3D Free-Form Modeling with Variational Surfaces

Alvaro Cuno, Claudio Esperança, Paulo Roma Cavalcanti, Ricardo Farias
Universidade Federal do Rio de Janeiro
Programa de Engenharia de Sistemas e Computação/COPPE
Rio de Janeiro, Brazil
{alvaro, esperanc, roma, rfarias}@lcg.ufrj.br

ABSTRACT

We describe a free-form stroke-based modeling system where objects are primarily represented by means of variational surfaces. Although similar systems have been described in recent years, our approach achieves both a good performance and reduced surface leak problems by employing a coarse mesh as support for constraint points. The prototype implements an adequate set of modeling operations, “undo” and “redo” facilities and a clean interface capable of resolving ambiguities by means of suggestion thumbnails.

Keywords
Free-form modeling, stroke based modeling, RBFs.

1. INTRODUCTION

Typical 3D modeling systems are mostly designed to handle the creation of technical models, i.e., objects with precise measures or which must obey well-defined geometric rules. Such systems are not well-suited to handle so-called *free-form* models, which can be regarded as 3D models akin to 2D free-hand sketches. One reason for this is the fact that interaction in 3D relies almost exclusively on 2D projections, since the only feasible alternative for effectively working in 3D space is by employing costly and cumbersome virtual reality gear. Thus, the user must ultimately manipulate 2D features in order to accomplish 3D editing tasks.

Perhaps the most salient features of any given 3D model are its edges and silhouette lines. Igarashi et al. [Iga99] used this observation to build a prototype 3D free-form modeler called *Teddy*. In contrast with common 3D modelers, Teddy is easy and intuitive enough to be used even by small children. It relies on a scheme by which free-hand drawing strokes representing silhouette lines are used to build and modify smooth closed surfaces. Teddy also innovates over other 3D modelers by not using the standard *WIMP* (Windows, Icons, Menus and Pointers) interface paradigm. Rather, all interaction is based upon stroke recognition and a very small number of command buttons.

Another key aspect that must be addressed in the construction of stroke-based interfaces is the resolution of ambiguities that may arise during a modeling session. For instance, a new stroke drawn by the user may be interpreted either as the cue for creating a new shell or as the profile of an extrusion operation. Our system copes with this problem by using a *suggestive interface* similar to the approach described in [Iga01]. Namely, thumbnail images representing the alternative results are displayed in a corner of the main display window, which must then be clicked by the user in order to select the desired outcome.

The remainder of this paper is organized as follows. Section 2 presents some relevant work related to the problem at hand. An overall description of the proposed system is presented in Section 3 and some concepts of the variational surfaces are introduced in Section 4. The involved algorithms are described in detail in Section 5. Some key aspects of the implementation are discussed in Section 6 and some results and limitations are presented in Section 7. Finally, some concluding remarks and suggestions for future work can be found in Section 8.

2. RELATED WORK

In the last few years, several experimental systems have been proposed which offer interfaces for the specification and construction of different types of three-dimensional scenes starting from 2D strokes [Zel96, Tol99, Mar99, Coh99, Coh00, Tol01, Iga01, Tai04]. Specially worthy of note is the *Teddy* system

The Journal of WSCG, Vol.13, ISSN 1213-6964
WSCG’2005, January 31-February 4, 2005
Plzen, Czech Republic.
Copyright UNION Agency – Science Press

proposed by Igarashi et al. [Iga99], which can be used to create simple models with spherical topology with only a few strokes. An initial model is created by drawing a simple closed curve which is then inflated resulting in a blob-like object such that the curve approximates its silhouette. Additional strokes can then be used to extrude protrusions, cut, bend or smooth the model.

Modeling operations in *Teddy* are performed on a polygonal mesh representation of the surface. Some of these operations necessarily require the subsequent use of smoothing algorithms on the edited mesh. Nevertheless, some models end up with undesirable protuberances and wrinkles due to triangles with awkward characteristics. Besides, *Teddy* does not support the creation of multiple objects in the same scene and therefore operations to combine these are unavailable.

Karpenko et al. [Kar02] deal with the problem of undesired surface roughness by using *Variational Surfaces* as the main representation scheme. These surfaces are zero-sets of a class of implicit functions known as *RBF-based implicits*. The term *RBF* -- or Radial Basis Functions -- refers to the fact that the basis functions used in the creation of the implicit are radially symmetric. The key advantage of variational surfaces lies in that they are naturally smooth, since their construction can be regarded as an energy minimization process. This, however, leads to other problems. For instance, models with creases and tips cannot be easily created. Also, the performance of the system is heavily dependent on the number of constraint points used in defining the implicit. This is worsened by the fact that model editing operations are performed using a great number of mesh vertices produced by the visualization process.

Owada et al. [Owa03] present a system that generates volumetric models from 2D strokes. Besides making it possible to create, cut and extrude surface features, their approach also allows the specification of internal structures in the models with arbitrary topology. The main disadvantage of that system is that simple smooth surfaces can be modeled only with high storage and computation costs.

Blobmaker [Ara03] is prototype system quite similar to the one presented by Karpenko et al. Its main contribution lies in the use of skeletons for model construction. This allows the creation of objects with arbitrary topology and an efficient application of edition operations. However, the use of constraint points positioned irregularly on the surface may lead to surface leaks after a few modeling steps.

Recently, Tai et al. [Tai04] described a system based on convolution surfaces for the construction of free-form models starting from a silhouette curve. The resulting shape has circular cross-section, but can be conveniently modified through a sketched profile or shape parameters. But, unlike the prototypes discussed above, their system employs menus and sliders in its modeling interface.

3. SYSTEM DESCRIPTION

The prototype system allows the user to quickly create simple 3D models by drawing 2D strokes directly on the system window. Once the model is created, it can be further edited with operations such as merging, extrusion and piercing, which are also specified by inputting additional 2D strokes. Thus, the execution of an operation depends solely on the stroke form and where it was made, making it unnecessary to press any button or select menu options.

The user interface is composed of a design window and five command buttons. The ***init*** button starts a new modeling session, ***save*** saves the polygonal mesh of the modeled object, ***undo*** cancels the effects of the last operation, ***redo*** cancels the most recent ***undo*** command, and the ***quit*** button exits the system. Operations ***undo/redo*** work on a linear history of editing operations starting at the most recent invocation of the ***init*** command. This mechanism enables the user to review all operations made during a modeling session.

Input strokes are drawn by dragging the mouse with the left button pressed. A model can be moved on the *xy* plane by positioning the mouse over the model and then dragging it with the right button pressed. Translation along the *z* axis is accomplished in a similar way, but the middle button is used instead. Rotation uses an arc-ball interaction style: first, the center of rotation is specified by clicking on the model with the right button, the rotation angle and direction is then input by dragging the mouse with the right button.

Operations

A modeling session begins with an empty design window. The user specifies the model silhouette to be constructed by drawing a simple closed curve with a single stroke. The system then constructs a plausible 3D model based on the input silhouette. This is accomplished by inflating the curve in both directions by an amount proportional to its width, this is, narrow areas will become thin regions while wide areas generate fat regions [Iga99]. Figure 1 shows examples of input strokes and the corresponding 3D models constructed by the system.

Object creation operations may be performed many times, thus allowing the construction of scenes with multiple objects (see Figure 1(d)).

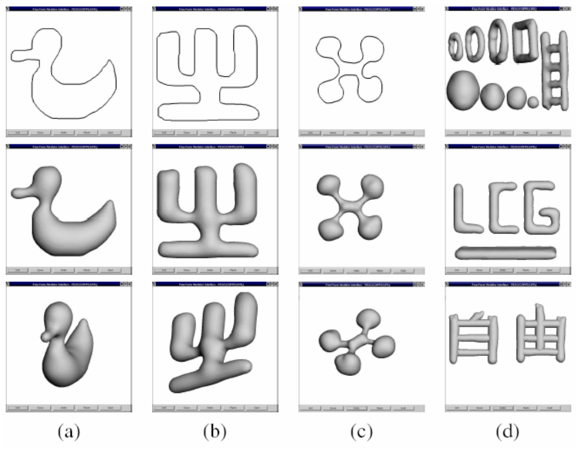

Figure 1. (a), (b) and (c) Object creation examples. (d) Scenes with multiple objects.

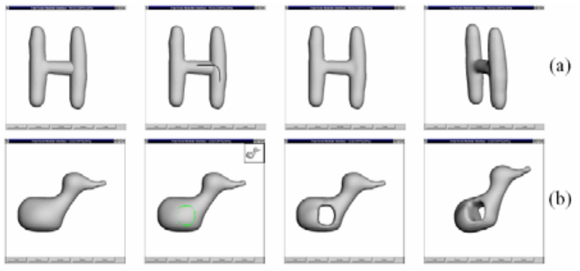

Figure 2. (a) Model merging. (b) Model piercing.

Model ***merging*** creates a new surface that approximates two previously existing models which are then discarded. The effect is to obtain a single implicit representation that smoothly blends two given shapes. The user commands this operation by drawing a simple open stroke starting inside the first input model and ending inside the second. The two input models must overlap in space for this operation to take place. Figure 2(a) shows an example.

The ***piercing*** operation can be used to make a hole in a model. The user must first draw a closed curve lying entirely inside the silhouette of the target model. This stroke can be interpreted in two ways by the system: either as a cue for performing a piercing operation or as an auxiliary element for performing an extrusion. At this point, the system will signal the ambiguity by displaying in the upper-left corner of the window a thumbnail image showing the result of the piercing operation. The user must click on this image in order to accept the operation (see Figure 2(b)). Any other action will trigger the other interpretation.

Extrusion is a modeling operation which allows the creation of a new protrusion on some part of a model. The extruded feature is described by a profile curve which is input as a simple open curve starting and ending inside the model's silhouette but extending beyond it. The area on which the protrusion will be "glued" can be defined either *explicitly* or *implicitly*. In the former case the gluing area is delimited by a closed curve drawn previously --see the preceding paragraph. In the latter case, the gluing area will correspond to a roughly circular region touching the two endpoints of the profile curve. Figure 3 illustrates this operation.

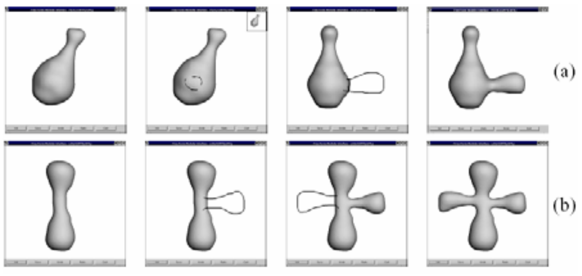

Figure 3. Extrusion examples: (a) using a base curve, and (b) automatic extrusion.

4. VARIATIONAL SURFACES

Although a through discussion of the math of implicit object modeling is outside the scope of this paper, for the sake of completeness, we try to lay down a few key concepts below. The interested reader is referred to the excellent introduction to the subject in [Tur99a].

The term "Variational Surface" refers to the zero-set of a RBF-based implicit function. Such functions are used in the context of scattered data interpolation. This is a problem where, given a set of *n* distinct constraint points $\{c_1, c_2, \ldots, c_n\}, c \in \Re^3$ and a set of *n* function values $\{v_1, v_2, \ldots, v_n\}$, it is sought a smooth function $f : \Re^3 \rightarrow \Re$ such that $f(c_i) = v_i$, for $i = 1 \ldots n$. The smoothness criteria usually involve some "deformation" energy that must be minimized. This entails the solution of a linear system with *n* equations. Solving this system is perhaps the most computationally intensive part of the system. We use a standard LU-decomposition algorithm for this task.

A variational surface can be modeled simply by choosing an adequate set of constraint points and associated values. The most used approach requires the placement of *n*/2 points with value equal to zero --these are known as *boundary constraint points*. Another set of *n*/2 points are obtained by displacing each boundary point by a small amount along the direction of the estimated surface normal at that point. These points, known as *normal constraint points*, are associated with a small positive constant *w* (see Figure 4).

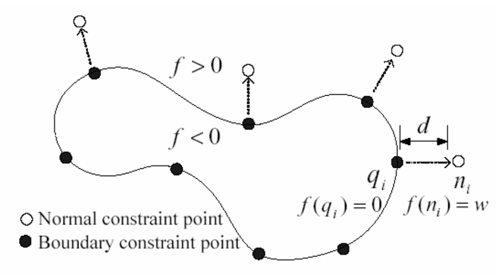

Figure 4. The normal constraint points n_i are placed along the estimated normal vector at a distance d from boundary constraint points q_i. The function f is such that $f(x) < 0$ for x inside the curve and $f(x) > 0$ outside the curve.

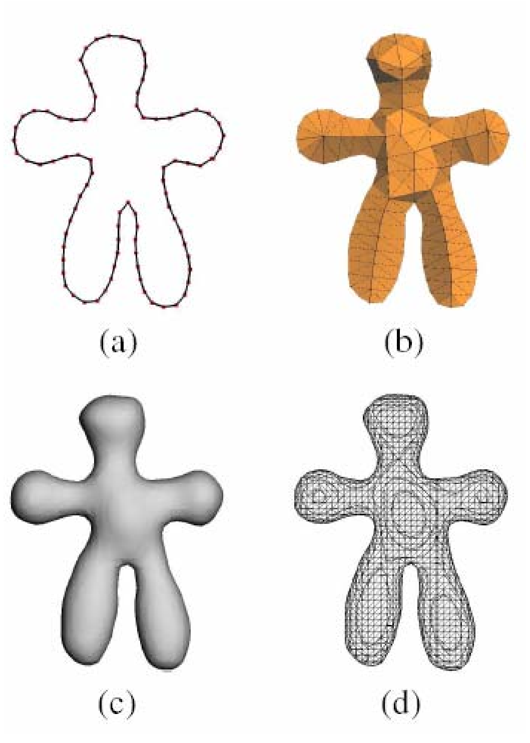

Figure 5. (a) 2D input stroke. (b) Coarse polygonal mesh of support for surface specification (177 vertices and 350 triangles). (c) Visualization of implicit surface $f = 0$, using smooth shading and (d) the triangular mesh (3620 vertices and 7236 triangles)

Any standard method for visualizing implicit objects can be used to render the modeled surface. In most cases, a polygonization scheme is employed and the resulting set of polygons is rendered using standard graphics hardware. It should be noted, however, that the polygonization scheme should be carefully chosen in order to minimize the number of function evaluations, since these are costly operations. We use a hierarchical variant of the Marching Cubes algorithm [Lor87].

5. ALGORITHMS

Creation

The creation algorithm consists essentially in specifying an adequate set of constraint points based on the user's input silhouette curve. The constraint points are chosen to coincide with the vertices of a coarse mesh built from the input stroke using an inflation algorithm. Figure 5 illustrates a global idea of the algorithm.

The construction of the coarse mesh follows the method described in [Iga99]. We found that this approach yields more pleasing results than the simpler algorithm adopted in [Kar02].

Merging

The merging operation consists in creating a new variational surface whose shape approximates the union of two other given surfaces. The algorithm consists of eliminating constraint points which are contained in the intersection of the two input shapes. Let us call h the resulting function and f and g the two input functions. Then, h contains a boundary constraint point x of f only if $g(x) > 0$. Similarly, h contains a boundary constraint point y of g only if $f(y) > 0$. Additionally, if a boundary constraint point is eliminated in this process, then the corresponding normal constraint point is also discarded. Figure 6 illustrates the idea.

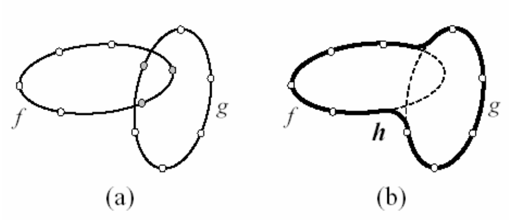

Figure 6. Merging illustration in 2D. (a) Constraint points positioned inside the intersection of the models represented for f and g are eliminated. (b) The new model represented by function h is built with points that remained after the elimination process.

Piercing

Let f be the function representing the model to be edited, C the 2D closed curve drawn by the user (represented by a simple polygon), and h the resulting model from this operation. Then, the piercing algorithm comprises the following steps:

1. Project each vertex C_i of C on the front-facing triangles of the polygonized model surface. Let F_i be the corresponding projected point. If the projection of any C_i yields more than one projected point, the piercing algorithm is aborted.

2. Similarly, project the vertices of C on the back-facing triangles of the polygonized model surface and call B_i the resulting projected vertices. As before, abort the algorithm if the more than one projection point is found for any given vertex.
3. Interpolate k evenly spaced points along each line segment F_iB_i. Let us call such points M_j. In our implementation, k = 3, i.e., three points are generated between each pair of vertices F_i and B_i.
4. Create an interpolating function g, which will be built by the creation procedure, but using F_i, B_i and M_j as boundary constraint points. The surface orientation is defined by placing an additional constraint point p placed at the approximate center of the shape and mapped to a negative value (-1 in our implementation). The position of p is estimated by computing the coordinate-wise average of all boundary constraint points. If this point does not lie inside curve C, then the piercing algorithm is aborted.
5. Perform the merging operation on f and g.

Explicit Extrusion

This type of extrusion is defined by two strokes: a base curve drawn directly on the model surface which defines the model area affected by the edition process, and a profile stroke. If f is the input model function, then the explicit extrusion is computed as follows:

1. Project the base curve on the polygonized object using the same rationale described in item 1 of the previous Sub-section. Let us use C to refer to this projected curve.
2. Project the profile curve on the plane that passes through the base curve's barycenter and is parallel to the viewing plane. Let us call P the resulting curve.
3. Create an interpolating function g using the vertices of C and P as boundary constraint points. Additionally, estimate normal constraint points by displacing the vertices of P outward.
4. Apply the merging operation to f and g.

Implicit Extrusion

This operation requires only an extrusion profile [Kar02]. The procedure is the following:

1. Select the silhouette vertices of the model's polygonized mesh vertices. A silhouette vertex is any vertex incident on two triangles whose normals point to opposite sides of the viewer plane. Find S and E, the silhouette vertices which are closest to the initial and end points of the profile curve, respectively.
2. Project the extrusion profile curve on the plane that passes by the middle point of line segment SE and is parallel to the viewing plane.
3. Create an interpolating function g using the vertices of the projected curve computed in the previous step as boundary constraint points. For each of these, add a normal constraint point by displacing it outward with respected to the curve.
4. Apply the merging operation between f and g.

6. IMPLEMENTATION DETAILS

The prototype system was written in the C++ language and the *OpenGL* library was used to render the polygonized models. All example models shown in this paper were built by the prototype system in a PC equipped with a 1.3 GHz AMD-Duron processor and 256 MB of main memory.

The system uses two main data structures: a scene representation and a command list. The scene is the model repository and the command list records the history of a modeling session (Figure 7).

Every time a new modeling operation is issued by the user, a corresponding command is inserted at the end of the command list. Depending on the command type, its execution can insert and/or remove models from the scene. For instance, a command “merge” will insert a new model in the scene, and will remove the input models.

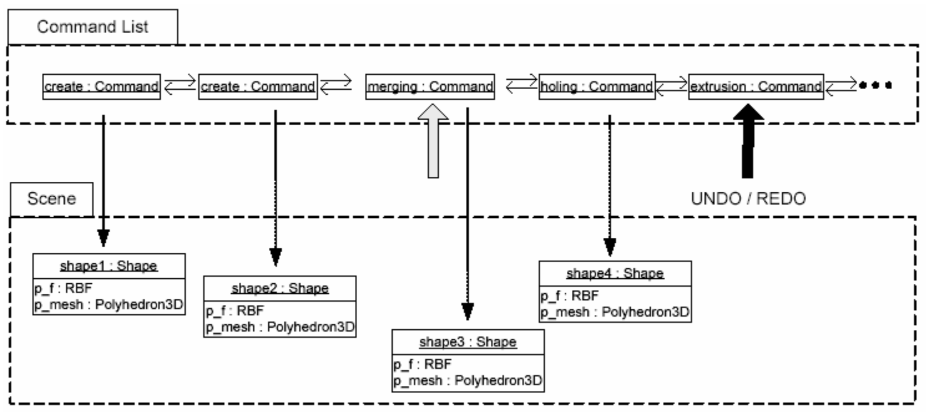

Figure 7. Main data structures of the system.

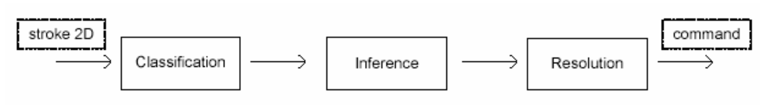

Figure 8. Stages for the command determination to execute starting from a 2D stroke.

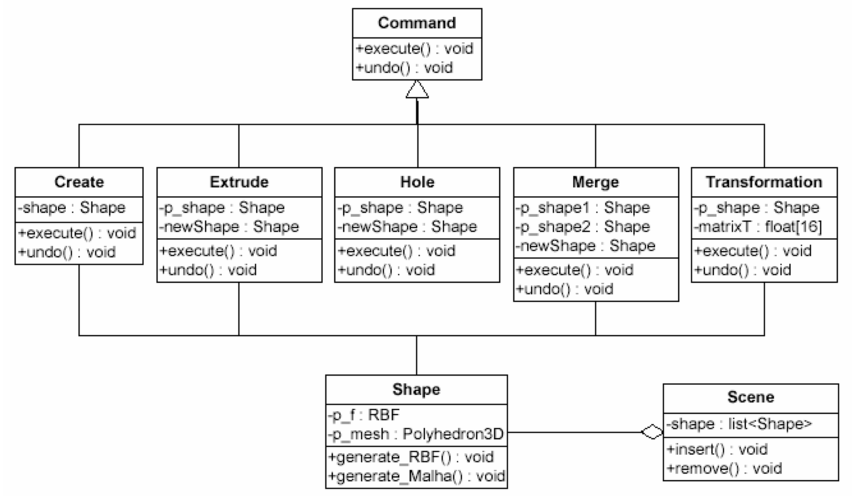

Figure 9. System class hierarchy.

Thus, the *Undo*/*Redo* mechanism works by scanning the command list in both directions replaying or undoing the commands appropriately.

The system determines the command type to be executed in response to the input 2D strokes using the following three-step approach (see Figure 8):

1. The classification stage determines the stroke type, i.e., simple or non-simple, closed or open.
2. Depending on the place where the stroke was drawn and on its type, the inference stage creates the appropriate command.
3. If an ambiguity is detected, the user is prompted to choose the desired outcome. The resolution stage then inserts the command in the list and executes it.

This approach is based on the ambiguities resolution proposal of Alvarado et al. [Alv2001].

A brief description of the system class hierarchy is presented in Figure 9. The class attributes labeled *p_shape* are pointers to models, while the absence of the prefix *p_* means a reference to the model itself.

Superclass **Command** is an abstract class with two methods: *execute()* and *undo()*. Method *execute()* executes the suitable actions for a command, while method *undo()* undoes the actions done by method *execute()*. For instance, in an extrusion operation, *undo()* removes the resulting model from the extrusion operation shape, and inserts the unextruded model *p_shape* again.

Create is a class that implements the model creation process. **Extrude** modifies the model pointed to by *p_shape* generating a resulting model *NewShape*.

The same happens with class **Pierce**. **Merge** produces a new shape *NewShape* starting from models *p_shape1* and *p_shape2*. Rigid motions (rotations and translations) are implemented in class **Transformation**.

Class **Shape** stores object geometry using two representations: *f*, an analytical representation of a RBF-based implicit function, and a triangle mesh generated by applying a polygonization algorithm on *f*.

Finally, class **Scene** contains a (possibly empty) model list that is manipulated by methods *insert()* and *remove()*.

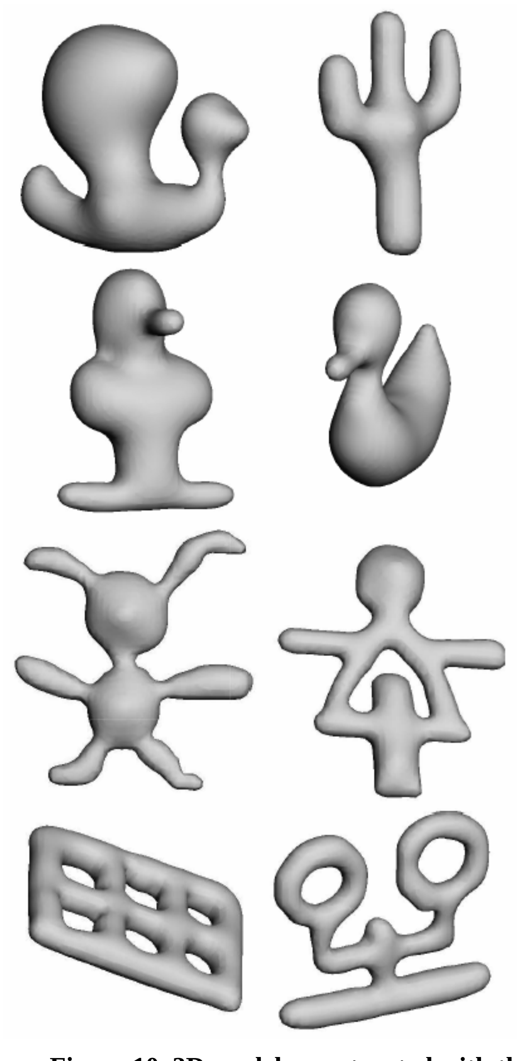

Figure 10. 3D models constructed with the prototype system.

7. RESULTS AND LIMITATIONS

Figure 10 shows some models built with our prototype system. They are smooth surfaces of arbitrary topology and exhibit a loose "look" which is characteristic of free-hand 2D drawings. The interested reader may access http://www.lcg.ufrj.br/Projetos/ffmodelling and download some of these models in OFF format [Ros89].

Due to the nature of the radial basis functions used in the underlying representation of our system, models with creases or sharp features cannot be created. Also, sometimes the result of a modeling operation is unintuitive. This is the case, for instance, when two small objects containing relatively few constraint points are merged (see Figure 11).

Another current limitation of the system lies in the fact that the piercing operation cannot be applied other than on relatively simple local geometries. For instance, if the intended hole would pierce the surface more than once, then the operation fails (see Figure 12(a)). In some other cases, the hole fails to properly pierce the model (Figure 12(b)).

Some modeling operations may incur in a problem known as ***surface leak***. This is due to the constraint points being distributed irregularly. Figure 13 illustrates this problem. We deal with this problem by using a coarse polygonal mesh introduced in the model creation process (see the Creation sub-section).

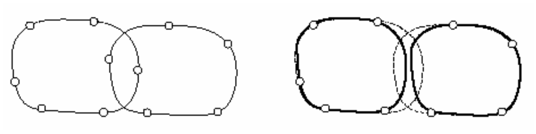

Figure 11. Merging two small models may yield unintuitive results.

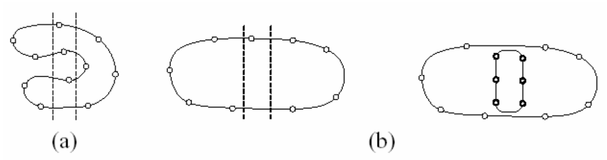

Figure 12. Limitations of the piercing operation. (a) Piercing fails in complicated situations. (b) Hole may fail to pierce the surface completely.

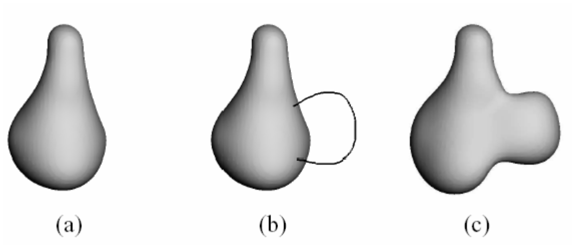

Figure 13. Surface leak problem. (a) Initial interpolation. (b) Extrusion. (c) Leak after extrusion.

8. CONCLUSIONS AND FUTURE WORK

Free-form modeling supported by RBF-based implicits enjoys quite a few advantages over traditional approaches employing parametric surfaces. In particular, the generated models are naturally smooth and many intuitive modeling operations can be implemented with relative ease. Its foremost limitations can be attributed to the time complexity of the scattered point interpolation scheme used.

Therefore, a natural extension of the present work consists of adopting more eficient interpolation schemes such as the FastRBF [Car01], which will enable our system to handle models with increased complexity. This, in turn, will help us cope with the surface leaking problems. Another venue that should be explored is the adoption of a more careful sampling strategy for the constraint points.

The set of modeling operations available in our systems is somewhat limited still. We are working on an enhanced algorithm for the piercing operation, as well as other operations such as cutting and bending. Regarding the visualization process, we are experimenting with a novel polygonization approach with some promising results (access http://www.lcg.ufrj.br/Projetos/ffmodelling for details).

9. ACKNOWLEDGMENTS

We are grateful to CNPq for providing financial support for the first author.

10. REFERENCES

[Alv01] Alvarado, C. and Davis, R. Resolving ambiguities to create a natural computer based sketching environment. In *Proceedings of IJCAI-*2001, pages 1365-1371.

[Ara03] Araujo, B. and Jorge, J. Blobmaker: Free-form modeling with variational implicit surfaces. In 12o *Encontro Portugues de Computação Grafica* (EPCG) - 2003.

[Car01] Carr, J. C., Beatson, R. K., Cherrie, J. B., Mitchell, T. J., Fright, W. R., McCallum, B. C., and Evans, T. R. Reconstruction and representation of 3D objects with radial basis functions. In *Proceedings of SIGGRAPH 2001*, pages 67-76. ACM Press. 2001.

[Coh00] Cohen, J. M., Hughes, F., and Zeleznik, R. C. Harold: A world made of drawings. In *Proceedings of NPAR 2000*, pages 83-90. ACM.

[Coh99] Cohen, J. M., Markosian, L., Zeleznik, R. C., Hughes, J. F., and Barzel, R. An interface for sketching 3D curves. In *Proceedings of Symposium on Interactive 3D Graphics*, pages 17-21. ACM Press.

[Din02] Dinh, H. Q., Turk, G., and Slabaugh, G. Reconstructing surfaces by volumetric regularization using radial basis functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(10):1358-1371, 2002.

[Iga01] Igarashi, T. and Hughes, J. A suggestive interface for 3D drawing. In *Proceedings of ACM UIST'01*, pages 173-181, 2001. ACM Press.

[Iga99] Igarashi, T., Matsuoka, S., and Tanaka, H. Teddy: A sketching interface for 3D freeform design. In *Proceedings of SIGGRAPH 99*, pages 409-416. ACM Press.

[Kar02] Karpenko, O., Hughes, J. F., and Raskar, R. Free-form sketching with variational implicit surfaces. *Computer Graphics Forum*, 2002.

[Lor87] Lorensen, W. and Cline, H. Marching cubes: a high resolution 3D surface construction algorithm. *Computer Graphics*, 21(4):163-169, 1987.

[Mar99] Markosian, L., Cohen, J. M., Crulli, T., and Hughes, J. Skin: a constructive approach to modeling free-form shapes. In Proceedings of SIGGRAPH 99, Annual Conference Series, pages 393-400. ACM Press, 1999.

[Owa03] Owada, S., Nielsen, F., Nakazawa, K., and Igarashi, T. A sketching interface for modeling the internal structures of 3D shapes. In *Proceedings of 3rd International Symposium on Smart Graphics*, pages 49-57. Springer, 2003.

[Ros89] Rost, R. J. (1989). A 3D object file format.
http://www.dcs.ed.ac.uk/home/mxr/gfx/3d/off.spec.

[Tai04] Tai, C., Zhang, H., and Fong, C. Prototype modeling from sketched silhouettes based on convolution surfaces. *Computer Graphics Forum*, 23(1):71-83, 2004.

[Tol99] Tolba, O., Dorsey, J., and McMillan, L. Sketching with projective 2D strokes. In *Proceedings of the 12th annual ACM symposium on UIST*, pages 149-157, 1999.

[Tol01] Tolba, O., Dorsey, J., and McMillan, L. A projective drawing system. In *Proceedings of 2001 ACM Symposium on Interactive 3D Graphics*, pages 25-34, 2001.

[Tur99a] Turk, G. and O'Brien, J. Shape transformation using variational implicit functions. In *Proceedings of SIGGRAPH 99*, pages 335-342, 1999.

[Tur99b] Turk, G. and O'Brien, J. Variational implicit surfaces. Technical Report, Georgia Institute of Technology, 1999.

[Tur02] Turk, G. and O'Brien, J. Modelling with implicit surfaces that interpolate. ACM *Transactions on Graphics*, pages 855-873, 2002.

[Zel96] Zeleznik, R. C., Herndon, K. P., and Hughes, J. F. SKETCH: An interface for sketching 3D scenes. In *SIGGRAPH 96 Conference Proceedings*, pages 163-170, 1996