EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling (2008)
C. Alvarado and M.- P. Cani (Editors)

A Sketch-Based Method to Control Deformation in a Skeletal Implicit Surface Modeler

Masamichi Sugihara [1], Erwin de Groot[2], Brian Wyvill[1] and Ryan Schmidt[3]

[1]University of Victoria, Canada
[2]University of Calgary, Canada [3] University of Toronto, Canada

Abstract

Skeletal implicit surfaces offer many advantages for sketch-based modeling systems, such as blending, CSG, and a procedural object hierarchy. Free-form deformation (FFD) is also extremely useful in this context, however existing FFD approaches do not support implicit surface representations, and FFD lattice manipulation is time-consuming compared to sketch-based techniques. In this paper, we describe an FFD technique suitable for implicit surface representations. To enhance real-time feedback, we split the problem into an approximate formulation used during interactive deformation, and a more robust variational technique which preserves desirable scalar field properties. As an interface to manipulate the deformation, we introduce a sketch-based volumetric peeling interface. The designer's task is to draw a curve on the surface, and pull or push the surface to the desirable position via the curve. Subsequently, the deformation is automatically defined. Results show that a desirable deformation can be easily achieved while preserving implicit properties.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling H.5.2 [Information Interfaces and Presentation]: User Interfaces

1. Introduction

Spatial deformation is a useful technique for modeling and animation. Conceptually, spatial deformation tools embed the object to be deformed in some simplified volumetric space. Deformations applied to the volume are then transferred to the embedded object. For example, Free-form deformation [SP86] embeds objects in 3D lattices; deformations are then specified by manipulating the lattice points. A key advantage of volumetric methods like FFD is that the deformation is independent of the geometry being deformed, so they can be applied to any surface representation based on point-sampling. Artists find these techniques very intuitive, and spatial deformation is widely utilized in commercial 3D modeling packages.

Skeletal implicit surface representations [Blo97] have many advantages for 3D modeling, such as simple formulations for shape blending and CSG, and continuity guarantees. Some spatial deformations, such as Barr warps [Bar84], can be used with implicit representations, but more advanced FFD-like methods are difficult to apply. A key problem is that the warp must be invertible to apply to a functional implicit representation [WGG99]. Second, as the deformation is applied to the entire scalar field, it must largely preserve properties like *normalization* [Sch06] to ensure that further application of blending and CSG operations produce sensible results.

Another issue with spatial deformations based on lattices is that the designer has to manipulate many control points to construct a deformation. Recently, [NISA07] described a deformation tool in which strokes sketched on a surface can be pushed or pulled in 3D space, similar to the Wires system [SF98]. However, these techniques are based on deformation of mesh surfaces, and do not easily extend to the functional implicit domain.

In this work we propose a new sketch-based tool for specifying interactive deformation of functional implicit surfaces. We demonstrate our method using the BlobTree [WGG99], but it is applicable to any other volumetric representation. Similar to [NISA07], the interface is based on drawing curves on the model surface, and then interactively manipulating them. A volumetric peeling technique is used to determine the deformation radius, inspired by curve peeling

algorithms [NISA07]. The volumetric deformation is specified by deforming an automatically-generated lattice, however the designer does not directly interact with the lattice - the warp is specified indirectly via the sketched curve. To preserve desirable properties of the underlying scalar field, the lattice points are used as constraints in the construction of a smooth variational warp, which makes the result usable for further interaction (Figure 1). However, this warp is computationally expensive to compute, so a fast approximation used during interactive manipulation is also described. Our contributions can be summarized as follows:

- We propose a new method for applying free form deformation (FFD) to skeletal implicit surfaces.
- We propose a sketch-based interface to manipulate a deformation that is designed to provide interactive feedback for implicit surfaces although it is not limited to this modeling paradigm.

Figure 1: *Blending and CSG operations can be performed interactively after variational warping.*

2. Related Work

2.1. Free-Form Deformation

Free-form deformation was introduced by Sederberg and Parry [SP86]. They described a tool consisting of a 3D grid of control points (the lattice). The points within this grid are displaced using a tri-variate interpolation with Bernstein polynomials. This assures a smooth deformation field, but requires manipulation of a large amount of control points.

Modifications to the original FFD method were proposed, such as the extended free-form deformation EFFD [Coq90] where the lattice can be made cylindrical. MacCracken and Joy [MJ96] used Catmull-Clark subdivision meshes for lattices of arbitrary topology.

Crespin [Cre99] uses an implicit free-form deformation tool (IFFD) to define the deformation space and the amount of deformation. Each deformation applied to the original model is weighted by the field values of an implicit surface. The big advantage of this method is that a combination of traditional FFD tools can be used to obtain more flexible deformations. Although an implicit surface is used as a deformation field, IFFD can only be applied to point based surfaces like polygon meshes.

Ono et al. [OCNF02] implements a system which automatically generates deformation fields suitable for a model. It is not a trivial task to define appropriate deformation fields, so the system makes FFD more accessible to designers. Since the deformation field is defined hierarchically, it is easy to apply both global and local deformations.

Unfortunately none of the methods discussed provide an inverse mapping, or a means to calculate one, so none of them can be applied to implicit surfaces efficiently. Using root finding or some other recursive algorithm to find inverse values is too time consuming to be used in an interactive system and therefore not a suitable solution.

2.2. Deformation with Implicit Surfaces

Jim Kleck, in his masters thesis [Kle89], mentions the possibility of "space warping" for skeletal implicit surfaces by simply applying the function:

$$f_A(p) = g(w(d_A(p))) \tag{1}$$

In this equation, $d_A(p)$ represents the distance between a point p and a skeleton A. $g : \mathbb{R} \rightarrow \mathbb{R}$ is a field function, and $f_A(p)$ is the field value in p. The function w is often referred to as a space warp. Earlier work such as Kleck's, defined the function w without allowing much in the way of user control.

Barr introduced the notion of global and local deformations using the operations *twist*, *taper* and *bend* [Bar84]. The Barr operations were applied to implicit surfaces by Pasko et al. for functionally defined models [PASS95, PASS01] and for skeletal implicit surfaces see [WvO97]. Cani (formerly Gascuel) [Gas93] defined a warp that represents the deformation of implicit objects under collision and extents this idea in [OC97] and [CD97]. Other specific functional warps applied to skeletal implicit surfaces were defined for animation as described in [Blo97]. Such methods include moving a skeletal implicit model through a warped space, or applying a time dependent warp function to the space in which the model exists.

Schmitt et al. [SPS03] proposed a deformation method for F-rep models [PASS95, PASS01]. A more flexible deformation is possible than the methods described above by defining another F-rep object as a deformation field. By applying this principle in a sketch based system, we are able to design a more intuitive interface, in which it is easy for the designer to define an appropriate deformation field to achieve the desirable deformation.

The above methods work well for implicit models but require a definition of the deformation. In our work we investigate a more controllable and user friendly deformation technique using a sketch-based approach.

2.3. Sketch-Based Modeling

Sketch-based modeling has become a common technique for free-form modeling, and various kinds of sketch-based modeling systems have been presented.

Igarashi et al. [IMT99] introduced the sketch-based modeling system called Teddy. A triangle mesh is used as an underlying shape representation. The system supports deformation operation as well as extrusion, cutting and smoothing operations. The designer draws two strokes and then the system warps the model from one stroke to the other using a deformation operation.

ShapeShop [SWSJ05] is a sketch-based modeling system in the style of Teddy, and uses hierarchical implicit models (BlobTree [WGG99]) as an underlying shape representation. BlobTree allows the designer to create complex models with blending and CSG operations. This process can be done interactively using a hierarchical spatial caching scheme [SWG05]. However, ShapeShop does not support a deformation operation.

FiberMesh [NISA07] is a 3D modeling system that uses 3D curves. In FiberMesh the designer can refine a model with 3D curves which the designer sketches. FiberMesh uses a peeling interface [IMH05] to manipulate deformations, and our approach is similar. The peeling effect is applied to the control curves and the 3D model deforms accordingly. Our approach applies the peeling effect directly to the model instead of a control curve. Moreover, we use a space warping deformation method which allows us to deform any kind of model representation, including implicit surfaces.

3. Deformation Interface

This section provides an overview of the interactive deformation process. Our tool is implemented within the ShapeShop sketch-based modeling system [SWSJ05]. ShapeShop is based on the BlobTree hierarchical implicit volume representation [WGG99]. Similar to Teddy, in ShapeShop the designer constructs a model by drawing 2D contours, which are inflated into 3D parts. These parts correspond to volumetric implicit primitives, which are combined in a dynamic hierarchy using composition operators such as blending and CSG. Hence, the representation is a tree, with primitives at the leaves, and interior nodes specifying compositions. Our deformation tool is also implemented as a composition operator, and hence its influence is localized according to the position of the deformation node in the BlobTree.

To begin a deformation, the designer draws a 2D stroke over top of the model. The system dynamically interprets this stroke as a request for a deformation action, and adds an appropriate icon to the suggestion list. Upon initiating the deformation, the vertices of the stroke are projected onto the current implicit surface, via ray-surface intersection. This produces a 3D poly-line that approximates a curve embedded in the surface. To deform the surface, the designer manipulates the embedded curve using 3D transformation tools. Similar to Wires [SF98], translating the curve pulls the surface along with it.

Conceptually, the curve is treated as a constraint in the underlying deformation. The designer is free to add more curves to the deformation, which are simply treated as additional constraints. Note that, without any special handling, deformation curves could easily conflict with each other. To avoid this, curves are dynamically updated to ensure that they are embedded in the currently-visible surface. When one of the curves is manipulated, the warp is re-computed using this new curve and all the previous curves. We then project all other curves onto the new surface (Figure 2).

Figure 2: *The designer can pull the model from several positions with multiple curves.*

3.1. Volumetric Peeling Technique

A standard interface problem with local volumetric deformation techniques is specifying the region of influence (ROI). In our system, as the designer manipulates constraint curves, vertices of an underlying volumetric lattice are modified (See Section 4 for details). As with other systems, we provide an interactive parameter for controlling the ROI of the curve on the lattice. However, the designer often has to repeatedly alternate between moving the curve and modifying the radius parameter to achieve the desired effect, which can be time-consuming.

An alternative to manual ROI specification is a dynamic *peeling* interface, as proposed by [IMH05]. In that system, the designer interactively deforms 2D curves by directly pulling on them. The further the designer pulls the curve from an initial point, the larger the region of influence along the curve (here determined by arc-length). [NISA07] includes a similar interface for 3D curve manipulation. We utilize a similar interface, but instead of peeling an ROI on the curve, we peel the volumetric ROI of the deformation specified by the curve.

When the designer first creates a deformation, the ROI starts out small. As the designer pulls the curve a small distance from the surface, a correspondingly small region is deformed. As the displacement of the curve grows, so does the

deformation ROI. When the designer pulls the curve sufficiently far away, the whole surface is deformed (Figure 3). The deformation region is indicated by changing color, from green to yellow. Hence, the effect for the designer is similar to an elastic deformation of a very flexible surface - initially only a local deformation is specified, but eventually the whole surface begins to move.

Volumetric peeling allows the designer to efficiently vary deformation ROI without having to manipulate parameters. The ROI growth rate is tuned to be compatible with the limitations of the deformation algorithm, and we have found it to be quite effective in practice. Even when a different ROI is desired, volumetric peeling generally gives a better "ballpark" estimate than a fixed ROI. Since designers will inevitably demand ROI control, we provide a slider which modulates the ROI growth rate, which loosely corresponds to the elastic stiffness or rigidity of the material (Figure 4).

Figure 3: *A peeling interface. Depending on how far the designer pulls the curve, the deformation region is determined.*

Figure 4: *The rigidity of the surface can be also manipulated by changing the growth rate.*

4. Deformation Algorithm

Our algorithm consists of three parts: voxelization, interactive deformation, and variational warping. Voxelization involves approximating the model with a grid of cubic voxels and associating them with the user-drawn curve. The designer's actions displace the vertices of this grid, and we infer our deformation from these displacements. Interactive deformation defines a new approximate implicit field based on the voxel grid, visually approximating the effect of the deformation. Variational warping computes a spatial deformation field based on the displacements, warping the implicit model.

Ideally, we would compute the deformation field in one pass and avoid our approximate deformation, however variational warping is too slow for interactive use. Therefore, we exploit the speed of the approximation and then either let the designer explicitly compute the variational warp, or take advantage of idle moments to do so, similar to the approach proposed in [BPWG07].

4.1. Voxelization

The designer draws a curve on the surface of the model to define the region of the deformation (Figure 5). A set of voxels are then created in the region surrounding the user-drawn curve. We call this the deformation grid. In ShapeShop, the implicit model is visualized using a polygon mesh extracted from a similar uniform voxelization [WMW86]. Hence, the fidelity of the surface visualization is limited by the voxel resolution. The deformation grid is also created in the same manner. However, the deformation grid is independent of polygonization, so the designer can use arbitrary grid resolution without affecting polygonization. This grid resolution limits the spatial frequency of the deformation, but as a smooth warp field is computed, the spatial frequency of the underlying implicit surface is not affected.

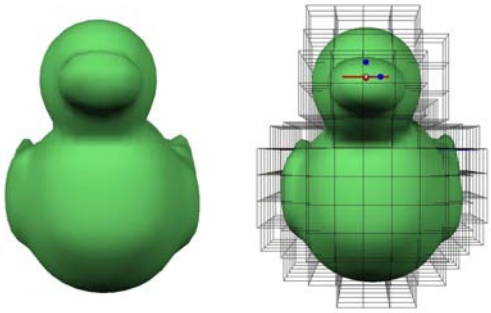

Figure 5: *The surface of the model is automatically voxelized right after sketching.*

Next, we embed the user-drawn 3D curve in the voxel grid. For each curve vertex, we find the nearest grid vertex - the set of all such grid vertices are the *handle vertices* (Figure 7). When the designer translates the curve, the same translation is applied to the handle vertices. This translation is then propagated to the each surrounding grid vertex, with a smoothly decreasing spatial influence based on the distance to the handle.

To define the smoothly decreasing weight on the interactive translation, we must compute the distance from each grid vertex to the handle. To reduce computational cost, we use a discrete propagation scheme to approximate these distances. The goal is to compute an approximate distance d for each vertex v with index (i, j, k). We initialize the handle distances to $d = 0$, and then sequentially propagate distances outward from each handle vertex using Dijkstra's algorithm on the graph of vertex neighbours, where the neighbours of v are the vertices v_n with indices $\{(i+p, j+q, k+r) : p, q, r \in [-1, 1]\}$. The distance at v is approximated by

$$d = d_n + s_{voxel}\sqrt{|i - i_n| + |j - j_n| + |k - k_n|} \qquad (2)$$

where d_n is the distance stored in a neighbour vertex (i_n, j_n, k_n), and s_{voxel} represents the size of a voxel. The

set of possible distance deltas are constant and can be precomputed (Figure 6).

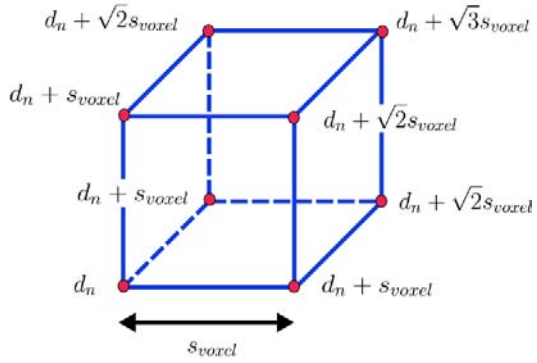

Figure 6: *An example of distance approximation. The distance from the handle vertex is calculated according to the distance from the neighbour vertex d_n.*

After this process, each vertex stores the approximate distance from the handle vertices (Figure 7). Note that if the designer has drawn multiple curves, we do this computation independently for each curve. Each vertex stores the separate shortest-distance for every curve in the deformation.

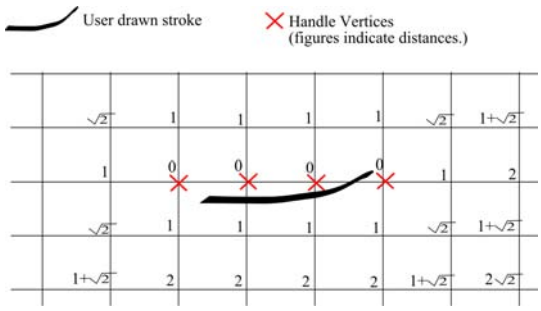

Figure 7: *Once the curve has been drawn, the handle vertices are found. Each vertex stores the approximate shortest distance to the handle, found using Dijkstra's algorithm.*

4.2. Interactive Deformation Approximation

During interactive manipulation, we take the curve translation as input and attempt to approximate the appearance of the deformed surface in real-time. The goal is to give interactive feedback to the designer. No time need be spent to preserve the field for subsequent use as that will be done in a separate pass.

When the system receives the translation T from the designer, the amount of translation of a vertex v is calculated as:

$$T_v = \begin{cases} g(\frac{d}{r\|T\|}) \cdot T & \|T\| \neq 0 \\ 0 & \|T\| = 0 \end{cases} \quad (3)$$

$$g(x) = \begin{cases} (1-x^2)^3 & x \leq 1 \\ 0 & x > 1 \end{cases} \quad (4)$$

where d is the distance at v, $r > 0$ is the voxel rigidity parameter, and $g(x)$ is the *Wyvill* function [Blo97] which smoothly decays from 1 to 0. Translation is applied to v if d is less than $r\|T\|$. Therefore, the more the curve is translated, the more the vertices are translated (Figure 8). At handle vertices, $d = 0$ and the full translation is applied.

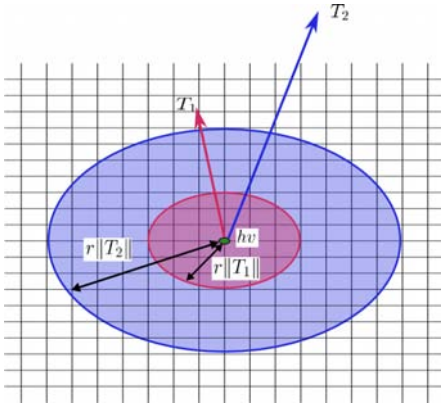

Figure 8: *When the handle vertex (hv) is translated with T_1, the vertices in the red circle are influenced. The same thing happens with T_2.*

If multiple curves are drawn by the designer, v has n distances d_i from n curves. We compute the translation T_{c_i} for each curve, and combine them to find T_v:

$$T_v = \sum_{i=1}^{n} g(\frac{d_i}{r\|T_{c_i}\|}) \cdot T_{c_i} \quad (5)$$

We can now compute a displacement field D which can be applied to any point inside the voxel grid by interpolating the translations T_{v_i} at each vertex v_i. The deformed position q if a point p in A is defined as (Figure 10a):

$$q = p + D(p) \quad (6)$$

The difficulty with implicit surfaces is that they are not defined as a set of points, but by an arbitrary scalar function $f(p)$, and we generally cannot compute the deformed scalar field $D(f(p))$. However, an implicit surface is visualized by evaluating f at a large number of sample points. At a point q in the deformed field, the field value is then $f(D^{-1}(q))$. Hence, we construct an inverse deformation field $\widetilde{D}$ which approximates D^{-1} (Figure 10b).

$\widetilde{D}$ is constructed by interpolating the inverse translations of vertices $-T_{v_i}$ using the Wyvill function (Equation 4). We suppose that every vertex is an implicit point primitive bounded by R. We are currently using two times the size of a voxel as R. When q is given, we efficiently find the set of vertices which influence q using a uniform spatial subdivision search structure.

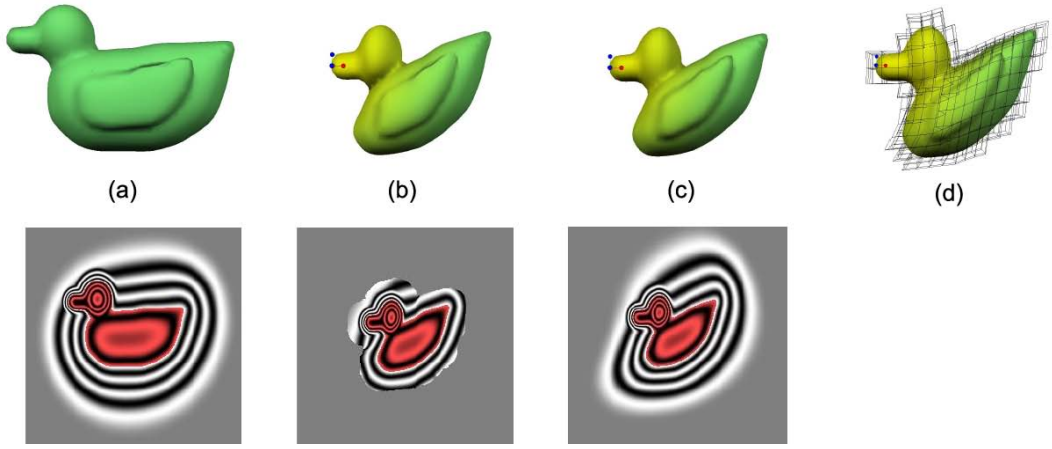

Figure 9: *An example deformation procedure with the duck shown in Figure 5. The top row shows the results of the visualized implicit model. The bottom row shows the implicit field images which were sampled from a slice along a plane. The original shape of the duck is shown in (a). By pushing the bill of the duck, the duck is squashed interactively in interactive deformation (b). However, the implicit field outside of the deformation field is lost. Once the model has been deformed, the implicit field of the duck is re-calculated with variational warping (c). A decent implicit field can be preserved with variational warping and the field is cached for further modeling. The deformed voxels are also shown in (d).*

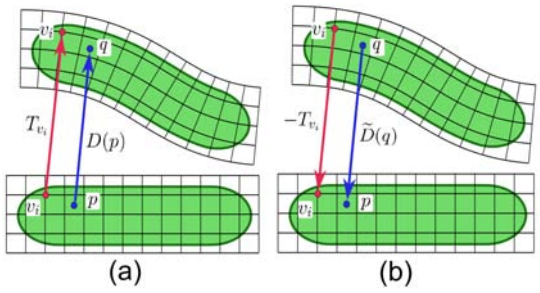

Figure 10: *Deformation for point based surface representations can be achieved with (a). However, inverse deformation is required to deform an implicit surface (b).*

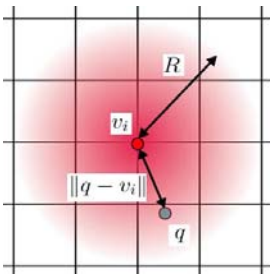

Figure 11: *q is influenced by v_i. The amount of the influence is calculated with Wyvill function. The amount is used as the weight for the interpolation.*

If a vertex v influences q, the field value at q is calculated as the weight using the Wyvill function: $g(\frac{\|q-v\|}{R})$ (Figure 11). The inverse translation $-T_v$ is weighted by the field value and then summed. Finally, the final result is divided by the sum of the field values, *sum*. We must also handle the case where q is outside $\widetilde{D}$ (Figure 12), there $\widetilde{D}(q)$ cannot be calculated as $sum = 0$, so $f_{A'}(q)$ returns 0. Hence, The field value at q is defined as:

$$f_{A'}(q) = \begin{cases} f_A(q + \widetilde{D}(q)) & sum \neq 0 \\ 0 & sum = 0 \end{cases} \tag{7}$$

$$\widetilde{D}(q) = -\frac{\sum_{i=1}^{m} g(\frac{\|q-v_i\|}{R}) \cdot T_{v_i}}{sum} \tag{8}$$

$$sum = \sum_{i=1}^{m} g(\frac{\|q - v_i\|}{R}) \tag{9}$$

Here again $g(x)$ is used as a smooth blending function. Since $f_{A'}(q)$ is 0 outside the voxel grid, the field is not smooth (Figure 9b), however as the iso-surface lies inside the voxels, this is acceptable. This technique produces a real-time interactive approximation to the final deformation, which we describe in the next section.

4.3. Variational Warping

When the designer is satisfied with the deformed model, variational warping described in [BK05] is applied to the model. Since variational warping has to solve a large matrix, the computational cost is high. However, it guarantees to globally interpolate all constraints (vertices of voxels)

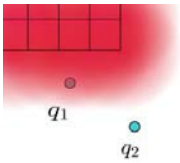

Figure 12: *The field value of q_1 is calculated with Wyvill function. However, the field value of q_2 cannot be calculated because no vertex influences q_2. In this case, the system simply returns 0 as the field value of q_2.*

with C^2 continuity. We do not put constraints outside the isosurface, so we cannot preserve a complicated implicit field. But the result is suitable to apply further blending and CSG operations (Figure 9c).

As in the previous section, our variational warp is essentially an inverse deformation $\widetilde{D}$. Therefore, the field value at q can be calculated as follows:

$$f_{A'}(q) = f_A(q + \widetilde{D}(q)) \tag{10}$$

$$\widetilde{D}(q) = \sum_{i=1}^{m} w_i \varphi(\|q - v_i\|) + P(q) \tag{11}$$

where $\varphi(r) = r^3$, and $P(q) = c_1 q_x + c_2 q_y + c_3 q_z + c_4$. The weights $w_i \in \mathbb{R}^3$ and coefficients c_1, c_2, c_3, and $c_4 \in \mathbb{R}^3$ can be calculated by solving a linear system by evaluating equation 11 at each known solution $\widetilde{D}(v_i) = -T_{v_i}$.

5. Results

The body of the dragon shown in Figure 13 was deformed with our deformation technique. The body was created at first and then additional parts such as the head and the legs were added to the body with blending operation. It is difficult to deform an implicit model as the designer wants with existing deformation techniques for implicit surfaces. Even after deformation, smooth blending can be applied because of variational warping and hierarchical spatial caching.

Our system is also useful to edit a model (Figure 14). The octopus was created with standard ShapeShop operations. The face of the octopus was deformed by pulling the curve drawn on the mouth. Since the ROI of the octopus is automatically specified, various kinds of face shapes can be intuitively achieved just by pulling the mouth. It is difficult to deform like Figure 14 with an interface peeling an ROI on the curve [NISA07]. By directly recording the deformation sequence, animation can be created easily.

6. Limitations

Existing FFD methods generally do not deal with self-collision. Similarly, our system does not handle self-collision. To enable interactive manipulation, we chose to split the deformation procedure. Interactive deformation uses a different interpolation technique than the final variational warp, so the resulting shapes are slightly different.

Our volumetric peeling interface is achieved by approximating the shortest distance from the curve to voxel vertices. If surfaces are not connected with each other but connected by voxels, deformation behaves as if the surfaces were connected. This can be avoided by using voxels of higher resolution, but computational cost increases.

Figure 13: *The dragon was deformed with our deformation technique (top). The body of the dragon was initially modeled on a plane (left). The body was deformed by pushing the middle of the body to the right and pushing the tail to the left (right).*

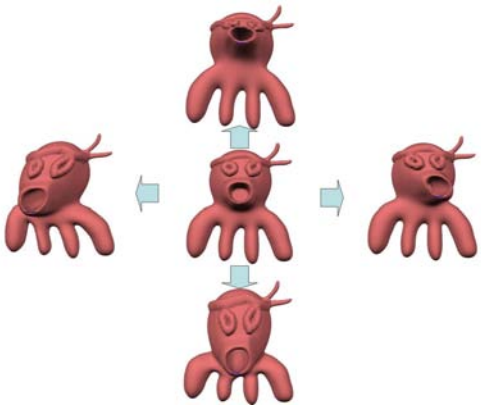

Figure 14: *The octopus model was initially sketched, and then the mouth of the octopus was pulled. The picture shows the results by pulling the mouth to the each direction.*

7. Conclusion and Future Work

While skeletal implicit surface representations have many 3D modeling advantages, applying FFD to implicit surface representations is difficult. Achieving a desirable deformation with lattice-based FFD can also be time-consuming. The system we have described addresses those two problems. Using a variational warp, FFD deformation for implicit surface representations can be achieved while preserving implicit properties, allowing smooth blending to be applied after deformation. Our volumetric peeling interface also allows the designer to intuitively deform a model without requiring manual parameter manipulation.

Our current system provides just one sketch-based operation for deformation(drawing a curve on the surface and pulling or pushing it). Future work can explore more deformation operations like twist and taper. Our current volumetric peeling interface is not limited to implicit surface representations, it can be applied to any point-sampled representation. Future experimentation could explore this and then be incorporated into the volumetric peeling interface.

References

[Bar84] BARR A. H.: Global and local deformations of solid primitives. In *ACM Computer Graphics SIGGRAPH '84* (1984), pp. 21–30.

[BK05] BOTSCH M., KOBBELT L.: Real-time shape editing using radial basis functions. *Comput. Graph. Forum 24*, 3 (2005), 611–621.

[Blo97] BLOOMENTHAL J.: *Introduction to Implicit Surfaces*. Morgan Kaufmann, ISBN 1-55860-233-X, 1997.

[BPWG07] BOTSCH M., PAULY M., WICKE M., GROSS M.: Adaptive space deformations based on rigid cells. *Computer Graphics Forum 26*, 3 (2007), 339–347.

[CD97] CANI M.-P., DESBRUN M.: Animation of deformable models using implicit surfaces. *IEEE Trans. Vis. Comput. Graph. (TVCG) 3*, 1 (1997).

[Coq90] COQUILLART S.: Extended free-form deformation: a sculpturing tool for 3d geometric modeling. In *Proc. SIGGRAPH '90* (1990), pp. 187–196.

[Cre99] CRESPIN B.: Implicit free-form deformations. In *Proc. Implicit Surfaces '99* (1999), pp. 17–23.

[Gas93] GASCUEL M.-P.: An implicit formulation for precise contact modeling between flexible solids. In *Proc. SIGGRAPH '93* (1993), pp. 313–320.

[IMH05] IGARASHI T., MOSCOVICH T., HUGHES J. F.: As-rigid-as-possible shape manipulation. *ACM Trans. Graph. 24*, 3 (2005), 1134–1141.

[IMT99] IGARASHI T., MATSUOKA S., TANAKA H.: Teddy: a sketching interface for 3d freeform design. In *Proc. SIGGRAPH '99* (1999), pp. 409–416.

[Kle89] KLECK J.: *Modeling Using Implicit Surfaces*. Master's thesis, University of California, Santa Cruz, June 1989.

[MJ96] MACCRACKEN R., JOY K. I.: Free-form deformations with lattices of arbitrary topology. In *Proc. SIGGRAPH '96* (1996), pp. 181–188.

[NISA07] NEALEN A., IGARASHI T., SORKINE O., ALEXA M.: Fibermesh: designing freeform surfaces with 3d curves. *ACM Trans. Graph. 26*, 3 (2007).

[OC97] OPALACH A., CANI M.-P.: Local deformation for animation of implicit surfaces. In *Spring Conference on Computer Graphics (SCCG)* (1997).

[OCNF02] ONO Y., CHEN B.-Y., NISHITA T., FENG J.: Free-form deformation with automatically generated multiresolution lattices. In *CW* (2002), pp. 472–479.

[PASS95] PASKO A., ADZHIEV V., SOURIN A., SAVCHENKO V.: Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer 11*, 8 (1995), 429–446.

[PASS01] PASKO A., ADZHIEV V., SCHMITT B., SCHLICK C.: Constructive hypervolume modeling. *Graphical models 63*, 6 (2001), 413–442.

[Sch06] SCHMIDT R.: *Interactive Modeling with Implicit Surfaces*. Master's thesis, University of Calgary, 2006.

[SF98] SINGH K., FIUME E. L.: Wires: A geometric deformation technique. In *Proc. SIGGRAPH 98* (1998), pp. 405–414.

[SP86] SEDERBERG T., PARRY S.: Free Form Deformation of Solid Geometric Models. *Proc. SIGGRAPH 86* (1986), 151–160.

[SPS03] SCHMITT B., PASKO A., SCHLICK C.: Shape-driven deformations of functionally defined heterogeneous volumetric objects. In *Proc. GRAPHITE 2003* (2003), pp. 127–134.

[SWG05] SCHMIDT R., WYVILL B., GALIN E.: Interactive implicit modeling with hierarchical spatial caching. In *SMI* (2005), pp. 104–113.

[SWSJ05] SCHMIDT R., WYVILL B., SOUSA M., JORGE J.: Shapeshop: Sketch-based solid modeling with blobtrees. In *Proc. SBIM 2005* (2005), pp. 53–62.

[WGG99] WYVILL B., GUY A., GALIN E.: Extending the CSG tree. warping, blending and boolean operations in an implicit surface modeling system. *Computer Graphics Forum 18*, 2 (1999), 149–158.

[WMW86] WYVILL G., MCPHEETERS C., WYVILL B.: Data structure for soft objects. *The Visual Computer 2*, 4 (1986), 227–234.

[WvO97] WYVILL B., VAN OVERVELD K.: Warping as a modelling tool for csg/implicit models. In *Shape Modelling Conference, University of Aizu, Japan* (1997).