Free-Form Deformation for Implicit Surfaces

by

Masamichi Sugihara
B.Sc., Hosei University, 2006

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Free-Form Deformation for Implicit Surfaces

by

Masamichi Sugihara
B.Sc., Hosei University, 2006

Supervisory Committee

Dr. Brian Wyvill, Supervisor
(Department of Computer Science)

Dr. Amy Gooch, Departmental Member
(Department of Computer Science)

Dr. Bruce Gooch, Departmental Member
(Department of Computer Science)

Supervisory Committee

Dr. Brian Wyvill, Supervisor
(Department of Computer Science)

Dr. Amy Gooch, Departmental Member
(Department of Computer Science)

Dr. Bruce Gooch, Departmental Member
(Department of Computer Science)

ABSTRACT

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 thesis, an FFD technique suitable for implicit surface representations is described. To enhance real-time feedback, the problem is split 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, a sketch-based volumetric peeling interface is introduced. The user'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. This technique has been implemented in a prototype implicit FFD system called Taco. Results created in Taco show that a desirable deformation can be easily achieved while preserving implicit properties.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

First I would like to thank my M.Sc. supervisor, Brian Wyvill for his patient support throughout my master's degree program. His kind assistance and encouragement was helpful for me to achieve a master's degree in Canada which was a completely new environment for me. His technical advice and suggestions also led me to research on the right track.

I wish to thank Ryan Schmidt and Erwin de Groot for their advice and helping me to research. I was able to accomplish this research thanks to their cooperation. Although my correspondence with Ryan was mostly on email, his explanation was easy to understand and contained many good ideas. Actually, I have not even met him yet, so I hope to do so in the future. I have met Erwin several times at UVic. It was valuable to discuss with him in person as he provided me with several helpful hints to solve problems.

I am grateful to my examining committee members, Amy Gooch, Bruce Gooch, and Alexandra Albu for accepting to be my examining committee. They provided many valuable comments and suggestions to refine my thesis during the oral examination.

I would like to thank all the members of the UVic Graphics Lab for providing an exciting opportunity for research. Especially, Zainab Meraj, Chris Serson, Sven Olsen, Jeremy Long, Anthony Estey, Pourya Shirazian, Stijn Stiefelhagen, and Ian Bentley. Thank you for valuable discussion, suggestions and sometimes having fun by playing games.

I wish to thank the members of Intel Advance Rendering Team in Victoria, Paul Lalonde, Jean-Luc Duprat, Jefferson Montgomery, and Andrew Lauritzen for helping me to work comfortably at Intel. This internship was a valuable experience to try my skill in the industry and it also stimulated my research.

Finally, I am grateful to my family and friends. I was able to research as a master student in Canada simply because of the support of my parents. I really appreciate their support, understanding and encouragement. My friends also supported me in various ways, such as entertainment, encouragement and friendship.

Chapter 1

Introduction

Deformation is a useful technique for modeling and animation. Deforming an object can achieve various kinds of shapes and also animate the geometric changes of the object. The first introduced deformation technique is Barr warps [4] which allow for global and local deformations using the operations such as bend, twist, and taper. This technique employs spatial deformation which is the most commonly used deformation technique in computer graphics.

A key advantage of spatial deformation is that the deformation is independent of the geometry being deformed, so spatial deformation can be applied to any shape representation based on point-sampling. One issue in deformation is that many kinds of underlying shape representations exist to define 3D models, which means a corresponding deformation technique is required for each shape representation. Spatial deformation can avoid this issue. Also, it is independent of the complexity of the embedded model, so high complex models can be easily handled. 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 [50] embeds objects in 3D lattices; deformations are then specified by manipulating the lattice points. Artists find these techniques are interactive and intuitive, and spatial deformation is widely utilized in commercial 3D modeling packages.

Free-form deformation (FFD) generally provides a user-friendly interface to manipulate deformations. The interface must be designed so that the manipulation method is easy enough and real-time feedback is also given to the users. This kind of system is extremely useful as a tool to make animation. The most common FFD tool is a 3D lattice which is intuitive and interactive enough. Nevertheless, the bottleneck

of lattice manipulation is that many control points must be manipulated to achieve a desirable deformation. It is also tedious to define a well fit lattice for the object being deformed.

3D implicit modeling [7] has many advantages for 3D modeling. For example, shape blending and Constructive Solid Geometry (CSG) can be defined with simple formulations. Therefore, topological changes can be treated trivially and continuity [1] is also guaranteed. These modeling features are quite difficult to achieve in other 3D shape representations. Nevertheless, defining deformations in implicit surfaces is problematic. Some spatial deformations, such as Barr warps [4], can be used with implicit surfaces, but more advanced FFD-like methods are difficult to apply.

The goal of this research is to develop an FFD for implicit surfaces. To achieve the goal, an FFD technique suitable for implicit surfaces must be proposed as well as better FFD tools than 3D lattices must be developed.

1.1 The Problem

Although 3D implicit modeling offers many modeling advantages, it is difficult to define deformations. A key problem is that the warp must be invertible to apply to a functional implicit representation [62]. This is because implicit representations are volume representations which are not defined as a set of points but by an arbitrary scalar function. Hence, the forward warp cannot be applied to implicit surfaces directly. Second, as the deformation is applied to the entire scalar field, it must also largely preserve implicit properties to ensure that further application of blending and CSG operations produces usable results. In point based representations such as polygon meshes, only the vertices which define the surface need to be warped to achieve the deformation. Because implicit surfaces are defined as a scalar field, however, the entire scalar field must be deformed as well as the iso-surface. This procedure is quite complicated compared to point based representations. Also, a good scalar field must be preserved after deformation to maintain predicable affects of implicit properties.

Another issue is that lattice-based FFD can be time-consuming. A 3D lattice is an intuitive and interactive FFD tool but many control points must be manipulated to construct a desirable deformation. An appropriate 3D lattice must also be defined

[1] The degree is dependent on the field function.

by the user. Recently, [32] described a deformation tool in which strokes sketched on a surface can be pushed or pulled in 3D space, similar to the Wires system [53]. Nevertheless, these techniques are based on deformation of mesh surfaces, and do not easily extend to the functional implicit domain. In this work, an improvement over the 3D lattice as an FFD interface for implicit surfaces is introduced.

1.2 The Solution

In this work, a novel sketch-based tool for specifying interactive deformation of functional implicit surfaces is introduced. This tool has been implemented in the system called Taco. Similar to [32], the interface of Taco 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 [32]. The volumetric deformation is specified by deforming an automatically-generated lattice surrounding the iso-surface, however the user does not directly interact with the lattice. The control points of the lattice are manipulated indirectly via the sketched curve.

An invertible warp is constructed with a smooth variational warp technique [8] using the control points of the lattice as constraints. The variational warp can preserve desirable properties of the underlying scalar field, so it makes the result usable for further interaction (Figure 1.1). Nevertheless, this warp is computationally expensive. To enhance real-time feedback, therefore, a fast approximation used during interactive manipulation is also introduced. In this framework, the user first interactively deforms implicit models with sketching. The system approximates the appearance of the deformed surface and provides it for the user in real-time. When the user is satisfied with the deformed surface, the variational warp is applied to the surface for further modeling.

1.3 Contributions

The contributions of this work can be summarized as follows:

- A new method for applying free-form deformation (FFD) to implicit surfaces.
- A sketch-based interface to manipulate a deformation that is designed to provide interactive feedback for implicit surfaces.

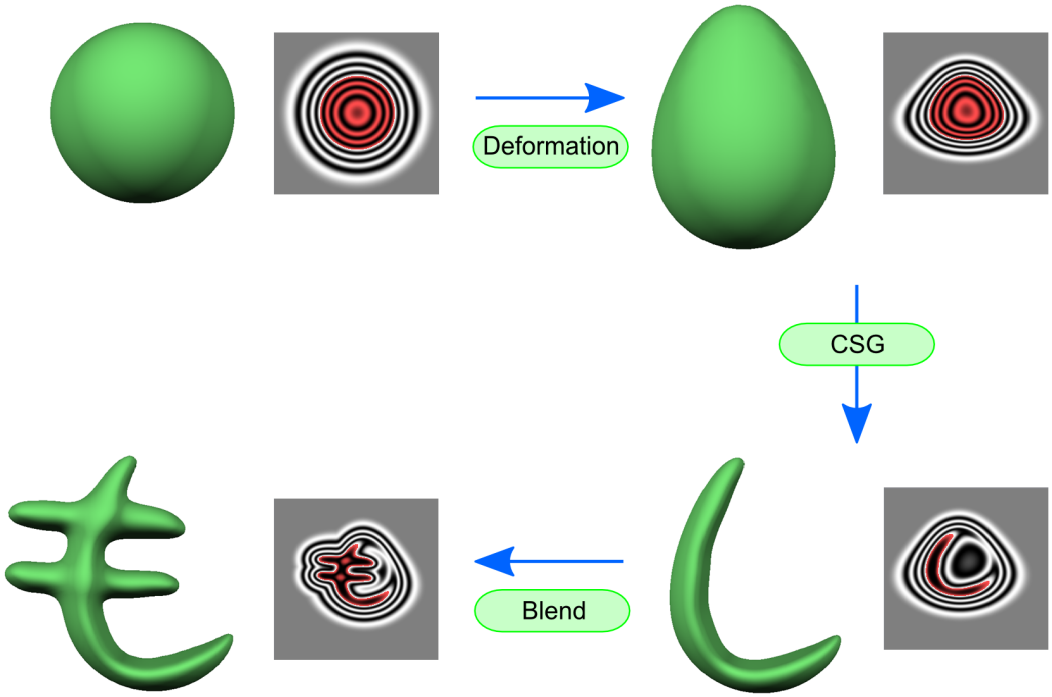


Figure 1.1: Deformation for implicit surfaces with a variational warp. Since the warp is applied not only to the surface but also the entire scalar field, the desirable scalar field property is preserved. Therefore, implicit operators, such as blending and CSG, can be applied even after deformation. In this figure, the iso-surface is highlighted in red.

- A method for re-forming a good scalar field after a warping operation. This scalar field maintains implicit properties.

While implicit surfaces have many 3D modeling advantages, applying an FFD to implicit surfaces is difficult as indicated in Section 1.1. Achieving a desirable deformation with lattice-based FFD can also be time-consuming. The system developed in this work addresses these two problems. Using a variational warp [8], an FFD for implicit surfaces can be achieved while maintaining desirable implicit properties, allowing smooth blending to be applied after deformation. The volumetric peeling interface allows the user to intuitively deform a model without requiring manual parameter manipulation. This FFD technique is not limited to this implicit modeling paradigm. In this work, the method is demonstrated and validated using the Blob-Tree [60], but it is applicable to any other representation.

1.4 Overview

The rest of this thesis is composed as follows. Chapter 2 contains the introduction to implicit surfaces and the BlobTree [60]. In Chapter 3, the background and previous work related to this work are presented. This chapter contains the survey of several deformation techniques including FFD, deformation for implicit surfaces, and sketch-based modeling systems. In Chapter 4, the deformation interface employed in Taco is introduced. A volumetric peeling interface is presented which can determine the deformation radius intuitively. In Chapter 5, an FFD based algorithm applied to implicit surfaces is described along with a volumetric peeling interface. In Chapter 6, Taco is demonstrated and evaluated by showing some results. Some limitations, future work, and the conclusion of this work are also contained.

This work has been published in The Fifth Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM) 2008 as *A Sketch-Based Method to Control Deformation in a Skeletal Implicit Surface Modeler* [54] by Sugihara, de Groot, Wyvill and Schmidt. The contributions described in this thesis are the work of the author, however the interactive system presented is based on the work of Schmidt.

Chapter 2

Introduction to Implicit Surfaces

Implicit surfaces are defined with scalar functions. A point $p \in \mathbb{R}^3$ is assigned a value called a *field value* $f(p) \in \mathbb{R}$ by applying a continuous scalar function $f : \mathbb{R}^3 \to \mathbb{R}$. The points which store the value called the *iso-value* form a surface called an *iso-surface*. An iso-surface can be defined as follows:

$$S = \{p \in \mathbb{R}^3 : f(p) = v_{iso}\} \tag{2.1}$$

where S is the iso-surface and v_{iso} is the iso-value. If an iso-surface is closed, this scalar function can also define an *implicit volume* by simply changing the equality into the inequality as follows:

$$V = \{p \in \mathbb{R}^3 : f(p) \geq v_{iso}\} \tag{2.2}$$

This equation divides 3D space into inside and outside using the iso-surface as the border. For example, a point p where $f(p) \geq v_{iso}$ is inside a volume V.

In practice, an iso-surface is visualized on a screen and recognized as a 3D model, however the strength of implicit surfaces is that an underlying model is a volume model and has several advantages of 3D modeling. Fundamental issues of implicit surfaces are then how to define a continuous scalar function f (Section 2.1 Modeling) and how to visualize the scalar field (Section 2.2 Rendering).

2.1 Modeling

2.1.1 Scalar Field Construction

There are several methods to define a scalar field for implicit surfaces. In this section, three scalar field construction methods, a *skeletal method*, *variational interpolation*, and *Function-Representation* (FRep), are introduced.

1. **Skeletal Method**

   A skeletal method constructs a scalar field by applying a *distance function* to a geometric shape called a *skeleton*. A simple geometric shape, such as a point and a line, is often used as a skeleton. A skeleton is placed in 3D space and a scalar field is created according to the distance $d(p) \in \mathbb{R}$ from the skeleton (Figure 2.1). The generated implicit shape is referred to as a *skeletal primitive*. The equation of an skeletal primitive can be defined as follows:

   $$f(p) = g \circ d(p) \tag{2.3}$$

   where $g : \mathbb{R} \rightarrow \mathbb{R}$ is the distance function. Several distance functions were proposed such as the *Blobby Molecules* function [6], the *Metaballs* function [33], the *Soft Objects* function [63], and the *Wyvill* function [7]. The distance functions are fall-off functions as shown in Figure 2.2, so a field value of a scalar field created with a distance function gradually decreases from a skeleton. In particular, the Metaballs function, the Soft Objects function, and the Wyvill function generate a scalar field with a zero field outside the finite distance from the skeleton. Such a scalar field is called a *bounded scalar field*. Bounded scalar fields can guarantee local influence which means they do not influence any other primitives outside the boundary. This property is important for interactive modeling. This is because if the user edits a primitive whose scalar field is not bounded, the shape of another primitive will also change. This change is not intuitive for the user because only the iso-surface is visible to the user and the underlying scalar field affecting another primitive is invisible.

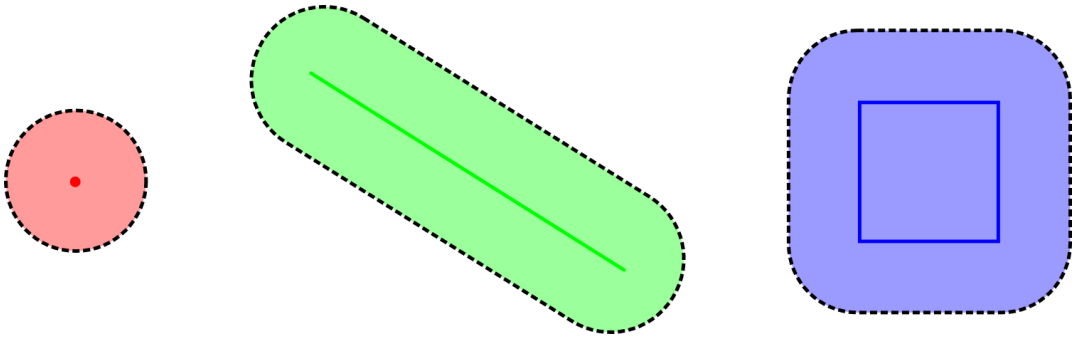
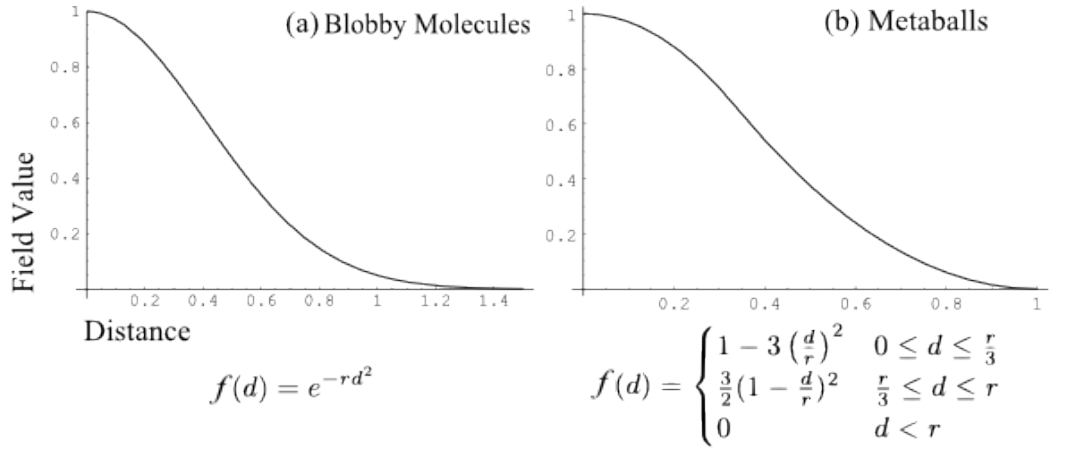


Figure 2.1: Skeletons and skeletal primitives generated from the skeletons. Deep color frames represent skeletons and light color areas are skeletal primitives.



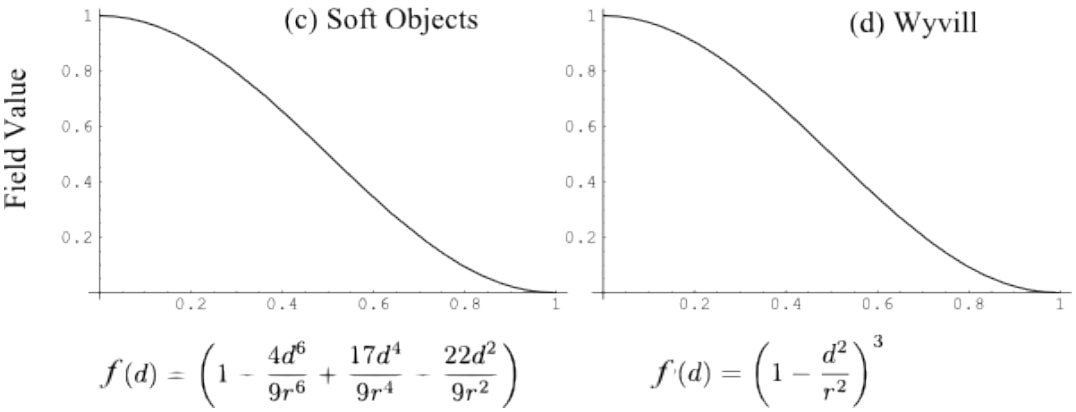


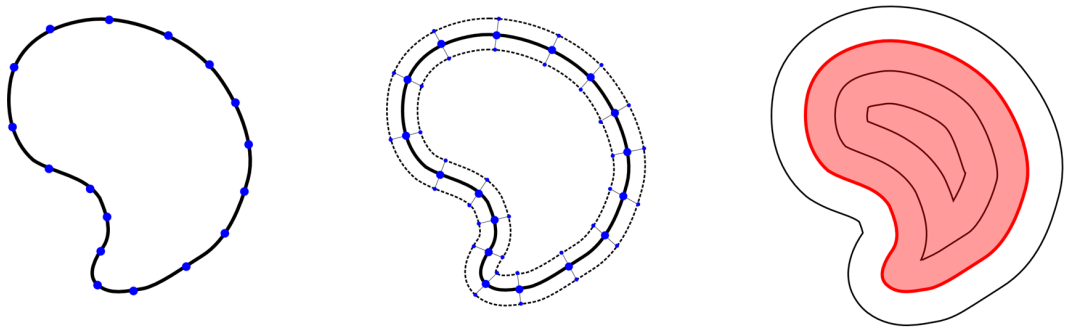
Figure 2.2: 4 distance functions. The Blobby Molecules function (a), the Metaballs function (b), the Soft Objects function (c), and the Wyvill function (d). These 4 functions are fall-off functions.

2. **Variational Interpolation**

   Variational interpolation is also used to define a scalar field [43, 57, 11, 58, 46]. This interpolation method is based on a radial basis function (RBF) which is a technique to solve the scattered data interpolation problem. A set of points where an iso-surface should pass are sampled and assigned the iso-value (Figure 2.3a). They are then interpolated using variational interpolation. Nevertheless, variational implicit surfaces cannot be constructed with this interpolation only because a trivial solution of this interpolation is $f(p) = v_{iso}$ where $p \in \mathbb{R}^3$. This means that the iso-value is assigned to every point in 3D space. To avoid this solution, *off-surface points* are sampled as additional constraints. Two off-surface points are defined to each sampled point along vectors normal to the iso-surface (Figure 2.3b). They are placed inside and outside the iso-surface. A variational implicit surface is then created by interpolating those sampled points (Figure 2.3c). Variational interpolation globally interpolates all constraints, so the result scalar field is not bounded. The method of [46] generates a bounded scalar field by applying the Wyvill function after variational interpolation. The advantages of variational interpolation are that a generated scalar field has C^2 continuity and arbitrary shapes can be flexibly defined. On the other hand, the computational cost of this method is very expensive. The accuracy of this method also really depends on the number of sampling points. Therefore, reducing the number of sampling points can speed up this calculation but results in poor interpolation. This method is often used to convert polygon meshes into implicit surfaces by assigning the iso-value to the points on the polygon meshes. Again, the polygon meshes can be approximated accurately by increasing the number of sampling points but the computational cost becomes more expensive.

3. **Function-Representation (FRep)**

   FRep is a representation to functionally define primitives [40, 39]. As a simple example, a sphere can be represented with this function: $f(p) = r^2 - \|p - p_o\|^2$, where r is the radius of the sphere and p_o is the center. Other primitives, such as an ellipsoid, a cylinder, and a torus, are also defined functionally like the sphere function, however FRep supports not only such functional definitions but also different representations such as skeletal primitives, parametric models, sweep models, and so forth. The concept of FRep is to provide C^n continuous definitions using R-Functions [51, 52]. Operators of FRep are based on R-



(a) (b) (c)

Figure 2.3: Variational interpolation procedure. A set of points are sampled along the contour and work as constraints of interpolation (a). Two additional points are assigned to each sampled point as off-surface points (b). Off-surface points are placed inside and outside the iso-surface and are along vectors normal to the iso-surface. A scalar field is created by applying variational interpolation to sampled points (c).

Functions and can define C^n continuous CSG operators (Section 2.1.2). FRep generally takes zero as the iso-value, and negative values represent outside and positive values represent inside. Hence, this scalar field is not bounded. A modeling language called HyperFun [1] has been developed based on the FRep concepts.

2.1.2 CSG

Constructive Solid Geometry (CSG) is a procedural modeling method to construct a complex solid model using boolean operators such as *union*, *intersection*, and *difference*. Simple geometric shapes such as spheres, cubes, and cones are combined using boolean operators during the procedure and compose a complex model. This procedure is often represented using a hierarchical structure called CSG tree. A function for applying CSG to implicit surfaces was introduced by Ricci [42]. Union $f_{A\cup B}(p)$, intersection $f_{A\cap B}(p)$, and difference $f_{A-B}(p)$ operators between two scalar fields $f_A(p)$ and $f_B(p)$ can be defined as follows:

$$\begin{cases} f_{A\cup B}(p) = max(f_A(p), f_B(p)) \\ f_{A\cap B}(p) = min(f_A(p), f_B(p)) \\ f_{A-B}(p) = min(f_A(p), 2v_{iso} - f_B(p)) \end{cases} \tag{2.4}$$

Examples of these operators are shown in Figure 2.4. One of advantages to use CSG is that topological changes can be handled easily. For example, a hole can be created on an implicit volume using the difference operator with another implicit volume. Nevertheless, CSG creates discontinuity between two scalar fields, which results in undesirable artifacts when blending (Section 2.1.3) is applied. There are several approaches for applying CSG operators while preserving continuity. For example, R-Functions in FRep construct C^n continuous CSG operators [40]. Barthe proposed CSG operators which can preserve G^1 continuous scalar fields [5].

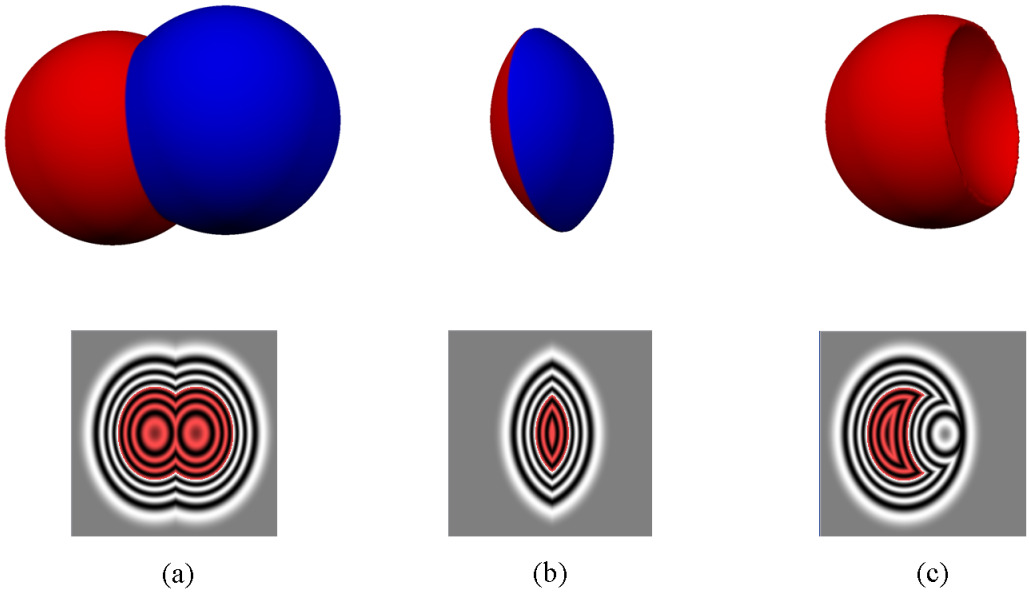


Figure 2.4: Two implicit spheres to which CSG operators are applied and the result scalar fields. Union operator (a), Intersection operator (b), and Difference operator (c).

2.1.3 Blending

Blending is another modeling advantage of implicit surfaces in addition to CSG operators. This operator can create smooth transitions between two objects. It is quite difficult to apply blending to other shape representations such as point based representations and parametric representations while blending can be trivially computed in implicit surface representations. A blending operator $f_{A+B}(p)$ between two scalar

fields $f_A(p)$ and $f_B(p)$ can be simply defined as follows:

$$f_{A+B}(p) = f_A(p) + f_B(p) \tag{2.5}$$

Nevertheless, this operator does not provide any manipulation of blending. To achieve more flexible blending, Ricci introduced another blending operator $f_{A\diamondsuit B}(p)$ by providing a parameter n [42]. This blending is referred to as *Ricci blend* while Equation 2.5 is called *summation blend*. Ricci blend is defined as follows:

$$f_{A\diamondsuit B}(p) = (f_A(p)^n + f_B(p)^n)^{\frac{1}{n}} \tag{2.6}$$

By changing n, the amount of blending can be manipulated (Figure 2.5). As one property, Ricci blend can represent another operator by substituting a particular number for n. For example, $f_{A\diamondsuit B}(p) = f_{A\cup B}(p)$ where $n = \infty$, $f_{A\diamondsuit B}(p) = f_{A\cap B}(p)$ where $n = -\infty$, and $f_{A\diamondsuit B}(p) = f_{A+B}(p)$ where $n = 1$.

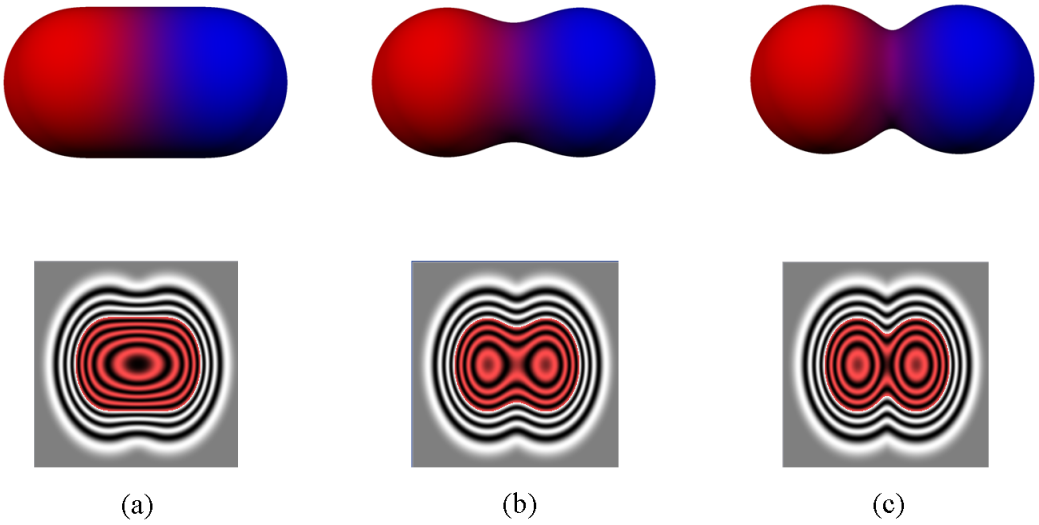


Figure 2.5: Two implicit spheres to which Ricci blend is applied and the result scalar fields. Blending parameter $n = 1$ (a), $n = 2$ (b), and $n = 4$ (c).

2.2 Rendering

Implicit surfaces are rendered by evaluating field values of a scalar field. A set of points which store the iso-value is visualized as the iso-surface. Rendering implicit surfaces are not straightforward compared to polygon meshes because they are not defined

explicitly but by an arbitrary scalar function. That is why a generated scalar field is evaluated at a large number of sampling points to extract rendering data. There are two major rendering methods of implicit surfaces: polygonization (Section 2.2.1) and ray tracing (Section 2.2.2). Implicit surfaces can be rendered in real-time with polygonization and rasterization in standard graphics hardware. On the other hand, ray tracing takes much time in order to generate a more realistic image since various optical effects are computed. In this context, polygonization is useful for interactive implicit modeling, while ray tracing is often utilized when a realistic image is required.

2.2.1 Polygonization

The most common rendering method of implicit surfaces is *polygonization* [63]. Polygonization is a method to extract a set of triangles from a scalar field. They are then rasterized in graphics hardware. This method first constructs uniform cubic cells called *voxels* in the region surrounding the iso-surface. This process is called *voxelization*: a single voxel is found which touches the iso-surface and then new voxels are propagated from the single voxel. This propagation incrementally encloses the iso-surface with new voxels. An example of a voxelized implicit model is shown in Figure 2.6. Next, the generated voxels produce triangles for rasterization. The triangles are formed by connecting the intersections between voxel edges and an iso-surface. The intersections, the points which store the iso-value on the voxel edges, can be found using regula falsi which is a root finding technique. The polygonization quality depends on the voxel resolution (Figure 2.7). If the voxel resolution is low, a small primitive, which does not intersect with voxel vertices, is missed and is not rendered. On the other hand, the high voxel resolution can render implicit surfaces accurately, however the number of triangles increases and much memory is required. Hence, the low resolution is often used for interactive modeling and then the result shape is rendered using high resolution or ray tracing to get a high quality image.

This polygonization method is quite simple and is often used in several applications. Nevertheless, there are some limitations, so a number of extensions have been proposed. For example, the standard polygonization method has several ambiguous corner configurations for a cube, which means triangles cannot be extracted from the cube because several triangulation possibilities exist. To avoid the ambiguities, the cube is subdivided into tetrahedra [7]. Because ambiguous corner configurations do not exist for a tetrahedron, an unique triangulation solution can be found. Another

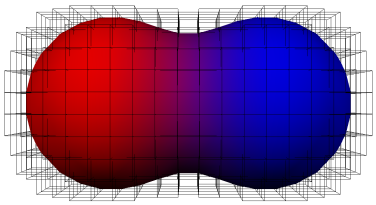
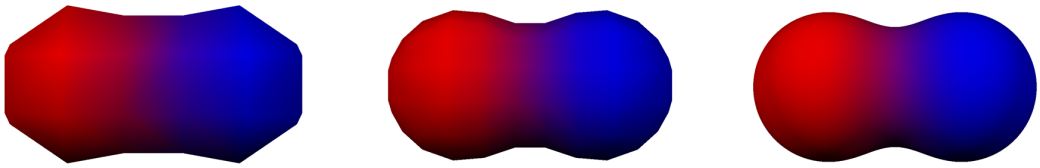


Figure 2.6: Voxelization. A set of voxels are generated around the iso-surface. This figure was visualized by rasterizing the triangles extracted from the voxels.



(a) (b) (c)

Figure 2.7: Polygonization with different voxel resolution. The polygonization quality increases from (a) to (c) as the voxel resolution does.

issue of the standard polygonization is that it produces degenerate triangles, so there are several optimization methods to improve mesh quality. In particular, some optimization algorithms produce high quality mesh with sharp features [61, 28, 35]. CSG operators create sharp edges, so accurate polygonization of sharp features is necessary for implicit modeling.

2.2.2 Ray Tracing

Another method to visualize implicit surfaces is *ray tracing*. In contract to polygonization, ray tracing renders implicit surfaces directly without any conversions, so more accurate images can be produced. Also, it can save memory because triangles do not need to be stored. Nevertheless, the bottleneck of ray tracing is that computational cost is expensive, so it is not suitable for interactive rendering.

The intersection tests for skeletal implicit surfaces are relatively straightforward. Because a scalar function $f(p)$ is defined with a distance function which is generally a low-degree polynomial, the intersection point with a ray $e + td$, where e is the eye and d is the direction of the ray, can be found by simply substituting $e + td$ for p and solving for t. Nevertheless, this function cannot be applied if a scalar function f is defined with a higher order or f is treated as a black box. Therefore, there are

more general techniques of the intersection tests between rays and a scalar field: L-G Surfaces [26] and Sphere Tracing [21]. These two techniques are based on Lipschitz constants. A constant L is called a Lipschitz constant if L bounds the magnitude of the derivative of a scalar function: $L > df/dt$. L-G surface technique finds a Lipschitz constant in a given interval and determines the existence of intersection points in the interval. This process continues by recursively subdividing the interval until intersection points are found. The bounding boxes of an implicit model are used as the initial intervals. Sphere Tracing is also a Lipschitz constant based technique but can deal with wider variety of implicit functions such as discontinuous functions which define CSG operators.

2.3 BlobTree

The *BlobTree* [60] is a hierarchical implicit volume representation to organize skeletal primitives and composition operators such as blending, warping (Section 3.2), and CSG operators. Skeletal primitives are stored at leaves and they are combined at interior nodes using composition operators. Because of a hierarchical structure, the influence of a composition operator can be localized according to the position of the composition operator in the BlobTree. Hence, complex solid models with arbitrary topology can be represented like a CSG tree. Texture mapping [56] and controlled blending techniques [20] were also implemented in the BlobTree. An example BlobTree model is shown in Figure 2.8.

In practice, the BlobTree has been employed by various applications. For example, natural phenomena, such as trees and shells, were modeled using the BlobTree [18]. Although it is quite difficult to model botanical trees due to complex branching structures, they were modeled by hierarchically controlling composition operators in the BlobTree. The resulting models showed high modeling capability of the BlobTree. Another work applied animation to the BlobTree, which is called the BlobTree animation system (BTA) [34]. By adding tracks to primitive and operator nodes, geometric changes of implicit surfaces can be simulated in animation. Since BTA supports a new Precise Contact Modeling (PCM) [19] in the BlobTree, deformations in collision between implicit objects can be simulated. The BlobTree was also employed as an underlying shape representation in the ShapeShope which is a sketch-based modeling system [48] (Section 3.3.2). A BlobTree structure is automatically constructed according to user generated sketching. Because the BlobTree can represent a complex

model, even novice users can create complex models easily with sketching.

The bottleneck of the BlobTree is visualization time. In order to visualize a BlobTree model, the entire BlobTree which consists of a large number of nodes must be traversed. This makes it difficult to visualize a complex BlobTree model at interactive rates. Schmidt et al. addressed this issue and proposed a solution to reduce visualization time for the BlobTree [47]. This technique inserts a cache node into the BlobTree, so the subtree of a cache node does not need to be traversed. de Groot et al. introduced the RaySkip system [15] which provides fast ray tracing of BlobTree model animations by reducing the portion of the rays.

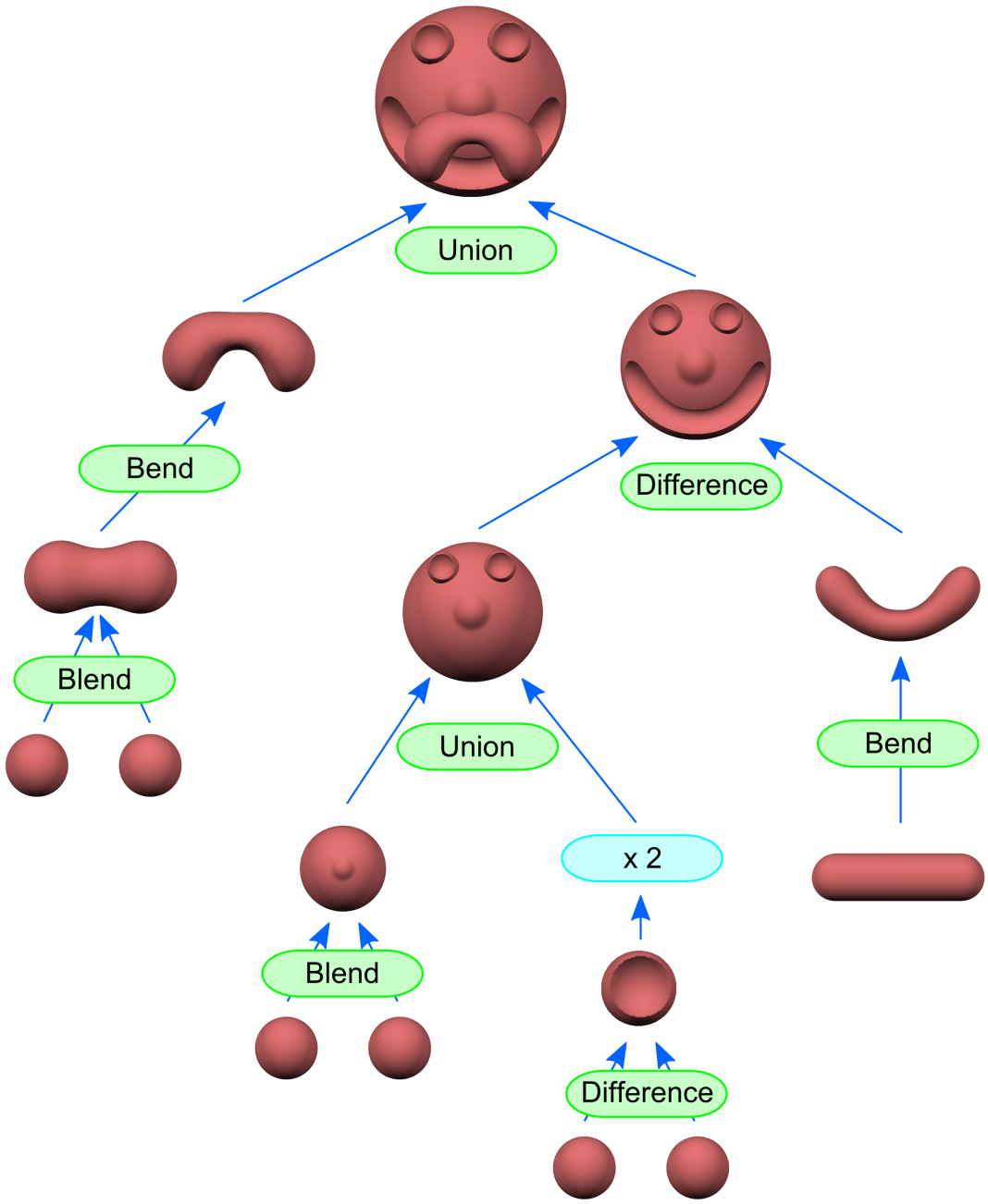


Figure 2.8: A BlobTree model. Spheres and a cylinder are stored at leaves as skeletal primitives. The face is composed of them using composition operators at each interior node. In this BlobTree model, blend, union, difference, and bend nodes exist.

Chapter 3

Background and Previous Work

This chapter introduces the background and previous work related to this research: free-form deformation (FFD) (Section 3.1), deformation for implicit surfaces (Section 3.2) and sketch-based modeling systems (Section 3.3). The goals of this research are to propose an FFD technique for implicit surfaces and also to develop a more efficient interface to manipulate FFD than 3D lattices. Since the fundamentals of implicit surfaces are described in the previous chapter (Chapter 2), this chapter rather focuses on deformation techniques. Section 3.1 gives previous FFD techniques which fall into lattice-based FFD (Section 3.1.1) and FFD without lattices (Section 3.1.2). Several deformation techniques for implicit surfaces are described in Section 3.2. Note that these deformation techniques are not FFD. This chapter also surveys interface designs and techniques for the second goal. Section 3.1 mentions interfaces for FFD as well. Because the solution for an FFD interface in this research is to develop a sketch-based interface, an overview of existing sketch-based modeling systems is provided in Section 3.3.

3.1 Free-Form Deformation

Free-Form Deformation (FFD) is a popular and well researched deformation technique in computer graphics and generally utilizes spatial deformation. Spatial deformation for computer graphics was first introduced by Barr [4]. A conceptual process of spatial deformation is to embed the object in a 3D volume called a *deformation field* which defines a space mapping $d : \mathbb{R}^3 \rightarrow \mathbb{R}^3$, and deform the deformation field. Accordingly, the embedded object is deformed. A survey of spatial deformation was

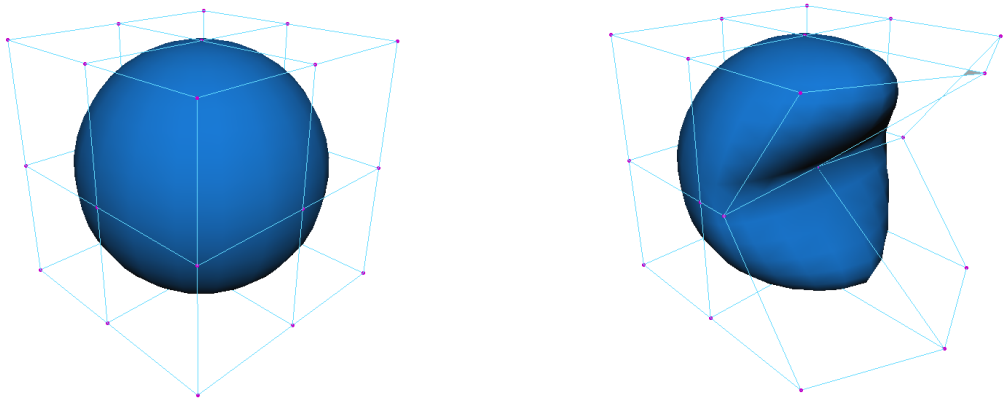
conducted in [17] and it describes more details of spatial deformation from a user prospective. Because spatial deformation can be applied to any surface representation based on point-sampling, it is a well suited technique for FFD. There are two important properties to define FFD:

- An intuitive user interface must be provided.
- Real-time feedback must be given to the user.

FFD is a user-oriented deformation. An intuitive interface lets the user manipulate a deformation as the user wants. Since deformation process is according to user manipulation, the target object (the object being deformed) must react interactively to the user manipulation as feedback. Deformation tools which are satisfied with these two properties allow the user to achieve a desirable deformation.

3.1.1 Lattice-Based FFD

FFD was first introduced by Sederberg and Parry [50]. The tool they proposed is a *lattice* which consists of a 3D uniform grid of control points. The target object is embedded in a lattice and the user deforms the lattice by manipulating the control points (Figure 3.1). The displacement of the control points are interpolated with trivariate Bernstein polynomials and the embedded object is deformed accordingly. The trivariate Bernstein polynomials assure a C^1 continuous deformation field, which is important to create a smooth transition between the deformed part and the undeformed part. Modifications to the original FFD method were proposed, such as the extended free-form deformation EFFD [13] where the lattice can be made cylindrical. MacCracken and Joy [31] used Catmull-Clark subdivision meshes for lattices of arbitrary topology. Ono et al. implemented a system which automatically generates a lattice suitable for a model [36]. The system first constructs a bounding box for the model and then subdivides the box to be fit for the model. Since each subdivision level is defined hierarchically as a set of lattices, it is easy to apply both global and local deformations. For example, a global deformation is applied if the user chooses a upper level lattice. It is not a trivial task to define an appropriate lattice, so this system makes FFD more accessible to the user.



(a) (b)

Figure 3.1: Lattice-based FFD. The embedded sphere is deformed according to the displacement of the control points which compose the lattice.

3.1.2 Various FFD methods

Deformation tools used for FFD are not only a lattice. Some methods utilized radial basis functions (RBF) to define a deformation field [8, 9]. An advantage of using RBF is that a C^2 continuous deformation field can be created without any restriction of control point arrangement, while a lattice of Sederberg's FFD must be a parallelepiped. Therefore, it is a suitable interpolation for non-uniform lattices or even for control points which do not compose a lattice. Nevertheless, the calculation cost is expensive compared to other deformation techniques because a dense matrix must be solved. Compactly supported radial basis functions (CSRBF) [1] [59, 29] were also used which can produce a sparse matrix and speed up computations. Since CSRBF is faster than RBF, C^2 continuous deformation fields can be constructed in real-time [30].

A parametric surface was also used as a FFD tool [12]. The target object is mapped onto a parametric surface, so the deformation is performed along the parametric surface the user manipulates. A technique of texture mapping which maps a 2D texture onto a 3D model is utilized to associate the target object with the parametric surface. The Wires technique [53] is a curve-based deformation technique. The target object is bound to a set of wires and it is deformed by manipulating the wires. The amount of influence of a wire is calculated using an implicit function. The full amount

[1] CSRBF is an RBF method with a finite distance.

of influence is applied at the wire and it smoothly decreases with distance away from the wire. The wires support more detail deformations such as handling wrinkles and creases. Crespin used an implicit field as a deformation tool, which is called an implicit free-form deformation tool (IFFD) [14]. Field values inside the implicit field work as weights of displacements. Although an implicit surface is used as a deformation field, IFFD can only be applied to point based surfaces such as polygon meshes.

Unfortunately none of the methods discussed above 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.

3.2 Deformation for Implicit Surfaces

Deformation for implicit surfaces generally employs spatial deformation and is implemented as a *warp* operator. An example of the warp operator is shown in Figure 3.2. A point $p \in \mathbb{R}^3$ is warped into $w(p) \in \mathbb{R}^3$ by applying a warp function $w : \mathbb{R}^3 \to \mathbb{R}^3$. Then, the field value of the point p is evaluated by applying a continuous scalar function $f : \mathbb{R}^3 \to \mathbb{R}$. Hence, a warped scalar field $f_w(p)$ can be defined as follows:

$$f_w(p) = f \circ w(p) \tag{3.1}$$

The resulting shape is dependent on a warp function w. Therefore, the related work described below is about how to define w. Note that w must be defined as an inverse warp. Since implicit surfaces are defined by an arbitrary scalar function, a forward warp cannot be applied directly as indicated in Section 1.1. More detail discussion about an inverse warp is also given in Section 5.3.2.

Pasko et al. adapted the Barr operations [4] to Frep [40, 39]. Barr operations consist of twist, taper, and bend, and can be applied locally and globally to models. By combining these operations, complex shapes can be created. Barr operations were also implemented for skeletal implicit surfaces [62] and these warp methods were incorporated into the BlobTree system [60] as one of composition operators. In addition to Barr operations, this work also introduced a deformation field to squash or stretch implicit surfaces over time. Such a deformation field is constructed in 3D space and implicit objects deform as going through the field. Precise Contact

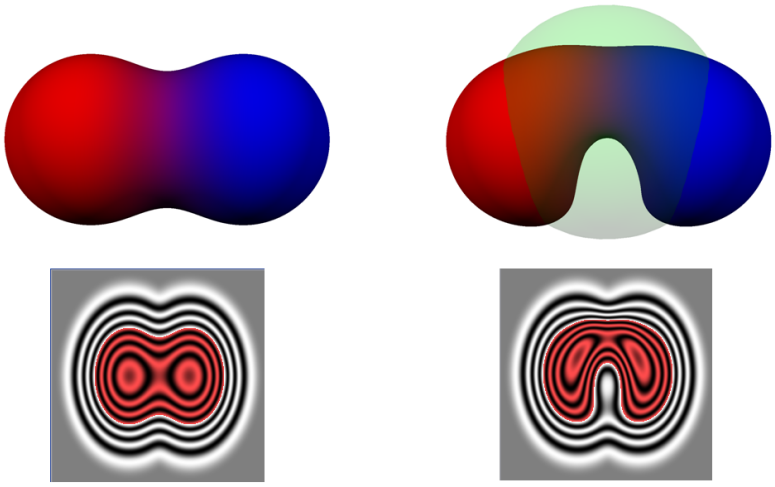


Figure 3.2: An implicit model before (a) and after (b) spatial deformation is applied. The green sphere represents a deformation field and the part inside the green sphere is bent.

Modeling (PCM) [19] was proposed to simulate deformations of implicit objects under collisions. Collisions are basically simulated by splitting the process into 2 steps: collision detection and collision response. PCM introduced a new step called contact modeling between those two steps. During the contact modeling step, the scalar fields of implicit objects are deformed and the contact surface between them is generated. C^1 continuity can be preserved even after collisions. This idea was extended into [37, 10] which discussed local deformations, volume preservation, and blending control for PCM. Schmitt et al. [49] proposed a deformation method for F-rep models [40, 39]. A more flexible deformation is possible than the methods described above by defining another F-rep object as a deformation field. This principle is applied not only to geometry but also attributes. For example, texture, one of the attributes, can also be deformed by following geometric deformation.

The above methods work well for implicit models, but none of them are FFD methods because they are not satisfied with the two important FFD properties (Section 3.1). In this research, a more controllable and user friendly deformation technique is investigated to achieve FFD for implicit surfaces using a sketch-based approach.

3.3 Sketch-Based Modeling

Sketch-based modeling has become a common technique for free-form modeling. The concept of sketch-based modeling is to allow anyone to create a 3D model. Recent modeling tools such as Maya can create impressive 3D models but are very complex user interface offering many facilities requiring a large investment in time to learn how to use the system. Hence, the users are limited to experts of computer graphics design. Sketch-based modeling systems have made 3D modeling more accessible to novice users by utilizing sketching which is an intuitive drawing method for many people. The general sketch-based modeling procedure is that the user first draws a 2D contour with sketching (Figure 3.3a). Then, the system automatically constructs a 3D shape from that sketching (Figure 3.3b).

Figure 3.3: Sketch-based modeling procedure. The user draws a closed stroke on the screen as user input (a). The system then inflates the user stroke into a 3D shape along the vector normal to the screen (b). (c) is the inflated shape from a different angle.

3.3.1 Background

The pioneer of sketch-based modeling systems is *Teddy* introduced by Igarashi et al. [24]. Triangle meshes were employed as an underlying shape representation. Inflation of a sketched 2D contour is based on chordal axis [41]. The width of an inflated 3D shape is proportional to the distance from the chordal axis to the 2D contour. Sketch-based operations such as extrusion, cutting and smoothing operations are supported in Teddy. It also supports a deformation operation. The user draws two strokes and then the system warps the model from one stroke to the other. This work was extended

into several systems. Chameleon [22] is a system to paint a Teddy model. A method to generate smoother meshes for sketch-based modeling systems was implemented by refining original rough meshes [23]. Volume Teddy [38] is a sketch-based modeling system with a binary volumetric representation. Original Teddy is limited to spherical topology but Volume Teddy enabled topological changes, such as a torus and a hollow sphere, by providing internal structures.

Implicit surface representations have also been employed as an underlying shape representation of sketch-based modeling systems. Skeletal implicit surfaces were used by Alexe et al. [2] and Tai et al. [55] to construct a 3D model from user sketching. These systems find skeletons from a sketched 2D contour and construct implicit surfaces based on the skeletons. Other systems used variational implicit surfaces [27, 3]. They sample constraints along a sketched 2D contour and interpolate them with variational interpolation. Variational interpolation can construct implicit surfaces which fit user sketching more precisely but requires expensive computation.

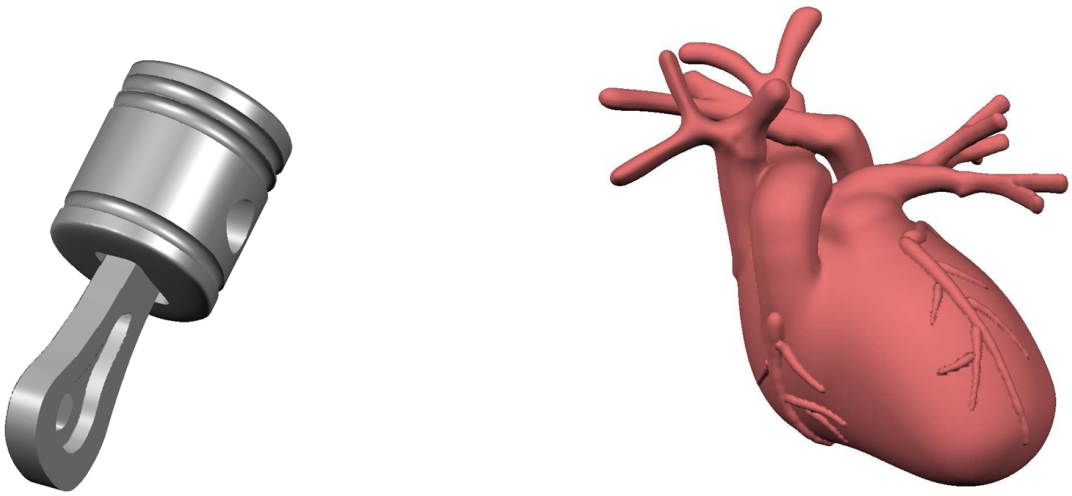
These systems made use of implicit surface properties. Blending can be applied to create smooth transitions between sketched components and the generated surfaces are smooth with arbitrary topology. Nevertheless, complex models cannot be constructed with those systems because it is expensive to evaluate implicit functions and also sharp edges are not supported. Schmidt et al. developed ShapeShop [48] and solved these problems. The more details of ShapeShop are described in Section 3.3.2.

The bottleneck of sketch-based modeling is that it is not suitable for iterative procedure, so detail editing is difficult. The above systems are basically used to create coarse models and the models are refined with another system. In order to do modeling and detail editing on the same system, some researchers have developed direct detail editing after coarse modeling. FiberMesh [32] allows the user to refine a model with 3D curves. The strokes which the user drew remain on the surface and work as handles for detail editing. The user can deform the model by pulling the curves. The curves also work to change a surface type. Sharp or smooth edges are defined on the surface along the curves. Schmidt et al. proposed a sketch-based procedural surface modeling which supports layer-based editing [45]. This allows for linked copy-and-paste and drag-and-drop operations. Based on the layer-based editing, other operations are dynamically combined, such as soft displacements, sharp creases, extrusions along 3D paths, and topological holes and handles.

As described above, sketch-based modeling systems are a powerful tool for free-form modeling. Intuitive manipulations can be achieved by using sketching as user

input. This section has also addressed that a sketch-based interface is well fit for implicit surface representations. Therefore, a sketch-based interface was employed as Taco's user interface. The FFD technique of Taco was implemented as a composition operator within the ShapeShop sketch-based implicit modeling system. Since Taco has been developed based on ShapeShop, the next section discusses ShapeShop more deeply.

3.3.2 ShapeShop



(a) (b)

Figure 3.4: ShapeShop models. The piston model has sharp edges which were created with CSG operators (a). The complex branching structure of the heart model were constructed by blending several sketched simple components (b). Copyright Schmidt et al. Used with permission.

ShapeShop [48] is a sketch-based 3D modeling system which uses implicit surfaces as an underlying shape representation. As described in Section 3.3.1, other sketch-based systems which use implicit surfaces can create smooth surfaces with arbitrary topology but these models are limited to simple shapes. There are two major drawbacks of employing implicit surfaces as an underlying shape representation. The first issue is evaluation time. Evaluating scalar fields is quite expensive, so it is impossible to interactively visualize complex models. The second issue is that sharp edges are not supported. The user can create only smooth models. ShapeShop solved these problems by combining BlobTree [60], free-form BlobTree primitives [46], and a hierarchical spatial caching scheme [47]. The features of ShapeShop can be summarized

as follows:

- ShapeShop enables the construction of complex models because of the BlobTree representations which hierarchically control blending and CSG operators.
- Free-form BlobTree primitives support sharp edges but keep continuity using the operators of [5].
- Using the hierarchical spatial caching scheme, even complex implicit models can be manipulated interactively.

ShapeShop automatically constructs a BlobTree structure according to user input. The user draws a component and chooses one composition operator, such as blending and CSG operators, to associate the component with the BlobTree. This node is then inserted into the top of the BlobTree. Since the BlobTree also stores the construction history of the model, each sketched component can be edited anytime. This allows the user to individually translate a hole or a blended component which composes the model. The models shown in Figure 3.4 are some results created with ShapeShop. These models show higher complexity than the models created with other sketch-based modeling systems.

Since Taco has been developed based on ShapeShop, the screenshot of ShapeShop user interface is shown in Figure 3.5 as the further reference of ShapeShop.

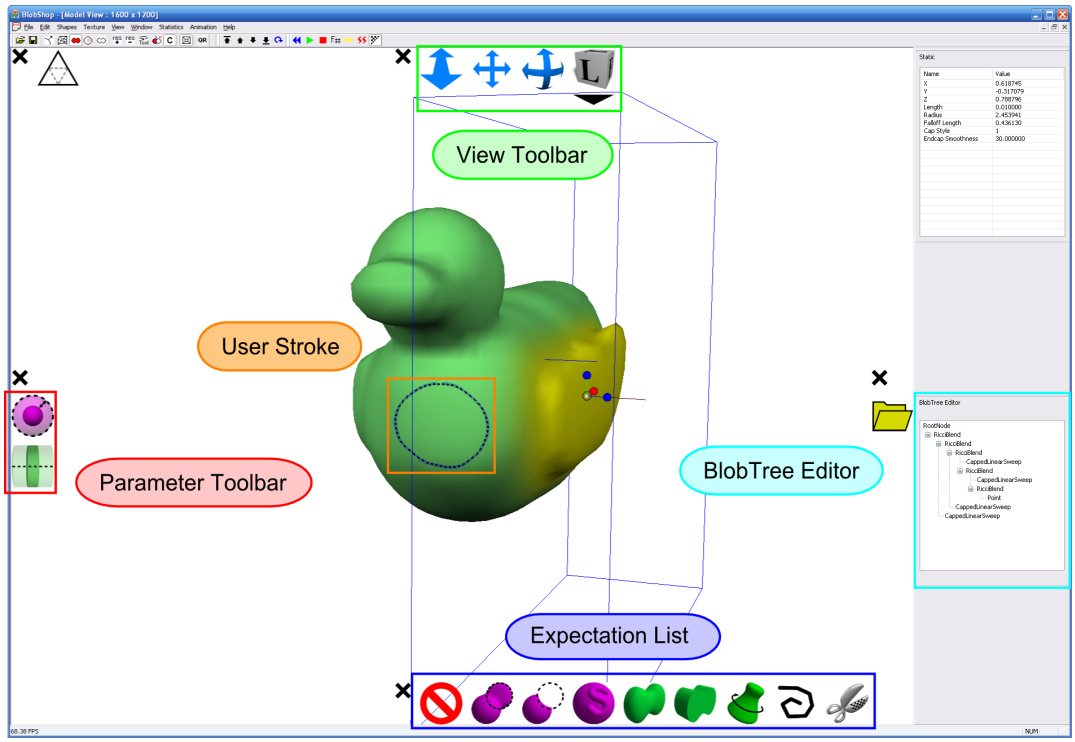


Figure 3.5: ShapeShop user interface. When the user draws a stroke, the expectation list will be displayed on the screen. This list suggests available modeling operations with the drawn stroke. The parameter toolbar provides the sliders for the user to manipulate the parameters of the selected node. The user can control the position of the camera with the view toolbar. The BlobTree is automatically constructed according to the user input, but the user can manually edit the BlobTree of the model with the BlobTree editor if the user desires.

Chapter 4

Deformation Interface

This chapter provides an overview of Taco's deformation interface. One of important factors to design an FFD interface is how to specify the region of interest (ROI). In Section 4.1, ROI is described first and then a novel *volumetric peeling interface* for ROI specification is proposed. Section 4.2 contains the interface design of Taco. The procedure of this interface consists of two passes: the interactive deformation approximation pass and the variational warping pass. Note that the volumetric peeling interface is the extension of a peeling interface proposed in [25] and the peeling interface has been well examined for use in 3D modeling [32].

4.1 Volumetric Peeling Technique

A standard interface problem with local volumetric deformation techniques is specifying the region of interest (ROI). In Taco, as the user manipulates constraint curves, vertices of an underlying volumetric lattice are modified (see Chapter 5 for details). As with previous systems, Taco provides an interactive parameter for controlling the ROI of the curve on the lattice, however the user 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 [25]. In that system, the user interactively deforms 2D curves by directly pulling on them. The further the user pulls the curve from an initial point, the larger the region of interest along the curve (here determined by arc-length). [32] includes a similar interface for 3D curve manipulation and conducted user study to verify the

effectiveness of that system. A similar interface is utilized in Taco, but instead of peeling an ROI on the curve, the user peels the volumetric ROI of the deformation specified by the curve.

When the user first creates a deformation, the ROI starts out small. As the user 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 user pulls the curve sufficiently far away, the whole surface is deformed (Figure 4.1). The deformation region is indicated by changing the color, from green to yellow. Hence, the effect for the user 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 user 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 users will inevitably demand ROI control, the system provides a slider which modulates the ROI growth rate, which loosely corresponds to the elastic stiffness or rigidity of the material (Figure 4.2).

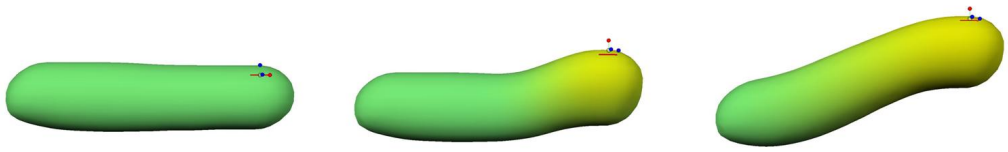


Figure 4.1: A peeling interface. Depending on how far the user pulls the curve, the deformation region is determined.

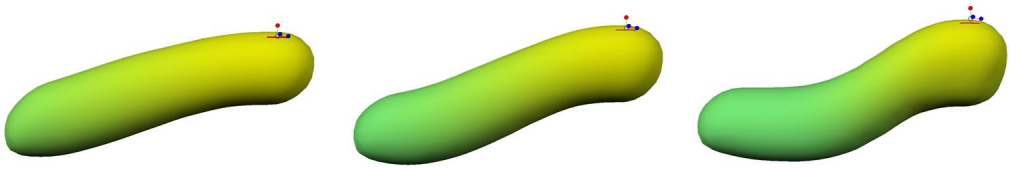


Figure 4.2: The rigidity of the surface can also be manipulated by changing the growth rate.

4.2 Interface Design

This deformation tool is developed within the ShapeShop sketch-based modeling system [48]. More details of ShapeShop are given in Section 3.3.2. The user first creates a BlobTree model with sketch-based operations which ShapeShop provides. FFD can be then applied to this model with the deformation tool. It is implemented as a composition operator and hence its influence is localized according to the position of the deformation node in the BlobTree. Since the BlobTree also stores the construction history of the model, the user can edit or delete the deformation node anytime.

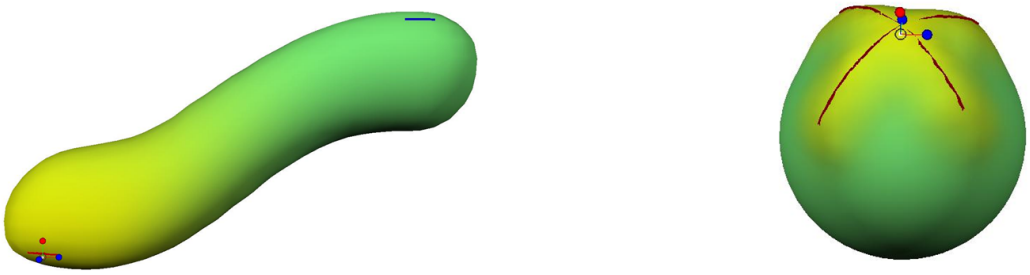
There are two passes in this deformation procedure: the interactive deformation approximation pass and the variational warping pass. The user interactively deforms the model with a volumetric peeling interface in the first pass. This interactive interface is achieved at the cost of a scalar field. Although the iso-surface of the model is deformed for visual feedback in real-time, the scalar field itself is changed in way that makes it behave poorly for further modeling such as CSG and blending operations. In the second pass, therefore, the user deforms the scalar field by applying variational warping [8]. Variational warping cannot be applied at interactive rates but the resulting scalar field is suitable for further modeling. This scalar field problem is discussed in Section 5.5

4.2.1 Interactive Deformation Approximation

To begin a deformation, the user 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 expectation list (Figure 4.7). 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 user manipulates the embedded curve using 3D transformation tools as described in Section 4.1. The user translates the curve and pulls the surface along with it, similar to Wires [53]. When a curve is selected, the rigidity slider is added to the parameter toolbar (Figure 4.7), enabling the rigidity of the surface to be controlled.

Conceptually, the curve is treated as a constraint in the underlying deformation. The user 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. All other curves are then projected onto the new surface (Figure 4.3).



(a) (b)

Figure 4.3: The user can pull the model from several positions with multiple curves.

The user can also turn on or off the visualization of the deformation grid (see Section 5.2 for details of the deformation grid) and the deformation curves. The default status is that the visualization of the deformation grid is off and the visualization of the deformation curves is on. The deformation grid is invisible in the default status because the user does not need to directly interact with it. For the user who wants to see the internal deformation grid, however, the system can present the visualization of it (Figure 4.4b). The deformation curves are visualized as handles to manipulate deformations, however the switch to turn off the deformation curves is provided so that the user can see the deformation results without the curves (Figure 4.4c).

4.2.2 Variational Warping

Once deformation is achieved, the user can apply variational warping to the model in order to re-construct the scalar field. This pass must be executed after interactive deformation because the resulting scalar field generated in the interactive deformation approximation pass is not usable for further modeling. If the user applies another operator such as blending and CSG without variational warping, undesirable visual artifacts are produced.

Variational warping can be applied at any time by crossing the button with a stroke. The button for variational warping is displayed in the parameter toolbar

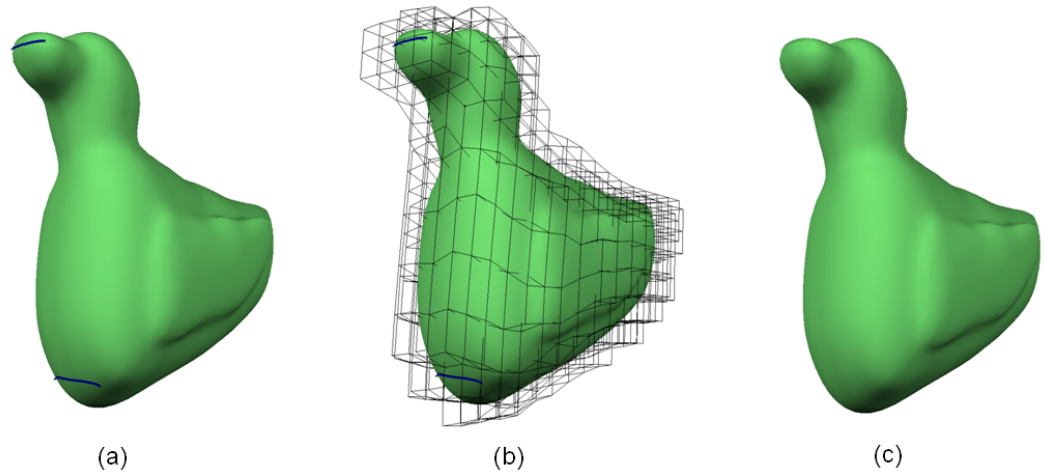


Figure 4.4: Grid and curve visualization. (a) is the duck model deformed with multiple curves. The grid which defines the deformation of the duck can be visualized (b). Also, the user can make the curves invisible to see the final deformation result (c).

when a curve or a deformation node is selected (Figure 4.7). Since variational warping is computationally expensive for interactive use, the re-constructed scalar field is stored in a cache node of the BlobTree [47] for further modeling. This caching schema allows for evaluating the re-constructed scalar field in real-time, so the user can interactively blend the model with another implicit primitive or apply a CSG operator to the model. An example model to which blending and CSG operators are applied after variational warping is shown in Figure 4.5. When the user manipulates the curve again, the system automatically goes back to the interactive deformation approximation pass. The model can be edited iteratively, and variational warping can be applied by crossing the button again.

4.3 Discussion

A peeling technique allows the user to intuitively deform a model without manual ROI specification. In particular, volumetric peeling transmits peeling effects through a 3D model (volume), so the deformed model behaves like a real object. For example, the user can stretch or squash a model by pulling or pushing it via the deformation curve as shown in Figure 4.6. This imitates real object behaviors as it is also stretched or squashed by this manipulation. Also, the growth rate of peeling effects is corresponding to the rigidity of the model. By changing the growth rate, the user can simulate various elastic materials.

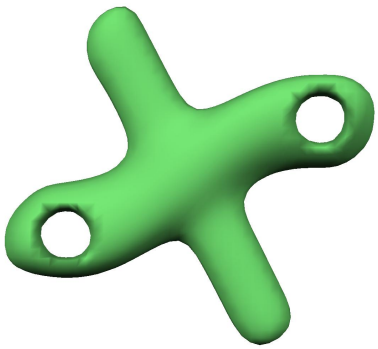


Figure 4.5: Blending and CSG operators can be performed interactively after variational warping. This model is the one shown in Figure 4.3a to which Blending and CSG operators were applied. This model shows that Blending and CSG operators work after variational warping is applied because this model has smooth transitions and sharp edges.

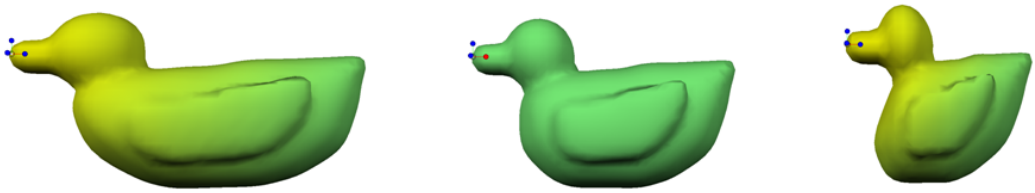


Figure 4.6: Stretched model (a). Original model (b). Squashed model (c). The user can stretch or squash the model just by pulling or pushing the deformation curve the user drew.

Although the volumetric peeling interface presented is similar to the interface of FiberMesh [32], deformation behaviors in the FiberMesh system are different from the volumetric peeling interface. This is because the interior of the deformed model of FiberMesh is not taken into account. In the FiberMesh system, the user first edits the shapes of the deformation curves with peeling effects and then the model is deformed according to the deformation curves. Therefore, peeling effects are restricted in deformation curves, so it is difficult to achieve the deformation results shown in Figure 4.6.

Nevertheless, this is also the strength of FiberMesh and the limitation of the volumetric peeling interface. FiberMesh can localize deformation along deformation curves by restricting peeling effects even if the user peels a deformation curve far away. On the other hand, the volumetric peeling interface shifts from local deformation to

global deformation according to the amount of peeling. This methodology limits the effects of achievable local deformation. A possible approach to have local effects in the volumetric peeling interface is to add a curve editing function like FiberMesh. In the current Taco system, the user can pull a entire deformation curve but cannot change the shape of the curve. By enabling deformation curves to be edited, the volumetric peeling interface would be able to have both of local and global peeling effects.

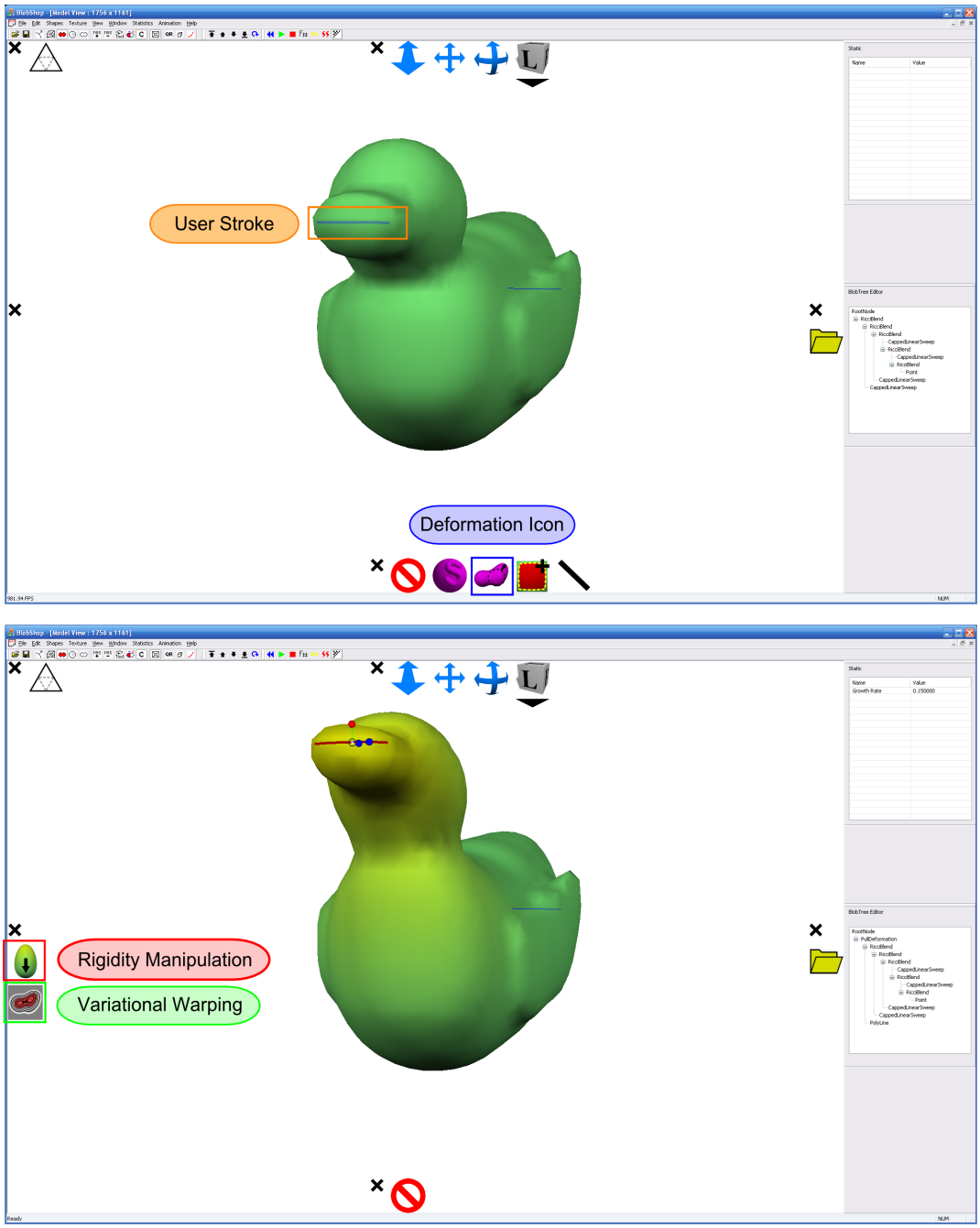


Figure 4.7: Deformation interface. When the user draws a curve on the surface, the deformation icon will be added to the expectation list. The user can manipulate the curve as a deformation handle by selecting the deformation icon. The rigidity slider and the icon for variational warping are provided in the parameter toolbar.

Chapter 5

Deformation Algorithm

This chapter provides the details of the deformation algorithm which is the core of Taco. The overview of the algorithm is given in Section 5.1 and the rest of sections are the descriptions of the algorithm following the overview.

5.1 Overview

The framework of the deformation algorithm is shown in Figure 5.1. It consists of three passes: *voxelization*, *interactive deformation approximation*, and *scalar field re-construction*. It can be summarized as follows:

1. **Voxelization**: Pre-computing pass. Voxelization involves approximating the model with a grid of cubic voxels and associating them with the user-drawn curve.

2. **Interactive Deformation Approximation**: This part gives the user visual feedback of the deformation in real-time according to the user input. The user's actions displace the vertices of the grid, and the deformation is inferred from these displacements. Interactive deformation defines a new approximate scalar field based on the voxel grid, visually approximating the effect of the deformation. The interface for this user input is described in Chapter 4.

3. **Scalar Field Re-Construction**: Post-computing pass. The deformation is re-calculated with variational warping to re-construct the deformed field after interaction. Variational warping computes a spatial deformation field based on the displacements, warping the implicit model.

Computing the deformation field in one pass and avoiding the interactive approximation are an ideal approach, however variational warping is too slow for interactive use. Therefore, this work exploits the speed of the approximation and then either lets the user explicitly compute the variational warp, or takes advantage of idle moments to do so, similar to the approach proposed in [9]. Since the interactive deformation approximation pass focuses on interactive visual feedback, only the iso-surface is taken into account and the scalar field itself loses the properties of an implicit surface. By applying variational warping in the scalar field re-construction pass, the scalar field can be deformed properly with preserving the properties. The resulting scalar fields created in both of passes are evaluated in Section 5.5.

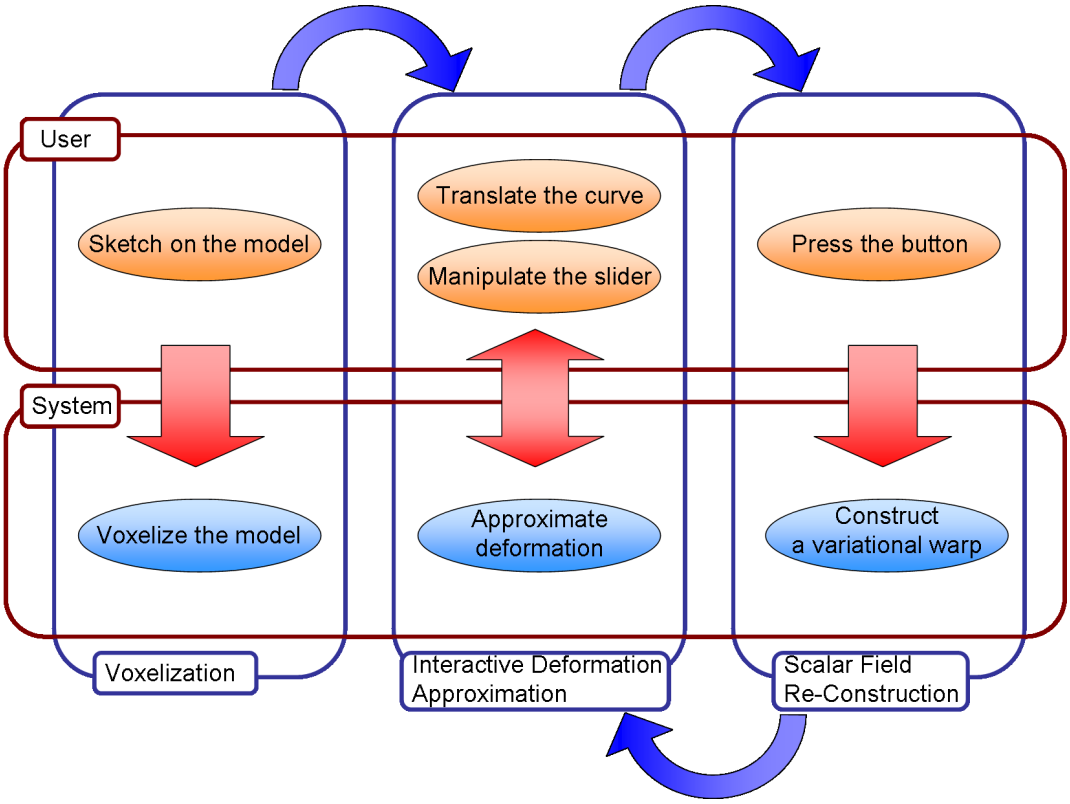


Figure 5.1: Framework of the deformation algorithm. The first row represents the user's action and the second row represents the system's action. The three steps of the deformation algorithm are shown in columns. The up-down arrow in the column of interactive deformation approximation represents the interaction between the user and the system. The system returns interactive feedback of approximated deformation appearance to the user according to the user input. Although the scalar field re-construction pass is a post-computing pass, the user can freely let the system go back to the interactive deformation approximation pass for iterative deformations.

5.2 Voxelization

The user draws a curve on the surface of the model to define the region of the deformation (Figure 5.2). A set of voxels are then created in the region surrounding the user-drawn curve. This is referred to as the *deformation grid* in this work. In ShapeShop, the implicit model is visualized using a polygon mesh extracted from a similar uniform voxelization [63]. This voxelization method is also described in Section 2.2.1. Hence, the fidelity of the surface visualization is limited by the voxel resolution. The deformation grid is also created in the same manner. Nevertheless, the deformation grid is independent of polygonization, so the user can use arbitrary grid resolution without affecting the 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.

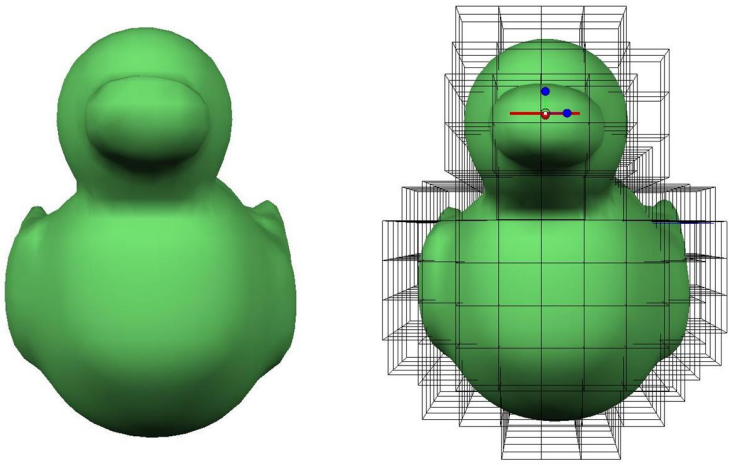


Figure 5.2: The surface of the model is automatically voxelized right after sketching.

5.2.1 Curve Association with the Deformation Grid

The user-drawn 3D curve is embedded in the voxel grid after voxelization. For each curve vertex, the nearest grid vertex is found - the set of all such grid vertices are the *handle vertices* (Figure 5.3). When the user translates the curve, the same translation is applied to the handle vertices. This translation is then propagated to each surrounding grid vertex, with a smoothly decreasing spatial influence based on the distance to the handle (Section 5.3.1).

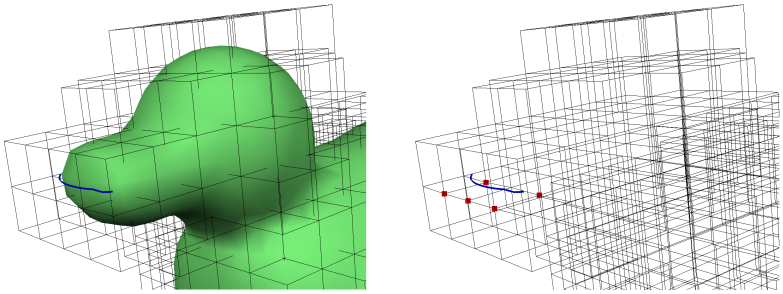


Figure 5.3: The user-drawn 3D curve is associated with the deformation grid. Since a curve consists of several vertices, the system searches for the nearest grid vertex from each curve vertex. The red points on the right figure represent handle vertices.

To define the smoothly decreasing weight on the interactive translation, the distance from each grid vertex to the handle must be computed. Dijkstra's algorithm [16] is used to approximate these distances. This algorithm was also used in a similar way to approximate exponential map for texturing [44]. The goal is to compute an approximate distance d for each vertex v with index (i, j, k). The handle distances are initialized to $d = 0$, and then distances are sequentially propagated 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 d at v is approximated by

$$d = d_n + s_{voxel}\sqrt{|i - i_n| + |j - j_n| + |k - k_n|} \tag{5.1}$$

where d_n is the distance stored in a neighbour vertex v_n with index (i_n, j_n, k_n), and s_{voxel} represents the size of a voxel (Figure 5.4). The set of possible distances are constant and can be pre-computed.

After this process, each vertex stores the approximate distance from the handle vertices (Figure 5.5). Note that if the user has drawn multiple curves, this computation is done independently for each curve. Each vertex stores the separate shortest-distance for every curve in the deformation (Figure 5.6).

5.3 Interactive Deformation Approximation

During interactive manipulation, the system takes the curve translation as input and attempts to approximate the appearance of the deformed surface in real-time. The goal is to give interactive feedback to the user. No time needs to be spent to preserve

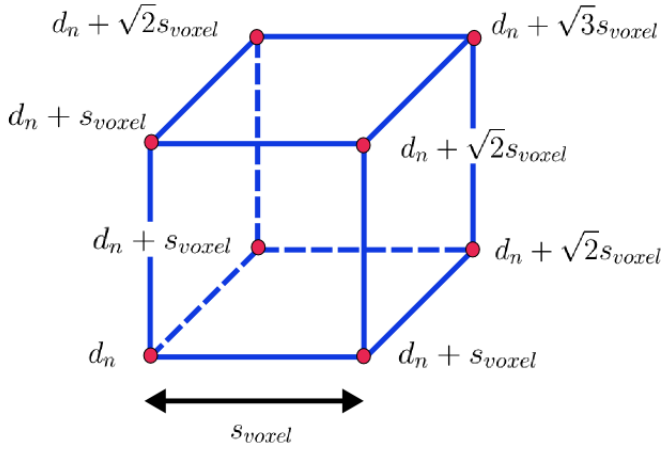


Figure 5.4: An example of distance approximation. The distance from the handle vertex is calculated according to the distance from the neighbour vertex d_n.

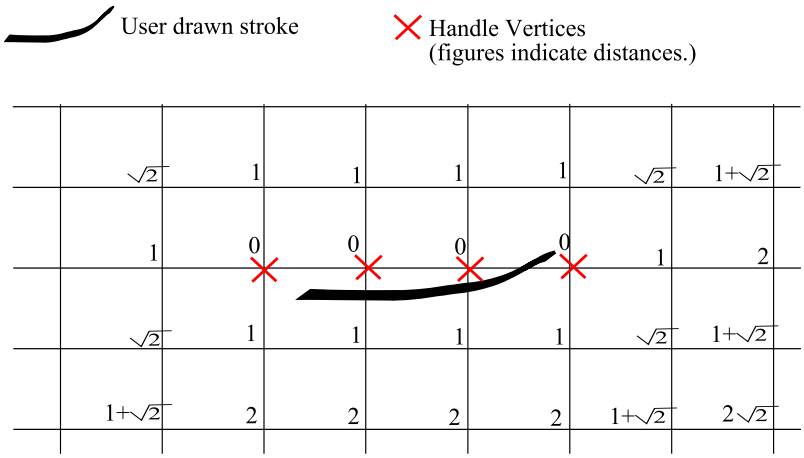


Figure 5.5: Once the curve has been drawn, the handle vertices are found. Each vertex stores the approximate shortest distance to the handle, found using Dijkstra's algorithm.

the field for subsequent use as that will be done in a separate pass (Section 5.4).

5.3.1 Voxel Translation

When the system receives the translation $T \in \mathbb{R}^3$ and the rigidity parameter $r > 0$ from the user, the amount of translation T_v of a vertex v is calculated as:

$$T_v = \begin{cases} g(\frac{d}{r\|T\|}) \cdot T & \|T\| \neq 0 \\ 0 & \|T\| = 0 \end{cases} \tag{5.2}$$

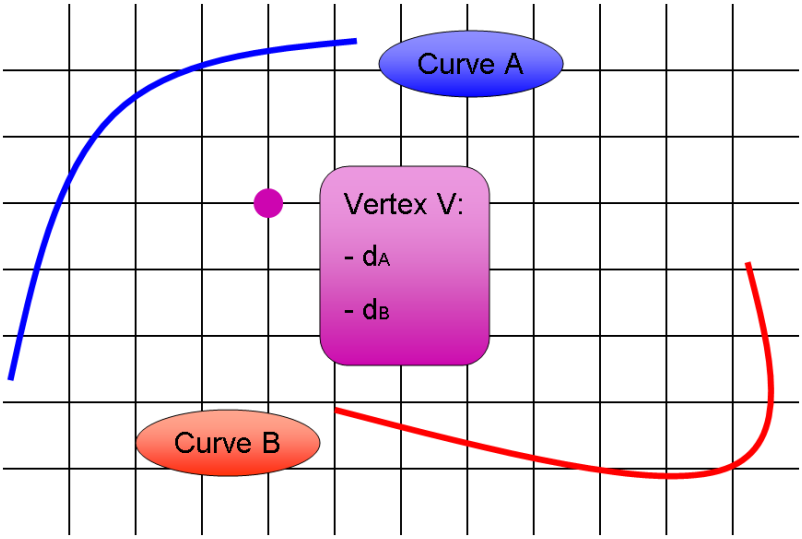


Figure 5.6: The vertex V stores two shortest distances. d_A and d_B are the shortest distances from the curve A and the curve B, respectively.

$$g(x) = \begin{cases} (1-x^2)^3 & x \leq 1 \\ 0 & x > 1 \end{cases} \tag{5.3}$$

where d is the distance at v and $g(x)$ is the *Wyvill* function [7] 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 5.7). The rigidity parameter r controls the growth rate of ROI. The higher the value of r, the more the vertices are translated. This effect makes the deformation behavior more rigid as can be seen in Figure 4.2. At handle vertices, $d = 0$ and the full translation is applied regardless of the value of r.

If multiple curves are drawn by the user, 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:

$$T_v = \sum_{i=1}^{n} g(\frac{d_i}{r\|T_{c_i}\|}) \cdot T_{c_i} \tag{5.4}$$

Currently, the Wyvill function is used to calculate the amount of translation because it provides a smooth transition between the translated part and the non-translated part. Nevertheless, the function for calculating the amount of translation is not limited to the Wyvill function. For example, if a function which changes sharply is adapted, the deformation behavior will also be sharp like the function.

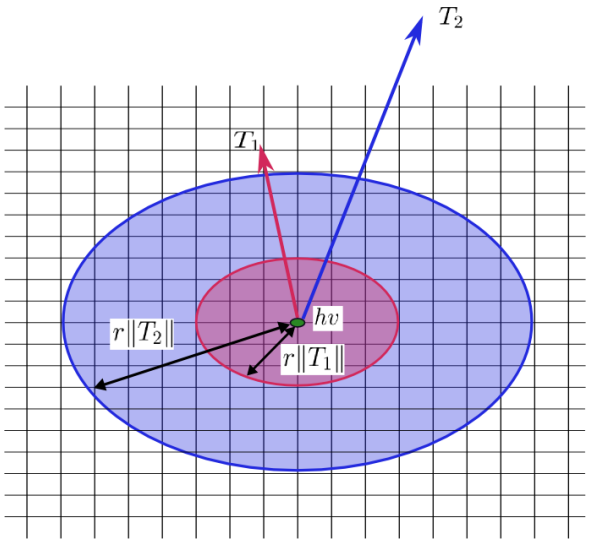


Figure 5.7: When the handle vertex (hv) is translated with T_1, the vertices in the red circle are influenced. The same thing happens with T_2.

Experimentation with other functions is left for future work.

5.3.2 Deformation Field Construction

A deformation field D is now computed 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 is in D is defined:

$$q = p + D(p) \tag{5.5}$$

This deformation can be applied to point based representations such as a polygon mesh as shown in Figure 5.8a because a set of points $p \in \mathbb{R}^3$ are explicitly defined to represent a surface. On the other hand, implicit surfaces are not defined as a set of points, but by an arbitrary scalar function f. Since a deformation field D takes a point $p \in \mathbb{R}^3$ as input, D cannot be applied to a scalar function f. To evaluate an implicit surface, it is necessary to calculate the field values at a large number of sample points. Instead of deforming f, therefore, the sample points are displaced to the scalar field by applying an inverse deformation. In Figure 5.8b, when a point q is evaluated, q is displaced to p with the inverse deformation field $\widetilde{D}$ and then $f(p)$ is returned as the field value of q. Hence, deformation of an implicit surface can be defined by constructing an inverse deformation field $\widetilde{D}$ which approximates D^{-1}.

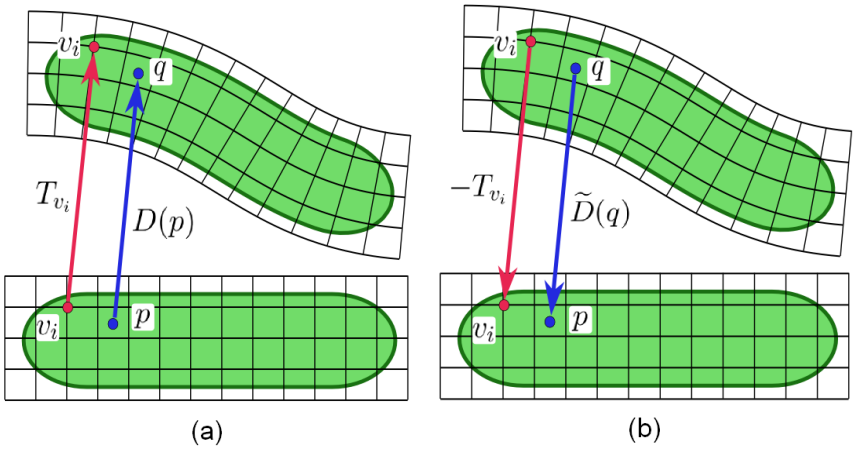


Figure 5.8: Deformation for point based surface representations can be achieved with (a), however inverse deformation is required to deform an implicit surface (b). v_i is a vertex of the deformation grid.

$\widetilde{D}$ is constructed by interpolating the inverse translations of vertices $-T_{v_i}$ using the Wyvill function (Equation 5.3). Suppose that every vertex is an implicit point primitive bounded by R. Two times the size of a voxel is used as R in Taco. When q is given, the set of vertices which influence q are found. Since the complexity of the process will be $O(m \cdot n)$ (where m is the number of the grid vertices and n is the number of the evaluated sample points) with a brute force search, a uniform spatial subdivision search structure is used to efficiently look for the set of vertices which influence q.

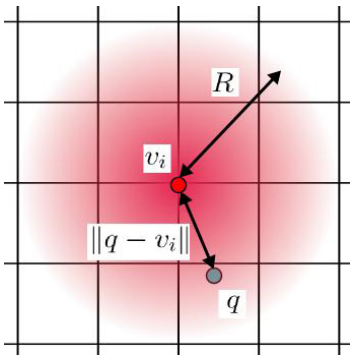


Figure 5.9: q is influenced by vertices v_i. The amount of the influence is calculated with the Wyvill function. The amount is used as the weight for the interpolation.

If a vertex v_i influences q, the field value at q is calculated as the weight using the Wyvill function: $g(\frac{\|q-v_i\|}{R})$ (Figure 5.9). The inverse translation $-T_{v_i}$ is weighted

by the field value and then summed. The final result is divided by the sum of the field values, sum. The case where q is outside $\widetilde{D}$ (Figure 5.10) must also be handled. There $\widetilde{D}(q)$ cannot be calculated as $sum = 0$, so 0 is returned as the field value. Hence, the approximated scalar field $f_{A'}$ at q is defined as:

$$f_{A'}(q) = \begin{cases} f_A(q + \widetilde{D}(q)) & sum > 0 \\ 0 & sum = 0 \end{cases} \tag{5.6}$$

$$\widetilde{D}(q) = -\frac{\sum_{i=1}^{m} g(\frac{\|q - v_i\|}{R}) \cdot T_{v_i}}{sum} \tag{5.7}$$

$$sum = \sum_{i=1}^{m} g(\frac{\|q - v_i\|}{R}) \tag{5.8}$$

where f_A is the scalar field before deformation and m is the number of grid vertices which influence q. Here again $g(x)$ is used as a smooth blending function. Since $f_{A'}(q)$ is 0 outside the voxel grid, the field has discontinuity (Figure 5.11b), 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 is described in the next section.

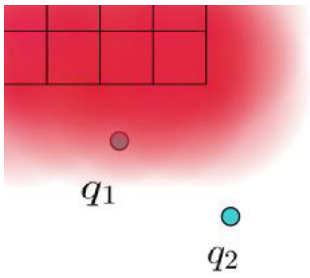


Figure 5.10: The field value of q_1 is calculated with the Wyvill function. Nevertheless, the field value of q_2 cannot be calculated because no vertex influences q_2. In this case, the system simply returns 0 as the field value of q_2.

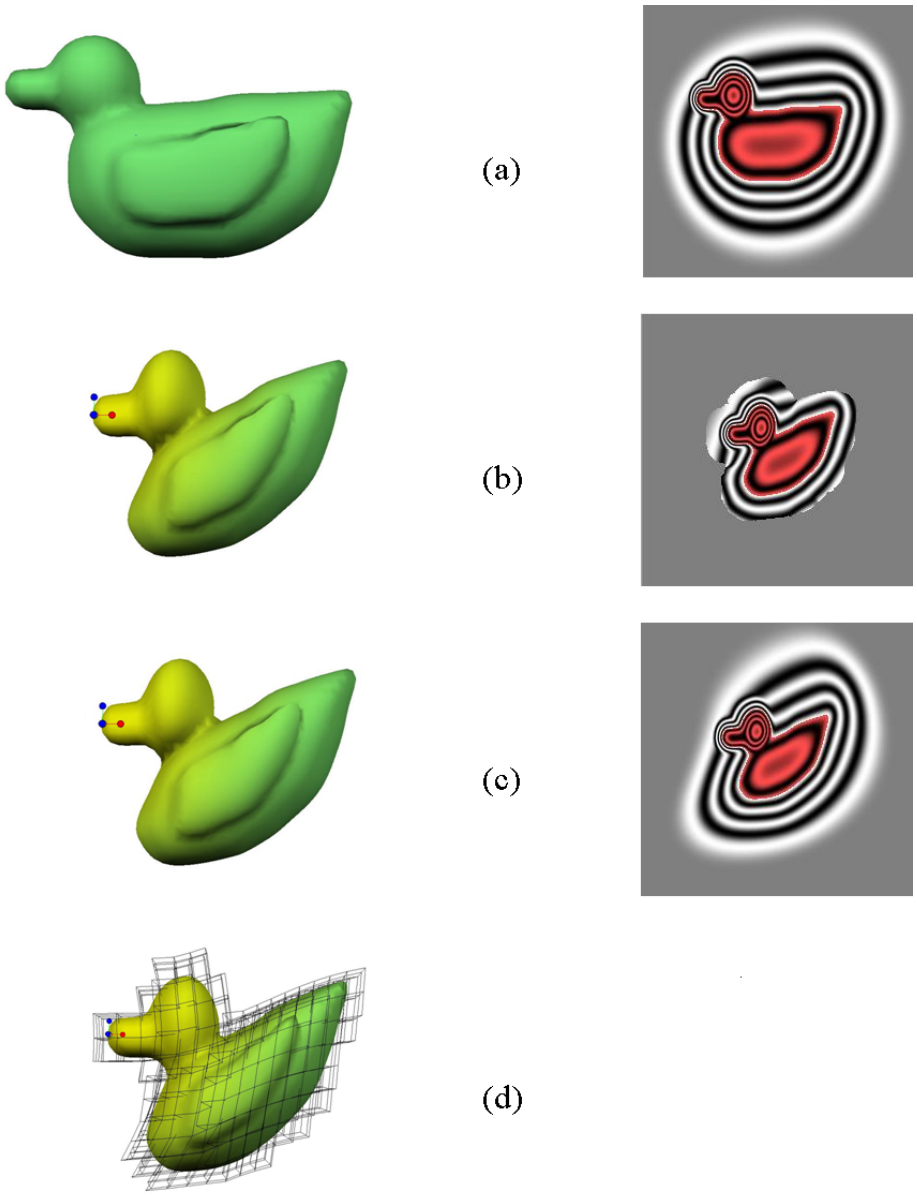


Figure 5.11: An example deformation procedure with the duck shown in Figure 5.2. The first column shows the results of the visualized implicit model. The second column shows the scalar field images which were sampled from a slice along a plane. The original shape of the duck is shown in (a). By pushing the bill of the duck, the duck is squashed interactively in interactive deformation (b), however the scalar field outside of the deformation field is lost. Once the model has been deformed, the scalar field of the duck is re-calculated with variational warping (c). A good scalar field can be preserved with variational warping and the field is cached for further modeling. The deformed voxels are also shown in (d).

5.4 Scalar Field Re-construction

When deformation is achieved, the scalar field of the model is re-constructed for further modeling. As described in Section 5.3, the purpose of the interactive deformation approximation pass is to give the user real-time feedback for the deformation. Interactive feedback is achieved at the cost of producing a broken field shown in Figure 5.11b. In this section, two methods for scalar field re-construction after interactive deformation approximation are described. Both of them use variational interpolation which has to solve a large matrix, so it is computationally too expensive for interactive use. Therefore, the resulting scalar field is stored in a cache node of the BlobTree [47] after the calculation. Since variational interpolation is not required after caching, the resulting scalar field can be evaluated in real-time and the user can apply further modeling operators interactively.

5.4.1 Variational Construction

Variational construction is a method based on the implicit sweep technique presented in [46]. In this technique, a 2D scalar field is constructed using variational interpolation and a 3D implicit surface is generated by sweeping the 2D scalar field. In the scalar field re-construction pass, the 2D scalar field construction method is adapted after deformation. Variational construction consists of 2 steps. First, a pseudo-distance field is constructed by interpolating the values stored at the vertices of the deformation grid using variational interpolation. Since variational interpolation globally interpolates all constraints, the pseudo-distance field is not bounded. Therefore, the pseudo-distance field is then bounded using the Wyvill function (Equation 5.3).

When the deformation grid is constructed inside the iso-surface, every vertex of the deformation grid stores $f_{\hat{A}}(p) = g^{-1} \circ f_A(p)$, where g^{-1} is the inverse function of the Wyvill function, f_A is the scalar field of the model before deformation and p is the 3D point at the vertex. Here the vertex stores $f_{\hat{A}}(p)$ instead of $f_A(p)$ because the resulting pseudo-distance field will be bounded using the Wyvill function. After interaction with the deformation grid via deformation curves (the vertices of the deformation grid are translated: $p \rightarrow q$), the stored $f_{\hat{A}}(p)$ of every vertex at q is interpolated using variational interpolation to generate a pseudo-distance field $f_{\hat{A}'}$ and then it is bounded with the Wyvill function. The re-constructed scalar field $f_{A'}$

at q can be defined as follows:

$$f_{A'}(q) = g \circ f_{\hat{A}'}(q) \tag{5.9}$$

Variational construction can construct a good scalar field, however many sample points are required. Also, the shape detail of the model is often lost as can be seen in Figure 5.12. This method is not warping but constructing a scalar field based on the sampling points (vertices of the deformation grid). The accuracy of re-construction strongly relies on the number of sampling points. Therefore, if a complex model is deformed, a high grid resolution is required to avoid losing the shape. On the other hand, variational warping (Section 5.4.2) does not limit the grid resolution, so shape detail is not lost with any grid resolution. In conclusion, variational construction is not suitable as a scalar field re-construction method.

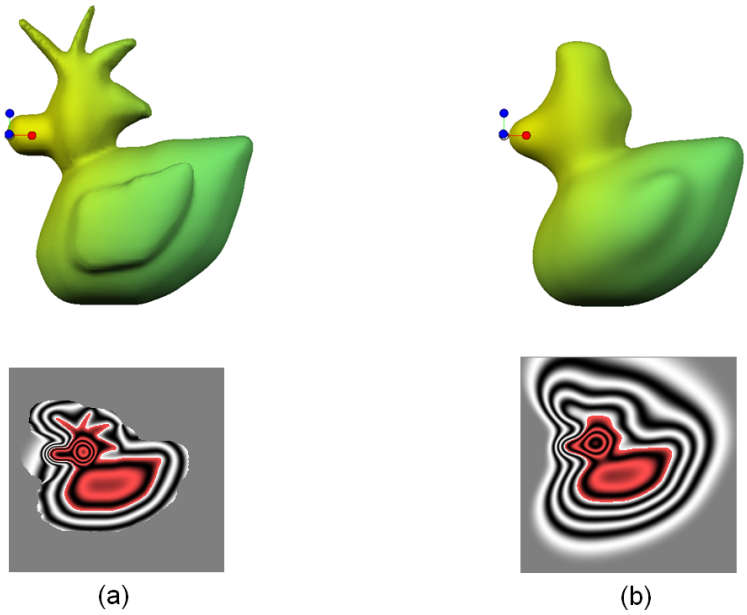


Figure 5.12: Variational construction result. (a) shows the deformation during interactive deformation approximation and (b) is the result after applying variational construction. As can be seen, the crest and feather of the model are lost due to the grid resolution. The details of the model can not be preserved. The upper scalar field is also not constructed well because of the same reason.

5.4.2 Variational Warping

Variational warping is the warp method described in [8]. In [8], polygon meshes are deformed by constraining the surface with variational interpolation. Since only constraining the surface is not enough to preserve a good scalar field, how to adapt this method for implicit surface deformation is described in this section.

The basic approach is same as interactive deformation approximation but an inverse deformation field $\widetilde{D}$ is calculated using variational interpolation. In the interactive deformation approximation pass, the influence of $\widetilde{D}$ is limited inside the deformation grid, so the deformed scalar field $f_{A'}$ is broken (Figure 5.11b). On the other hand, $\widetilde{D}$ computed as a variational warp has the global influence since variational interpolation guarantees to globally interpolate all constraints with C^2 continuity. Constraints outside the iso-surface are not put in Taco, so a complicated scalar field cannot be preserved. But the result is suitable to apply further blending and CSG operators (Figure 5.11c). The re-constructed scalar field $f_{A'}$ at q can be calculated as follows:

$$f_{A'}(q) = f_A(q + \widetilde{D}(q)) \tag{5.10}$$

$$\widetilde{D}(q) = \sum_{i=1}^{m} w_i \varphi(\|q - v_i\|) + P(q) \tag{5.11}$$

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 5.11 at each known solution $\widetilde{D}(v_i) = -T_{v_i}$.

As described in Section 5.4.1, variational warping is not limited in the grid resolution (the number of constraints). Since the grid resolution just limits the frequency of the warp, the shape detail of the model can be preserved with any grid resolution as shown in Figure 5.13.

Essentially, variational warping uses a different interpolation technique from the interactive deformation approximation pass, so the resulting shapes are slightly different. For example, the crest shape of the model shown in Figure 5.13 slightly changes from (a) to (b) although this difference is very tiny. In computer graphics, however, a 3D model is created interactively by the modeler with a low resolution mesh, and then it is rendered by applying a high resolution mesh or ray tracing in order to

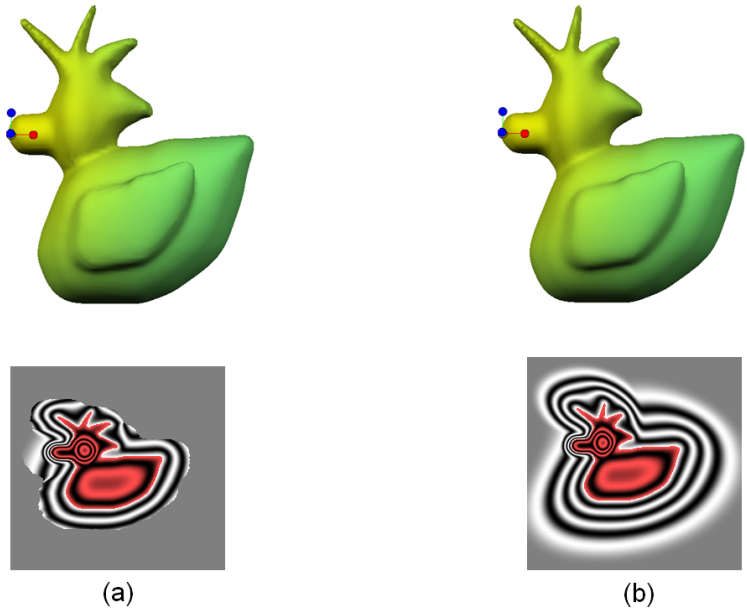


Figure 5.13: Variational warping result. (b) is the result after applying variational warping. Although the same grid resolution as Figure 5.12 is used, the details of the model is preserved. Also, a better scalar field is constructed compared to the scalar field in Figure 5.12.

generate a good quality image. Splitting the deformation process into the interactive deformation approximation pass and the scalar field re-construction pass in Taco is the same concept, so the resulting shape difference is acceptable.

5.5 Scalar Field Evaluation

In this section, scalar fields created in the interactive deformation approximation pass (Section 5.3) and the scalar field re-construction pass (Section 5.4) are evaluated. A deformed scalar field must preserve certain properties of an implicit surface such as blending, CSG, and guaranteed continuity. Scalar field evaluation is conducted by applying these modeling properties to scalar fields created in both of passes and observing the results.

In the interactive deformation approximation pass, a deformation field is computed by interpolating the vertices of the deformation grid. The deformation field is constrained to lie within a finite distance from the deformation grid. Since the deformation grid is constructed around the iso-surface, the entire scalar field cannot be contained in the deformation field. Thus, 0 is returned as the field value if a point

outside the deformation field is evaluated. In order to keep continuity of a scalar field, however, the point where the scalar field reaches 0 must have a zero first derivative. Therefore, a field discontinuity is generated on the boundary of the deformation field. This discontinuity can be seen in Figure 5.14b. Due to this discontinuity, a scalar field created in the interactive deformation approximation pass cannot guarantee continuity. When a blending operator is applied to the scalar field, it also creates undesirable visual artifacts (Figure 5.14c). This resulting transition is not smooth, so the blending property is also not satisfied.

Variational warping is used to compute a deformation field in the scalar field reconstruction pass. Since variational warping globally interpolates the vertices of the deformation grid, the entire scalar field can be contained in the deformation field. Variational warping has C^2 continuity, so the continuity of the deformed scalar field is also guaranteed. Blending and CSG operators are applied to a scalar field after variational warping in Figure 5.15c. Since constraints (the vertices of the deformation grid) are not placed outside the iso-surface, the resulting scalar field is a little distorted compared to a scalar field created based on a distance field. Nevertheless, this small distortion is acceptable because the resulting model still has a smooth transition and sharp edges.

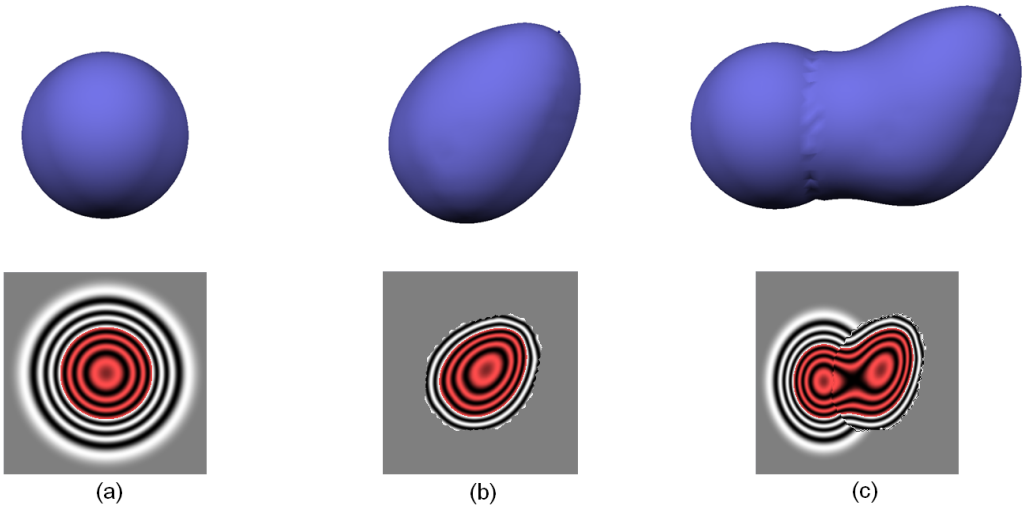


Figure 5.14: Evaluation of the interactive deformation approximation pass. The point primitive (a) is deformed by pulling the upper right of it. The discontinuity of the scalar field is generated near the iso-surface (b). A crease is also created in the discontinuity region when a blending operator is applied to the scalar field (c).

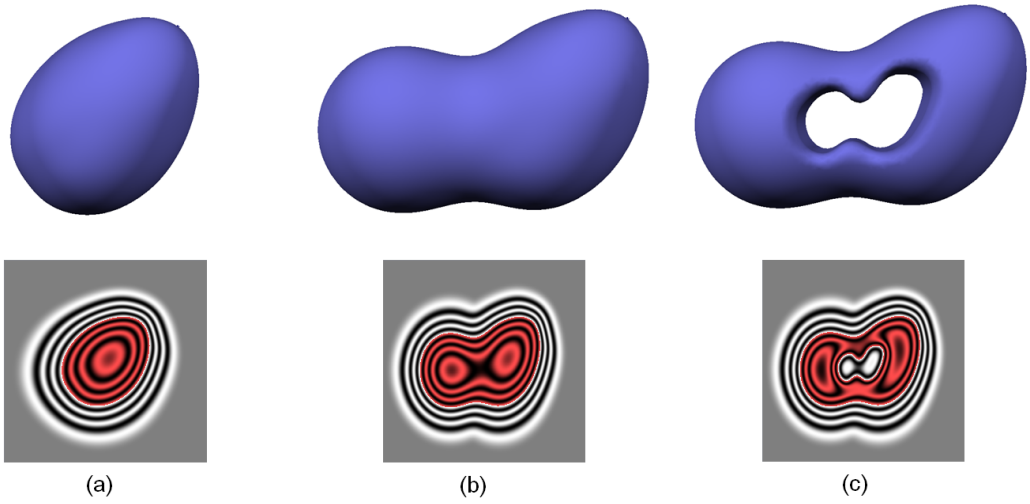


Figure 5.15: Evaluation of variational warping. Variational warping is applied to the scalar field of the deformed point primitive shown in Figure 5.14b (a). Unlike Figure 5.14c, a smooth transition can be created when a blending operator is applied (b). A CSG operator also creates sharp edges with keeping continuity (c).

Chapter 6

Results and Conclusion

This chapter provides the results, limitations, future work and conclusion of this work. Several models were created using Taco and these models show that a desirable deformation can be easily achieved while preserving implicit properties (Section 6.1.1). A benchmark is also provided in Section 6.1.2 which shows the timings of the interactive deformation approximation pass and the scalar field re-construction pass. Some limitations of Taco are described in Section 6.2, possible future work is proposed in Section 6.3 and this thesis is concluded in Section 6.4.

6.1 Results

Taco is evaluated by deforming some implicit models created in ShapeShop [48]. None of the FFD methods support implicit surfaces, so the contribution of Taco can be demonstrated by showing the properties of implicit surfaces (i.e. blending and CSG operators) after applying the FFD technique of Taco to the model. The volumetric peeling interface of Taco can also be evaluated by achieving various kinds of deformations and showing intuitiveness and easiness of the interface. By providing a benchmark, interactivity of Taco is evaluated.

6.1.1 Deformed Models

A significant benefit of implicit surfaces is that a smooth transition between two objects can be created with simple formulations (blending). It is difficult to apply blending to the other representations. The dragon model shown in Figure 6.1 demonstrates this property after applying Taco's deformation. The body of this dragon was

deformed with the 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 a blending operator. Even after deformation, smooth blending can be applied because of variational warping and hierarchical spatial caching. This model also demonstrates another benefit of Taco. Sketch-based modeling is a modeling method which constructs a 3D model according to user sketching, so the user input is limited to 2D. Although the user sketch is inflated into a 3D shape, the inflated shape must be along a 2D plane. This makes it difficult to create a more 3D-like shape. For example, the shape of the dragon body cannot be created in standard sketch-based modeling systems because this shape is not along a 2D plane. By adding a sketch-based deformation operator, modeling such a shape as sketch-based modeling was achieved.

Taco employs a peeling interface to manipulate deformations. A peeling interface is a deformation tool which automatically specifies ROI, and it has been well examined for use in 3D modeling [32]. Since a *volumetric* peeling interface has been developed in this research, the effects of a volumetric peeling interface were evaluated using the octopus model shown in Figure 6.2 and were compared with [32]. The face of the octopus was deformed by pulling the curve drawn on the mouth to several directions. The peeling effects volumetrically grows through the model from the curve. The octopus model demonstrates that various kinds of face shapes can be achieved just by pulling the mouth. This manipulation is intuitive for the user because it loosely imitates an interaction with a real object. It is difficult to deform like Figure 6.2 with a peeling interface proposed in [32] because the peeling effects are restricted in the user-drawn curve. More details of this comparison are discussed in Section 4.3. By directly recording the deformation sequence, animation can also be created easily.

In order to verify that Taco's deformation is suitable for implicit surfaces, one more deformation result is shown in Figure 6.3 where the Barr operation for implicit surfaces [62] was applied after Taco's deformation. First, the horse model shown in Figure 6.3 was created in ShapeShop. The neck of the horse model was then stretched by pulling the head upward to transform it into a giraffe model. The giraffe model was twisted by applying the Barr operation for implicit surfaces [62]. The warp [62] is applied along an axis, and in Figure 6.3 the giraffe model was twisted along the y-axis. The result shows that a scalar field created with Taco's deformation is also compatible with existing warp operations.

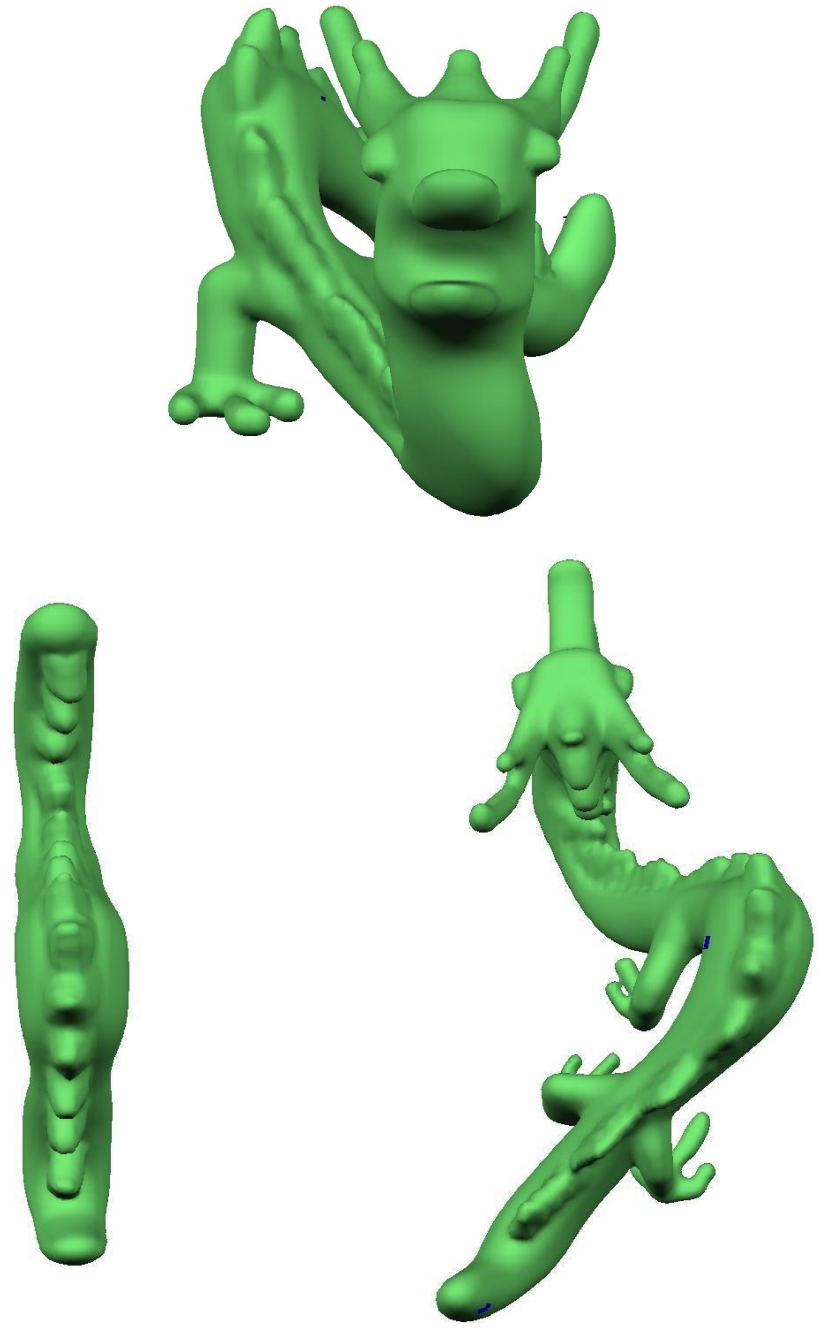


Figure 6.1: The dragon was deformed in Taco (top). The body of the dragon was initially modeled on a plane (left). The body was then deformed by pushing the middle of the body to the right and pushing the tail to the left (right).

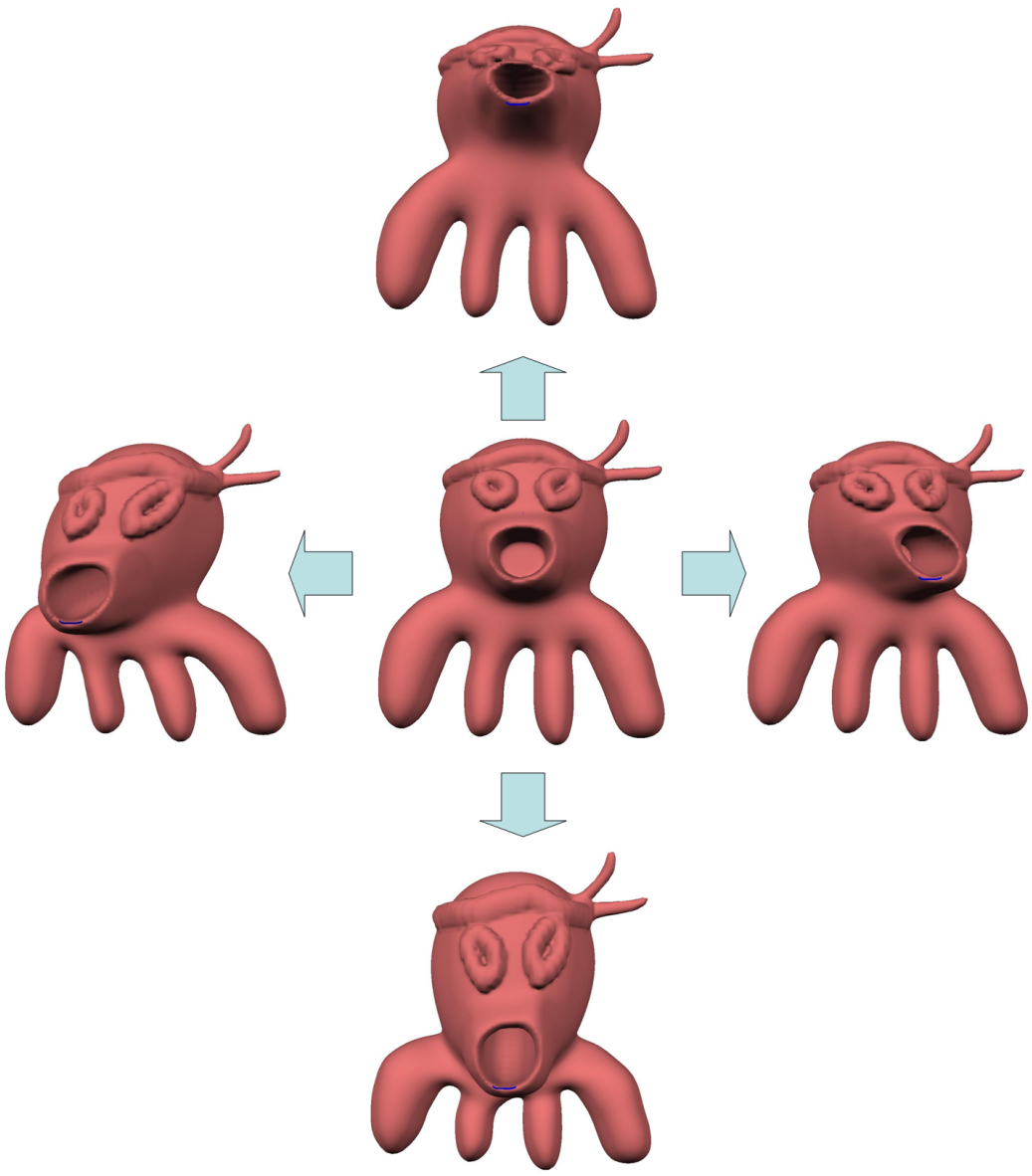


Figure 6.2: The octopus model was initially sketched, and then the mouth of the octopus was pulled. This figure shows the results deformed by pulling the mouth to each direction.

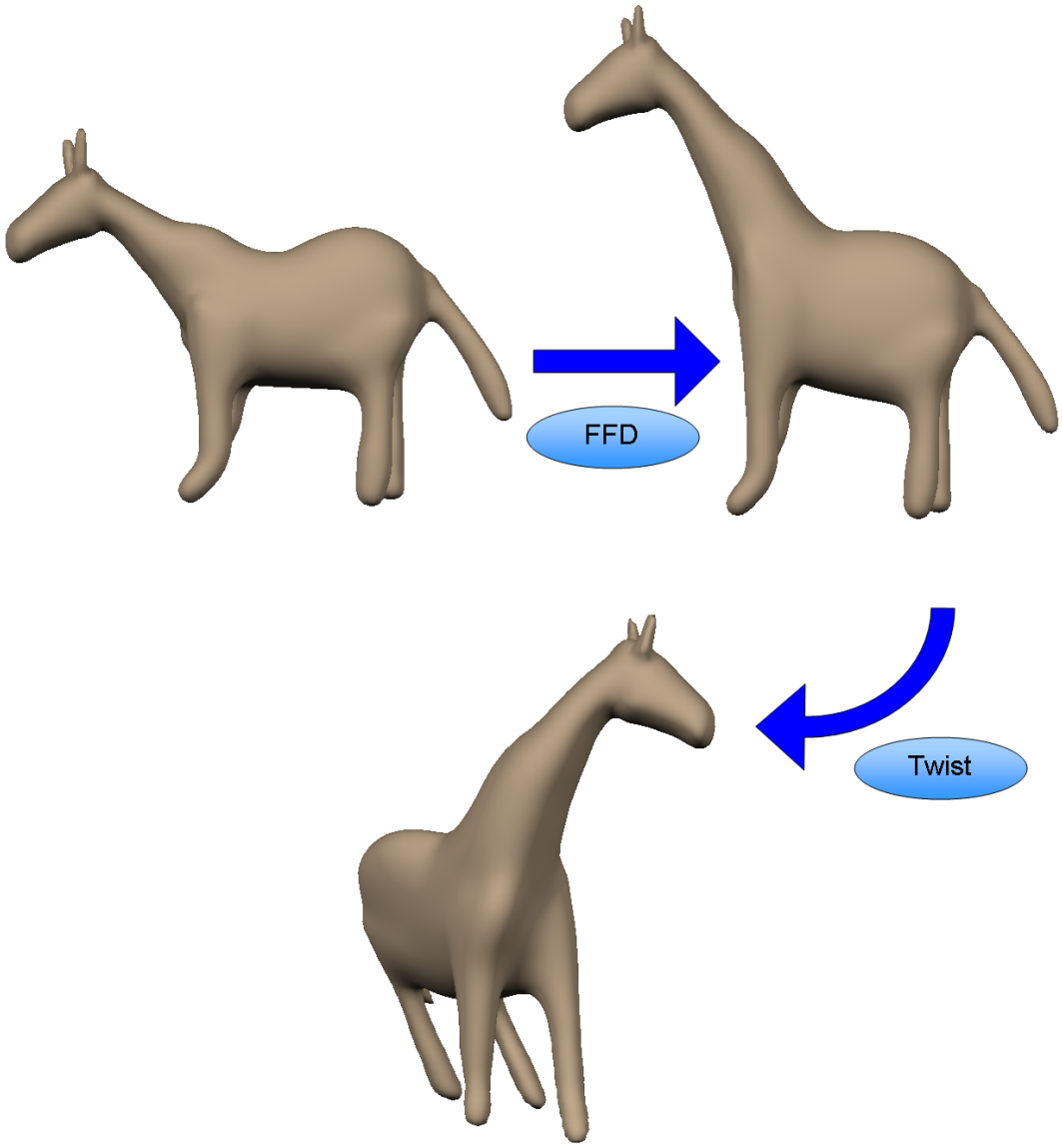


Figure 6.3: The horse model was transformed into the giraffe model by stretching the neck with Taco's free-form deformation. The giraffe model was then twisted along the y-axis.

6.1.2 Benchmark

Table 6.1 shows the performance of Taco. Taco was compiled using the Microsoft Visual Studio 2003 C++ compiler in release mode. All timings in Table 6.1 were measured on a workstation with Intel 2.66GHz Core 2 Quad CPU and 2.75GB of RAM. The performance was evaluated using the bar model (Figure 4.1), the duck model (Figure 5.2), and the octopus model (Figure 6.2). These three models were compared by measuring frame rates of the interactive deformation approximation pass and calculation time of variational warping. In this evaluation, polygonizer resolution and deformation grid resolution were fixed at 60^3 cubes and 30^3 cubes, respectively. Since the frame rates and the calculation time are not constant to each model but vary depending on deformation condition, the results shown in Table 6.1 are approximated numerical values.

Table 6.1 indicates that Taco lets the user deform implicit models interactively and re-constructs their scalar fields with variational warping in a short time. The frame rates of the duck model and the octopus model are slower than a simple model such as the bar model, however Taco still shows real-time performance. Variational warping is not suitable for interactive use but does not require too much calculation time. Therefore, this deformation tool is compatible with an interactive modeling system such as ShapeShop. The user can restart further modeling after deformation without having to wait for long time. Although polygonizer resolution was fixed at 60^3 cubes in this evaluation, higher resolution can be applied to polygonizer in real-time after deformation. This is because the resulting scalar field is stored at a cache node.

	Interactive Approximation (fps)	Variational Warping (seconds)
Bar	35	0.21
Duck	8	1.51
Octopus	6.5	1.78

Table 6.1: The timings of the interactive deformation approximation pass (fps) and variational warping (seconds) for three models. Polygonizer resolution and deformation grid resolution are fixed at 60^3 cubes and 30^3 cubes, respectively.

6.2 Limitations

There are a few limitations of Taco. First, Taco does not handle self-collision. This is a general problem with lattice-based FFD methods. If the deformation grid self-collides, the overlapped part of an inverse deformation field cannot be calculated correctly and sample points are warped into unexpected positions. Therefore, the user has to pull deformation curves so that the deformed model does not self-collide.

Second, controlling deformations relies on the voxel resolution. The volumetric peeling interface is achieved by approximating the shortest distance from the curve to each voxel vertex. Therefore, constructed voxels must be small enough to be able to approximate the shape of the deformed model. If surfaces of the deformed model are not connected with each other but connected by voxels, deformation behaves as if the surfaces were connected (Figure 6.4). To construct small voxels, higher resolution is required but computational cost increases.

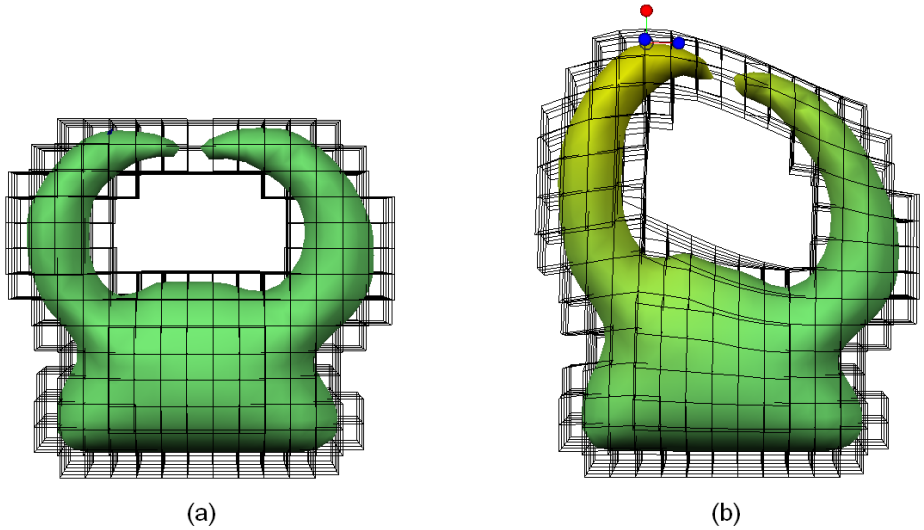


Figure 6.4: The horns of the model do not touch each other but the constructed voxels are connected. Because peeling effects grow through the voxels, the horns are deformed as if they were connected.

Third, the volumetric peeling interface is currently limited to translations, so the deformation behavior is only bending. By adding rotation and scaling manipulation to the user interface, it would be possible to apply twist and taper effects to the deformed model. In this work, rotation manipulation tried to be implemented for twisting, however a scalar field could not be re-constructed properly with variational warping as shown in Figure 6.5c. That is because constraints for calculating a variational warp

are not placed outside the iso-surface in Taco. A warp for bending can be calculated without the constraints outside the iso-surface, but they are required for twisting. In Taco, voxel vertices are used for constraints of variational warping. Voxels cannot be constructed outside the iso-surface, however, because the shortest distances cannot be calculated and undesirable peeling effects are introduced.

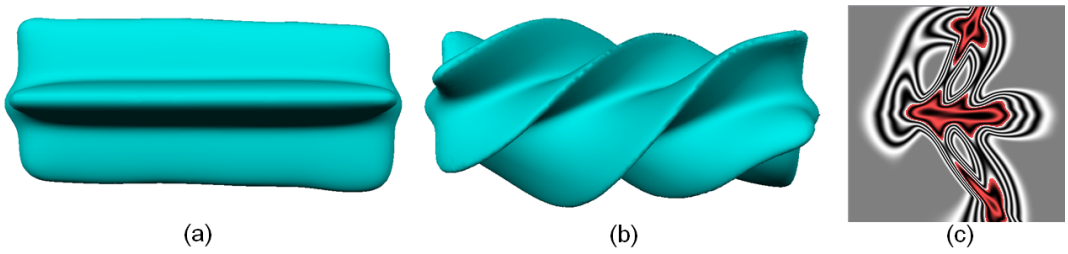


Figure 6.5: The twisting result with rotation manipulation. This fan model was twisted by rotating the right side of the model (b). This rotation smoothly decreases from right to left in order to archive twist effects. Due to an insufficiency of constraints, however, the re-constructed scalar field with variational warping is distorted (c).

6.3 Future Work

There are some possible extensions and improvements of Taco as future work. As discussed in Section 4.3, a curve editing function would improve the deformation flexibility of Taco. In the current Taco system, the ROI cannot be localized. If the user pulls the curve far away, the ROI grows out to the entire shape and this deformation turns out to be global. Also, the user-drawn curve works as a handle to deform a 3D model and the curve itself cannot be deformed. By enabling the curve to be deformed and restricting the ROI on the curve, local deformation would be achieved even if the curve is pulled far away.

The deformation of Taco behaves like the function which defines the amount of translation (Section 5.3.1). Since the Wyvill function is currently used in Taco, a smooth transition is created between the deformed part and the non-deformed part like the Wyvill function. Therefore, experimentation with more functions would enable various deformation behaviors.

The volumetric peeling interface of Taco can be improved by adding rotation and scaling manipulations. Rotating and scaling a curve would be associated with a twist operator and a taper operator, respectively. As discussed in Section 6.2, however, a

twisted scalar field cannot be re-constructed in the current Taco system. A method to place constraints outside the iso-surface is required to solve this problem. Scaling has not been implemented yet in Taco, so this operator should also be experimented in the future.

The volumetric peeling interface is not limited to implicit surfaces. Although the volumetric peeling interface has been developed to manipulate implicit models in this research, it could be adapted to any point based representation in theory. Since a forward warp can be applied to point based representations, adapting a volumetric peeling interface to them is relatively straightforward compared to implicit surfaces. Future experimentation could explore this and then be incorporated into the volumetric peeling interface.

6.4 Conclusion

In this thesis, a novel free-form deformation method for implicit surfaces has been introduced. In Chapter 1, two major issues of developing an FFD method for implicit surfaces were addressed. While implicit surfaces have many 3D modeling advantages, defining deformations in the implicit domain is difficult. Achieving a desirable deformation with lattice-based FFD can also be time-consuming. Therefore, the goals of this research were to propose an FFD technique suitable for implicit surfaces and develop an interface to control the FFD technique.

The background of this research was provided in Chapter 2 and Chapter 3. Chapter 2 described the fundamentals of implicit surfaces which are the shape representation used in this research. Chapter 3 focused on previous work related to this research. FFD techniques and deformation for implicit surfaces were discussed for the first goal - to propose an FFD for implicit surfaces. The survey of sketch-based modeling techniques was given for the second goal - to develop an efficient interface for FFD.

A volumetric peeling interface for controlling an FFD was proposed in Chapter 4. Since a standard FFD interface problem is how to specify the region of interest, a peeling technique which lets the user intuitively deform a model without manual ROI specification was adapted to an FFD. Unlike existing peeling interfaces which peel an ROI on the curve, the volumetric peeling interface proposed in Chapter 4 peels a 3D model itself volumetrically. The intuitiveness of this interface was demonstrated by stretching and squashing a 3D model which loosely imitates real object behaviors. A

discussion about advantages and disadvantages of a volumetric peeling interface was also provided at the end.

The main deformation algorithm was introduced in Chapter 5. The deformed model is represented by voxels and the user manipulates them via deformation curves. A variational warp technique was employed to construct a deformation field using the vertices of the voxels as constraints. This technique was able to achieve an FFD for implicit surfaces while preserving implicit properties, allowing smooth blending to be applied after deformation. Since variational warping is computationally expensive for interactive use, however, the interactive deformation approximation pass which gives the user interactive feedback was also presented. This pass visually approximates the effect of the deformation interactively based on the voxel grid. When the user is satisfied with the resulting shape, variational warping is applied.

Taco was evaluated by deforming three implicit models, and the results showed that a desirable deformation can be achieved intuitively while preserving advantages of implicit modeling. This achieves the goals of the research. In the future, it is hoped that Taco will be utilized in various modeling applications not only for implicit surfaces but also other shape representations. Since deformation is also a useful technique for animation, Taco is promising as an interactive animation interface.

Bibliography

[1] Adzhiev, V., Cartwright, R., Fausett, E., Ossipov, A., Pasko, A., and Savchenko, V. Hyperfun project: a framework for collaborative multidimensional f-rep modeling. In *Proceedings of Implicit Surfaces '99* (1999), pp. 59–69.

[2] Alexe, A., Gaildrat, V., and Barthe, L. Interactive modelling from sketches using spherical implicit functions. In *AFRIGRAPH '04: Proceedings of the 3rd international conference on Computer graphics, virtual reality, visualisation and interaction in Africa* (2004), pp. 25–34.

[3] Araújo, B., and Jorge, J. Blobmaker: Free-form modelling with variational implicit surfaces. In *Proceedings of 12th Encontro Português de Computação Gráfica* (2003), pp. 17–26.

[4] Barr, A. H. Global and local deformations of solid primitives. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques* (1984), pp. 21–30.

[5] Barthe, L., Wyvill, B., and de Groot, E. Controllable binary CSG operators for soft objects. *International Journal of Shape Modeling 10*, 2 (2004), 135–154.

[6] Blinn, J. F. A generalization of algebraic surface drawing. *ACM Transactions on Graphics 1*, 3 (July 1982), 235–256.

[7] Bloomenthal, J. *Introduction to Implicit Surfaces*. Morgan Kaufmann, ISBN 1-55860-233-X, 1997.

[8] Botsch, M., and Kobbelt, L. Real-time shape editing using radial basis functions. *Computer Graphics Forum 24*, 3 (2005), 611–621.

[9] Botsch, M., Pauly, M., Wicke, M., and Gross, M. Adaptive space deformations based on rigid cells. *Computer Graphics Forum 26*, 3 (2007), 339–347.

[10] Cani, M.-P., and Desbrun, M. Animation of deformable models using implicit surfaces. *IEEE Transactions on Visualization and Computer Graphics (TVCG) 3*, 1 (1997).

[11] 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 *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), pp. 67–76.

[12] Chen, B.-Y., Ono, Y., Johan, H., Ishii, M., Nishita, T., and Feng, J. 3d model deformation along a parametric surface. In *Proceedings of the 2nd IASTED International Conference on Visualization, Imaging and Image Processing* (2002), pp. 282–287.

[13] Coquillart, S. Extended free-form deformation: a sculpturing tool for 3d geometric modeling. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques* (1990), pp. 187–196.

[14] Crespin, B. Implicit free-form deformations. In *Proceedings of Implicit Surfaces '99* (1999), pp. 17–23.

[15] de Groot, E., and Wyvill, B. Rayskip: faster ray tracing of implicit surface animations. In *GRAPHITE '05: Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia* (2005), pp. 31–36.

[16] Dijkstra, E. W. A note on two problems in connexion with graphs. *Numerische Mathematik 1* (1959), 269–271.

[17] Gain, J., and Bechmann, D. A survey of spatial deformation from a user-centered perspective. *ACM Transactions on Graphics 27*, 4 (2008), 1–21.

[18] Galbraith, C., Mundermann, L., and Wyvill, B. Implicit Visualization and Inverse Modeling of Growing Trees . *Computer Graphics Forum 23*, 3 (2004), 337–348.

[19] Gascuel, M.-P. An implicit formulation for precise contact modeling between flexible solids. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (1993), pp. 313–320.

[20] Guy, A., and Wyvil, B. Controlled blending for implicit surfaces using a graph. In *Implicit Surfaces '95* (Apr. 1995), pp. 107–112.

[21] Hart, J. C. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer 12*, 10 (Dec. 1996), 527–545.

[22] Igarashi, T., and Cosgrove, D. Adaptive unwrapping for interactive texture painting. In *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics* (2001), pp. 209–216.

[23] Igarashi, T., and Hughes, J. F. Smooth meshes for sketch-based freeform modeling. In *I3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics* (2003), pp. 139–142.

[24] Igarashi, T., Matsuoka, S., and Tanaka, H. Teddy: a sketching interface for 3d freeform design. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (1999), pp. 409–416.

[25] Igarashi, T., Moscovich, T., and Hughes, J. F. As-rigid-as-possible shape manipulation. *ACM Transactions on Graphics 24*, 3 (2005), 1134–1141.

[26] Kalra, D., and Barr, A. H. Guaranteed ray intersections with implicit surfaces. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques* (1989), pp. 297–306.

[27] Karpenko, O., Hughes, J., and Raskar, R. Free-form sketching with variational implicit surfaces. *Computer Graphics Forum 21* (2002), 585–594.

[28] Kobbelt, L. P., Botsch, M., Schwanecke, U., and Seidel, H.-P. Feature sensitive surface extraction from volume data. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), pp. 57–66.

[29] Kojekine, N., Savchenko, V., Berzin, D., and Hagiwara, I. Software tools for compactly supported radial basis functions. In *CGIM 2001, IASTED*

Fourth International Conference on Computer Graphics and Imaging (2001), pp. 13–16.

[30] Kojekine, N., Savchenko, V., Senin, M., and Hagiwara, I. Real-time 3d deformations by means of compactly supported radial basis functions. In *Short papers proceedings of Eurographics* (2002), pp. 35–43.

[31] MacCracken, R., and Joy, K. I. Free-form deformations with lattices of arbitrary topology. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (1996), pp. 181–188.

[32] Nealen, A., Igarashi, T., Sorkine, O., and Alexa, M. Fibermesh: designing freeform surfaces with 3d curves. *ACM Transactions on Graphics 26*, 3 (2007).

[33] Nishimura, H., Hirai, M., Kawai, T., Kawata, T., Shirakawa, I., and Omura, K. Object modeling by distribution function and a method of image generation. *Transactions IECE Japan, Part D J68-D*, 4 (1985), 718–725.

[34] Nur, M. A. Animating implicit surfaces. Master's thesis, University of Calgary, 2002.

[35] Ohtake, Y., Belyaev, A., and Pasko, A. Dynamic mesh optimization for polygonized implicit surfaces with sharp features. *The Visual Computer 19* (2002), 115–126.

[36] Ono, Y., Chen, B.-Y., Nishita, T., and Feng, J. Free-form deformation with automatically generated multiresolution lattices. In *CW '02: Proceedings of the First International Symposium on Cyber Worlds* (2002), pp. 472–479.

[37] Opalach, A., and Cani, M.-P. Local deformation for animation of implicit surfaces. In *Spring Conference on Computer Graphics (SCCG)* (1997).

[38] Owada, S., Nielsen, F., Nakazawa, K., and Igarashi, T. A sketching interface for modeling the internal structures of 3d shapes. In *Proceedings of the 4th International Symposium on Smart Graphics* (2003), pp. 49–57.

[39] Pasko, A., Adzhiev, V., Schmitt, B., and Schlick, C. Constructive hypervolume modeling. *Graphical models 63*, 6 (2001), 413–442.

[40] Pasko, A., Adzhiev, V., Sourin, A., and Savchenko, V. Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer 11*, 8 (1995), 429–446.

[41] Prasad, L. Morphological analysis of shapes. *CNLS Newsletter 139* (1997), 1–18.

[42] Ricci, A. A constructive geometry for computer graphics. *Computer Journal 16*, 2 (May 1973), 157–160.

[43] Savchenko, V., Pasko, A., Okunev, O., and Kunii, T. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum 14* (1995), 181–188.

[44] Schmidt, R., Grimm, C., and Wyvill, B. Interactive decal compositing with discrete exponential maps. *ACM Transactions on Graphics 25*, 3 (2006), 605–613.

[45] Schmidt, R., and Singh, K. Sketch-based procedural surface modeling and compositing using Surface Trees. *Computer Graphics Forum 27*, 2 (2008), 321–330.

[46] Schmidt, R., and Wyvill, B. Generalized sweep templates for implicit modeling. In *GRAPHITE '05: Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia* (2005), pp. 187–196.

[47] Schmidt, R., Wyvill, B., and Galin, E. Interactive implicit modeling with hierarchical spatial caching. In *SMI '05: Proceedings of the International Conference on Shape Modeling and Applications 2005* (2005), pp. 104–113.

[48] Schmidt, R., Wyvill, B., Sousa, M., and Jorge, J. Shapeshop: Sketch-based solid modeling with blobtrees. In *Proceedings of the 2nd Eurographics Workshop on Sketch-Based Interfaces and Modeling* (2005), pp. 53–62.

[49] Schmitt, B., Pasko, A., and Schlick, C. Shape-driven deformations of functionally defined heterogeneous volumetric objects. In *GRAPHITE '03: Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia* (2003), pp. 127–134.

[50] Sederberg, T. W., and Parry, S. R. Free-form deformation of solid geometric models. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques* (1986), pp. 151–160.

[51] Shapiro, V. Theory of r-functions and applications: a primer. Tech. Rep. CPA88-3, Cornell University, 1988.

[52] Shapiro, V. Real functions for representation of rigid solids. *Computer Aided Geometric Design 11*, 2 (1994), 153–175.

[53] Singh, K., and Fiume, E. Wires: a geometric deformation technique. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (1998), pp. 405–414.

[54] Sugihara, M., de Groot, E., Wyvill, B., and Schmidt, R. A sketch-based method to control deformation in a skeletal implicit surface modeler. In *Proceedings of the 5th Eurographics Workshop on Sketch-Based Interfaces and Modeling* (2008), pp. 65–72.

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

[56] Tigges, M., and Wyvil, B. A field interpolated texture mapping algorithm for skeletal implicit surfaces. In *CGI '99: Proceedings of the International Conference on Computer Graphics* (1999), pp. 25–33.

[57] Turk, G., and O'Brien, J. F. Shape transformation using variational implicit functions. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (1999), pp. 335–342.

[58] Turk, G., and O'brien, J. F. Modelling with implicit surfaces that interpolate. *ACM Transactions on Graphics 21*, 4 (2002), 855–873.

[59] Wendland, H. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Advances in Computational Mathematics 4*, 1 (1995), 389–396.

[60] Wyvill, B., Guy, A., and 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.

[61] Wyvill, B., and van Overveld, K. Polygonization of implicit surfaces with constructive solid geometry. *Journal of Shape Modelling 2*, 4 (1996), 257–274.

[62] Wyvill, B., and van Overveld, K. Warping as a modelling tool for csg/implicit models. In *SMA '97: Proceedings of the 1997 International Conference on Shape Modeling and Applications* (1997), pp. 205–214.

[63] Wyvill, G., McPheeters, C., and Wyvill, B. Data structure for soft objects. *The Visual Computer 2*, 4 (1986), 227–234.