure 14a-b). Let X be a terminal triangle; then X has two exterior edges and one interior edge. We erect a semicircle whose diameter is the interior edge, and which lies on the same side of that edge as does X. If all three vertices of X lie on or within this semicircle, we remove the interior edge and merge X with the triangle that lies on the other side of the edge.



a) start from T-triangle b) advance c) stop d) fan triangles

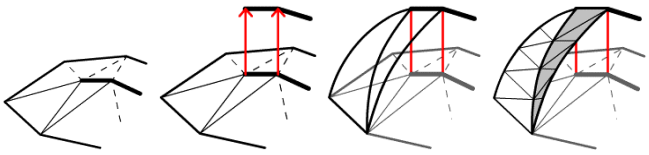
e) advance to J-triangle f) fan triangles at J-triangle

Figure 14: Pruning.

If the newly merged triangle is a sleeve triangle, then X now has three exterior edges and a new interior edge. Again we erect a semicircle on the interior edge and check that all vertices are within it. We continue until some vertex lies outside the semicircle (Figure 14c), or until the newly merged triangle is a junction triangle. In the first case, we triangulate X with a "fan" of triangles radiating from the midpoint of the interior edge (Figure 14d). In the second case, we triangulate with a fan from the midpoint of the junction triangle (Figure 14e-f). The resulting fan triangles are shown in Figure 13d. The pruned spine is obtained by connecting the midpoints of remaining sleeve and junction triangles' internal edges (Figure 13e).

The next step is to subdivide the sleeve triangles and junction triangles to make them ready for elevation. These triangles are divided at the spine and the resulting polygons are triangulated, so that we now have a complete 2D triangular mesh between the spine and the perimeter of the initial polygon (Figure 13f).

Next, each vertex of the spine is elevated proportionally to the average distance between the vertex and the external vertices that are directly connected to the vertex (Figure 15a,b). Each internal edge of each fan triangle, excluding spine edges, is converted to a quarter oval (Figure 15c), and the system constructs an appropriate polygonal mesh by sewing together the neighboring elevated edges, as shown in Figure 15d. The elevated mesh is copied to the other side to make the mesh closed and symmetric. Finally, the system applies mesh refinement algorithms to remove short edges and small triangles [12].



a) before b) elevate spines c) elevate edges d) sew elevated edges

Figure 15: Polygonal mesh construction.

5.2 Painting on the Surface

The system creates surface lines by sequentially projecting each line segment of the input stroke onto the object's surface polygons. For each line segment, the system first calculates a bounded plane consisting of all rays shot from the camera through the segment on the screen. Then the system finds all intersections between the plane and each polygon of the object, and splices the resulting 3D line segments together (Figure 16). The actual implementation searches for the intersections efficiently using polygon connectivity information. If a ray from the camera crosses multiple polygons, only the polygon nearest to the camera position is used. If the resulting 3D segments cannot be spliced together (e.g., if the stroke crosses a "fold" of the object), the algorithm fails.

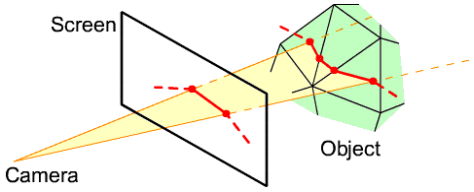
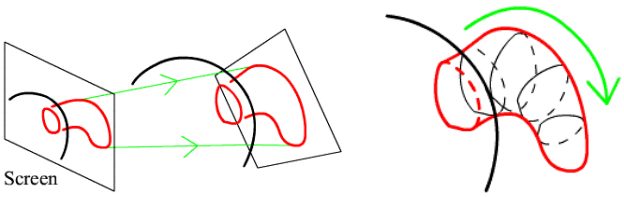


Figure 16: Construction of surface lines.

5.3 Extrusion

The extrusion algorithm creates new polygonal meshes based on a closed base surface line (called the base ring) and an extruding stroke. Briefly, the 2D extruding stroke is projected onto a plane perpendicular to the object surface (Figure 17a), and the base ring is swept along the projected extruding stroke (Figure 17b). The base ring is defined as a closed 3D polyline that lies on the surface of the polygonal mesh, and the normal of the ring is defined as that of the best matching plane of the ring.



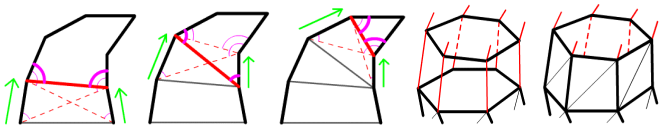
a) projection of the stroke b) sweep along the projected stroke

Figure 17: Extrusion algorithm.

First, the system finds the plane for projection: the plane passing through the base ring's center of gravity and lying parallel to the normal of the base ring[2]. Under the above constraints, the plane faces towards the camera as much as possible (Figure 17a).

Then the algorithm projects the 2D extruding stroke onto the plane, producing a 3D extruding stroke. Copies of the base ring are created along the extruding stroke in such a way as to be almost perpendicular to the direction of the extrusion, and are resized to fit within the stroke. This is done by advancing two pointers (left and right) along the extruding stroke starting from both ends. In each step, the system chooses the best of the following three possibilities: advance the left pointer, the right pointer, or both. The *goodness* value increases when the angle between the line connecting the pointers and the direction of the stroke at each pointer is close to 90 degrees (Figure 18a). This process completes when the two pointers meet.

Finally, the original polygons surrounded by the base ring are deleted, and new polygons are created by sewing the neighboring copies of the base ring together [1] (Figure 18b). The system uses the same algorithm to dig a cavity on the surface.



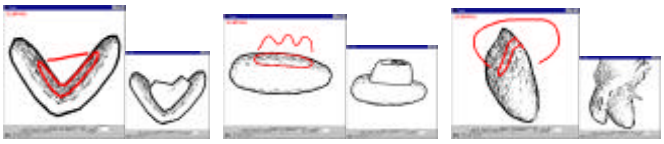
a) pointer advancing b) sewing adjacent rings

Figure 18: Sweeping the base ring.

This simple algorithm works well for a wide variety of extrusions but creates unintuitive shapes when the user draws unexpected extruding strokes or when the base surface is not sufficiently planar (Figure 19).

---

[2] The normal of the ring is calculated as follows: Project the points of the ring to the original XY-plane. Then compute the enclosed "signed area" by the formula:
`Axy = 0.5*sum(i=0, i=n-1, x[i]*y[i+1]-x[i+1]*y[i])`
(indices are wrapped around so that `x[n]` means `x[0]`).
Calculate `Ayx` and `Azx` similarly, and the vector
`v=(Ayz,Azx,Axy)` is defined as the normal of the ring.



a) flat extrusion b) wavy extrusion c) wrapping extrusion

Figure 19: Unintuitive extrusions.

5.4 Cutting

The cutting algorithm is based on the painting algorithm. Each line segment of the cutting stroke is projected onto the front and back facing polygons. The system connects the corresponding end points of the projected edges to construct a planer polygon (Figure 20). This operation is performed for every line segment, and the system constructs the complete section by splicing these planer polygons together. Finally, the system triangulates each planer polygon [22], and removes all polygons to the left of the cutting stroke.

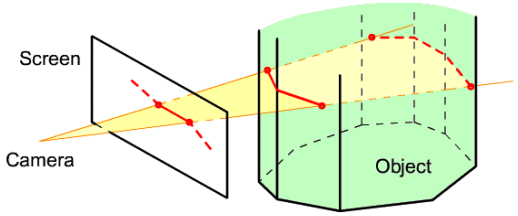


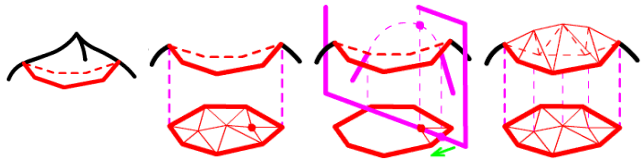
Figure 20: Cutting.

5.5 Smoothing

The smoothing operation deletes the polygons surrounded by the closed surface line (called a ring) and creates new polygons to cover the hole smoothly. First, the system translates the objects into a coordinate system whose Z-axis is parallel to the normal of the ring. Next, the system creates a 2D polygon by projecting the ring onto the XY-plane in the newly created coordinate system, and triangulates the polygon (Figure 21b). (The current implementation fails if the area surrounded by the ring contains creases and is folded when projected on the XY-plane.) The triangulation is designed to create a good triangular mesh based on [22]: it first creates a constrained Delaunay triangulation and gradually refines the mesh by edge splitting and flipping; then each vertex is elevated along the Z-axis to create a smooth 3D surface (Figure 21d).

The algorithm for determining the Z-value of a vertex is as follows: For each edge of the ring, consider a plane that passes through the vertex and the midpoint of the edge and is parallel to the Z-axis. Then calculate the z-value of the vertex so that it lies on the 2D Bezier curve that smoothly interpolates both ends of the ring on the plane (Figure 21c). The final z-value of the vertex is the average of these z-values.

Finally, we apply a surface-fairing algorithm [24] to the newly created polygons to enhance smoothness.



a) before b) triangulation c) calculating Z-value d) result

Figure 21: Smoothing algorithm.

6 IMPLEMENTATION

Our prototype is implemented as a 13,000 line Java program. We tested a display-integrated tablet (Mutoh MVT-14, see Figure 1) and an electric whiteboard (Xerox Liveboard) in addition to a standard mouse. The mesh construction process is completely real-time, but causes a short pause (a few seconds) when the model becomes complicated. Teddy can export models in OBJ file format. Figure 2 shows some 3D models created with Teddy by an expert user and painted using a commercial texture-map editor. Note that these models look quite different from 3D models created in other modeling systems, reflecting the hand-drawn nature of the shape.

7 USER EXPERIENCE

The applet version of Teddy has undergone limited distribution, and has been used (mainly by computer graphics researchers and students) to create different 3D models. Feedback from these users indicates that Teddy is quite intuitive and encourages them to explore various 3D designs. In addition, we have started close observation of how first-time users (mainly graduate students in computer science) learn Teddy. We start with a detailed tutorial and then show some stuffed animals, asking the users to create them using Teddy. Generally, the users begin to create their own models fluently within 10 minutes: five minutes of tutorial and five minutes of guided practice. After that, it takes a few minutes for them to create a stuffed animal such as those in Figure 2 (excluding the texture).

8 FUTURE WORK

Our current algorithms and implementation are robust and efficient enough for experimental use. However, they can fail or generate unintuitive results when the user draws unexpected strokes. We must devise more robust and flexible algorithms to handle a variety of user inputs. In particular, we plan to enhance the extrusion algorithm to allow more detailed control of surfaces. We are also considering using implicit surface construction techniques.

Another important research direction is to develop additional modeling operations to support a wider variety of shapes with arbitrary topology, and to allow more precise control of the shape. Possible operations are creating creases, twisting the model, and specifying the constraints between the separate parts for animation systems [20]. While we are satisfied with the simplicity of the current set of gestural operations, these extended operations will inevitably complicate the interface, and careful interface design will be required.

9 ACKNOWLEDGEMENTS

This project was not possible without the support and encouragement of people. We would especially like to thank Marshall Bern, Lee Markosian, Robert C. Zeleznik, and Hiromasa Suzuki for their invaluable advice in the development of the system, and we thank John F. Hughes, Randy Pausch, Jeffrey S. Pierce, Joe Marks, Jock D. Mackinlay, Andrew Glassner, and Brygg Ullmer for their help in the preparation of this paper. We also thank the SIGGRAPH reviewers for their thoughtful comments. This work is supported in part by JSPS Research Fellowship.

References

1. G. Barequet and M Sharir. Piecewise-linear interpolation between polygonal slices. *ACM 10th Computational Geometry Proceedings*, pages 93-102, 1994.
2. T. Baudel. A mark-based interaction paradigm for free-hand drawing. *UIST'94 Conference Proceedings*, pages 185-192, 1994.
3. J. Bloomenthal and B. Wyvill. Interactive techniques for implicit modeling. *1990 Symposium on Interactive 3D Graphics*, pages 109-116, 1990.
4. J.M. Cohen, L. Markosian, R.C. Zeleznik, J.F. Hughes, and R. Barzel. An Interface for Sketching 3D Curves. *1999 Symposium on Interactive 3D Graphics*, pages 17-21, 1999.
5. W.T. Correa, R.J. Jensen, C.E. Thayer, and A. Finkelstein. Texture mapping for cel animation. *SIGGRAPH 98 Conference Proceedings*, pages 435-456, 1998.
6. M. Deering. The Holosketch VR sketching system. *Communications of the ACM*, 39(5):54-61, May 1996.
7. L. Eggli, C. Hsu, G. Elber, and B. Bruderlin, Inferring 3D models from freehand sketches and constraints. *Computer-Aided Design*, 29(2): 101-112, Feb.1997.
8. C.Grimm, D. Pugmire, M. Bloomental, J. F. Hughes, and E. Cohen. Visual interfaces for solids modeling. *UIST '95 Conference Proceedings*, pages 51-60, 1995.
9. T. Galyean and J.F. Hughes. Sculpting: an interactive volumetric modeling technique. *SIGGRAPH '91 Conference Proceedings*, pages 267-274, 1991.
10. M.D. Gross and E.Y.L. Do. Ambiguous intentions: A paper-like interface for creative design. *UIST'96 Conference Proceedings*, pages 183-192, 1996.
11. P. Hanrahan, P. Haeberli, Direct WYSIWYG Painting and Texturing on 3D Shapes, *SIGGRAPH 90 Conference Proceedings*, pages 215-224, 1990.
12. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. *SIGGRAPH 93 Conference Proceedings*, pages 19-26, 1993.
13. J. Hultquist. A virtual trackball. *Graphics Gems* (ed. A. Glassner). Academic Press, pages 462-463, 1990.
14. J.A. Landay and B.A. Myers. Interactive sketching for the early stages of user interface design. *CHI'95 Conference Proceedings*, pages 43-50, 1995.
15. R. MacCracken and K.I. Joy. Free-form deformations with lattices of arbitrary topology. *SIGGRAPH 96 Conference Proceedings*, pages 181-188, 1996.
16. L. Markosian, M.A. Kowalski, S.J. Trychin, L.D. Bourdev, D. Goldstein, and J.F. Hughes. Real-time nonphotorealistic rendering. *SIGGRAPH 97 Conference Proceedings*, pages 415-420, 1997.
17. H. Nishimura, M. Hirai, T. Kawai, T. Kawata, I. Shirakawa, K. Omura. Object modeling by distribution function and a method of image generation. Transactions of the Institute of Electronics and Communication Engineers of Japan, J68-D(4):718-725, 1985
18. L. Markosian, J.M. Cohen, T. Crulli and J.F. Hughes. Skin: A Constructive Approach to Modeling Free-form Shapes. *SIGGRAPH 99*, to appear, 1999.
19. K. van Overveld and B. Wyvill. Polygon inflation for animated models: a method for the extrusion of arbitrary polygon meshes. *Journal of Visualization and Computer Animation*, 18: 3-16, 1997.
20. R. Pausch, T. Burnette, A.C. Capeheart, M. Conway, D. Cosgrove, R. DeLine, J. Durbin, R. Gossweiler, S. Koga, and J. White. Alice: Rapid prototyping system for virtual reality. *IEEE Computer Graphics and Applications*, 15(3): 8-11, May 1995.
21. L. Prasad. Morphological analysis of shapes. *CNLS Newsletter*, 139: 1-18, July 1997.
22. J.R. Shewchuk. Triangle: engineering a 2D quality mesh generator and Delauny triangulator. *First Workshop on Applied Computational Geometry Proceedings*, pages 124-133, 1996.
23. K. Singh and E. Fiume. Wires: a geometric deformation technique. *SIGGRAPH 98 Conference Proceedings*, pages 405-414, 1998.
24. G. Taubin. A signal processing approach to fair surface design. *SIGGRAPH 95 Conference Proceedings*, pages 351-358, 1995.
25. S.W. Wang and A.E. Kaufman, Volume sculpting. *1995 Symposium on Interactive 3D Graphics*, pages 109-116, 1995.
26. W. Welch and A. Witkin. Free-form shape design using triangulated surfaces. *SIGGRAPH 94 Conference Proceedings*, pages 247-256, 1994.
27. L. Williams. Shading in Two Dimensions. Graphics Interface '91, pages 143-151, 1991.
28. L. Williams. 3D Paint. *1990 Symposium on Interactive 3D Graphics*, pages 225-233, 1990.
29. R.C. Zeleznik, K.P. Herndon, and J.F. Hughes. SKETCH: An interface for sketching 3D scenes. *SIGGRAPH 96 Conference Proceedings*, pages 163-170, 1996.