THE UNIVERSITY OF CALGARY

# Interactive Modeling with Implicit Surfaces

by

Ryan Schmidt

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA
August, 2006

# THE UNIVERSITY OF CALGARY
# FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "Interactive Modeling with Implicit Surfaces" submitted by Ryan Schmidt in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE.

Supervisor,
Dr. Brian Wyvill
Department of Computer Science

External Examiner,
Dr. Neil Duncan
Department of Civil Engineering

Dr. Faramarz Samavati
Department of Computer Science

Dr. Mario Costa Sousa
Department of Computer Science

Date

# Abstract

Interactive tools for shape modeling with hierarchical implicit surfaces have been limited both by the high computational cost of visualization, and the lack of techniques for direct surface manipulation. To address the visualization issue, Hierarchical Spatial Caching is developed. This novel technique combines caching and spatial approximation to accelerate queries of the functional model tree. By reducing the cost of evaluating cached branches from $O(N)$ to $O(1)$, an order-of-magnitude improvement in visualization speed is realized. A new implicit sweep surface formulation which supports direct manipulation of the sweep profile is also developed, providing a powerful and flexible free-form implicit primitive. These new techniques form the core of a proof-of-concept interactive modeling environment, called ShapeShop. ShapeShop provides a level of interactive control over hierarchical implicit models which has not been previously available. A survey of current techniques for shape modeling with implicit surfaces is also provided.

# Acknowledgements

Where does one begin? Clearly, there is Ailidh. Thank you for your patience (particularly at paper deadlines), encouragement, understanding, and support. And thank you for listening. Without you, this whole thing simply wouldn't have happened.

Of course, there is Brian Wyvill, with his subtle pushes at opportune moments. Thanks for all your encouragement, advice, and willingness to let me figure things out for myself (hopefully I wasn't too much trouble). Oh, and thanks for the second chance. I also have to thank Frank Maurer, for introducing me to the research game (Sorry I skipped out on you for graphics - but then, you always knew I was a hacker). There have been many other professors who have made an impact on my work. Mario Costa Sousa, Faramarz Samavati, Przemek Prusinkiewicz, Jon Rokne, Ehud Sharlin, Sheelagh Carpendale, Saul Greenberg, Richard Lobb, and Joaquim Jorge - whether technical assistance, encouragement, or simply stimulating conversation, you have all helped me along the way. Also thanks to Eric Galin, your help with the SMI paper was invaluable, and Neil Duncan who gave me a glimpse of life beyond computer graphics.

There have been many students who have helped me along the way, far too many to list here. Hopefully, you know who you are. First and foremost, thanks to all the members of the Graphics Jungle and Interactions Lab at the University of Calgary. In particular, Pauline Jepp, Tobias Isenberg, Michael Boyle, Callum Galbraith, Kevin Foster, Anand Agarawala, Stacey Scott, Mark Hancock, Torre Zuk, Mark Fox, Chester Fitchett, Robson Lemos, Brendan Lane, Carla Davidson, Lars Muenderman, Marilyn Powers, and Peter MacMurchy. Thank you for your assistance, encouragement, conversation and friendship. Also thanks to the many great people I have met at conferences, particularly Tamy Boubekeur, Patricio Simari, Nisha Sudarsanam, and Loic Barthe. And, of course, my agent, Rob Leclerc, who got me into graphics in the first place, and is always willing to challenge my thinking. The support staff at the University of Calgary have also been fantastic, particularly Wayne Pearson, Camille Sinanan, and the front-office staff.

Finally, my friends and family have been a source of never-ending support. The residents of the Sugar Shack made a significant impact on my work (and mental well-being), as have all my other house-mates along the way. Thanks to my parents, for their support (both financial and otherwise), and to Kyle and Mandy, who each keep me on my toes in their own way.

# Table of Contents

# Chapter 1

# Introduction

Interactive shape modeling is a wide-ranging topic with a history reaching back to the earliest days of interactive computing. For example, one of the first works in computer graphics was the SketchPad system, the subject of Ivan Sutherland's dissertation in 1963 [111]. Sketchpad, one of the pioneering works in Computer-Aided Design [47], was essentially a 2D modeling system. The interaction techniques introduced in Sketchpad have heavily influenced the design of modern 3D modeling interfaces.

One issue inherent in any 3D modeling system is how to represent shapes. The common shape representation techniques have varying strengths and weaknesses with respect to different interaction techniques. For example, parametric spline patches can be easily sculpted by manipulating their control points, but applying similar deformations to a sphere defined as an implicit surface is quite difficult. In contrast, creating a blend surface between two solid objects represented using parametric patches can be extremely difficult, while blending two implicitly-defined solid objects is nearly trivial. Modern shape modeling interfaces are heavily influenced by these differences. In fact, most commercial modeling tools are identified by how they represent shapes - some are "parametric" or "patch-based" modelers, while others are "mesh editors", "subdivision systems", "volume sculpting tools", and so on. The design of these interfaces is driven by what kinds of operations the underlying shape representation easily supports.

In this respect, implicit surfaces are simply another option in the library of shape representation techniques. However, implicit surfaces have some tangible benefits in the domain of *solid modeling* [85, 86], which is a subset of shape modeling explicitly concerned with the representation of 3D models which are in a sense "isomorphic" to some real-world shape. Not all models used in computer graphics are solid models - for example, pieces of cloth are often represented by infinitely-thin triangle meshes. However, this is usually done for efficiency, and solid models arguably would be more accurate since all physical objects have a volume (and in the case of deformable surfaces, are known to produce more realistic simulations [31]). With the increasing availability of rapid prototyping machines, also known as "3D printers", interest in solid modeling is growing, and with it the need for solid modeling interfaces. This work will focus on such interfaces.

Solids can be represented in a variety of ways. The most common approach in computer graphics is to represent a solid by describing it's *boundary* - the infinitely-thin surface that makes up the exterior of the solid. Complex boundaries are usually described by combining a set of boundary *patches*. The union of the boundary patches encloses some 3D space, which is the volume of the solid. Hence, the boundary of a cube could be 6 square patches. This type of solid is often referred to as a *boundary represention* or *B-Rep* [85, 86]. Common techniques used to define boundary patches include parametric spline surfaces, triangle meshes, subdivision meshes, and analytic surfaces such as conics.

The alternative to boundary representations are volumetric representations, where the solid is represented directly. Volumetric representations are based on the concept of spatial enumer-

ation, where each point is classified as being inside or outside the volume. Note that while any volumetric representation can be considered a boundary representation (since the surface of the volume is it's boundary), the reverse is not true. The fundamental distinction is that with volume representations, any 3D point can always be classified as inside or outside the surface. This is known as the *point classification* test, and is inherently related to the notion of *validity* of solid models. With B-Reps, it is possible to have a hole or "crack" between two boundary patches, or to have two boundary patches intersect. In these cases, "inside" and "outside" are not mathematically well-defined, and hence (for the purposes of representing a solid) the model is *invalid* [85].

With respect to validity in solid modeling, volumetric models have clear benefits over boundary representations. While this may seem like a minor difference, invalid models are a significant problem because many applications of solid modeling, such as physical simulation, assume that the input models are valid. With invalid models, the results of the simulation are meaningless. Invalid boundary models are so prevalent that extensive research is being carried out on techniques to automatically repair broken B-Rep surfaces [20].

This validity problem motivates the need for interactive volume modeling interfaces. However, there are many different types of volumetric representation. In this work, a specific type of volume model known as a BlobTree implicit model [120] will be used. The BlobTree modeling framework has a variety of useful properties. Solids can easily be combined using the Boolean operations of Constructive Solid Geometry (CSG). The BlobTree also supports smooth blending between solids, as well as global deformation operations. The BlobTree is built on functional implicit volumes and operators, and hence curved surfaces are mathematically smooth (except at creases). Since operators are functional, rather than based on geometric algorithms, they are agnostic to scale and complexity. Hence, two solids at grossly different scales can be combined without any need for special processing or algorithms[1]. Regardless of how intricately detailed the two shapes are, one can always be subtracted from the other by applying trivial mathematical operations, and the result is always mathematically exact. The procedural, hierarchical nature of BlobTree models supports animation directly. And, since the entire model hierarchy is dynamically evaluated, any modeling operation which has been applied can always be reversed at any later time, essentially providing an infinite and non-linear "undo" facility.

These properties are very desirable in an interactive solid modeling context. However, there are some practical difficulties. Common boundary representations such as meshes and spline patches, with their simple explicit definitions of the surface, provide a straightforward pipeline to both visualization (rendering) and direct surface manipulation. BlobTree implicit models provide no such facility. The terms "implicit model" and "implicit surface" are often used interchangeably, and as this name suggests, the surface is defined implicitly as the solution set of some general equation. Solving for this three-dimensional solution set requires a time-consuming search through 3D space. Time-consuming visualization algorithms do not fit well into traditional interactive 3D modeling interfaces, which rely heavily on frequent design iteration and constant visual feedback. The lack of an explicit surface also makes basic shape editing very difficult, as the surface cannot be directly manipulated. Direct surface manipulation has long been a

[1] of course, simultaneous visualization of both scales can be quite difficult

feature of B-Rep modeling systems, and is virtually a necessity for any modern interactive shape modeling tool.

It seems clear that implicit surfaces have the potential to greatly simplify solid modeling interfaces. However, when comparing the array of commercially available B-Rep and implicit modeling systems, an overwhelming disparity is immediately apparent. B-Rep systems are clearly the current favorite for interactive modeling. Implicit surfaces are making inroads in some areas, such as surface reconstruction from range scans, and many of the aforementioned B-Rep repair systems are based on implicit techniques [20]. However, the two problems noted above - interactive visualization and direct surface editing - essentially preclude the use of implicit surfaces in interactive modeling systems. It is these issues which this research aims to address.

## 1.1 Goals

Since implicit models lack an explicit definition of the surface, visualization techniques must "find" the surface using spatial searches. The only information available about a general implicit surface is the value of it's defining scalar field at points in space. Hence, to conduct a spatial search, many evaluations of the equation which produces the scalar field are necessary. Since these functions are generally quite expensive, visualization is slow (See Section 4.1 for more details).

As noted, this work focuses on BlobTree implicit models. At a conceptual level, a BlobTree is defined by combining simple *primitives*, defined by scalar functions, using *operators*, which are also scalar functions. BlobTree models are created by successively composing these simpler functions, resulting in a hierarchy of primitive and operator functions. These functions are treated as "black boxes", meaning that the internal structure of the functions are completely unknown. While this does lead to more general algorithms, it prohibits many types of optimization which could speed up visualization.

The first goal of this work is to increase the speed at which a changing BlobTree model can be visualized. Since visualization algorithms are based on evaluations of the BlobTree scalar field, the most promising direction for reducing visualization time is to make evaluations faster. Hence, a more precise goal is to reduce the cost of evaluating the BlobTree scalar field. A variety of techniques for achieving this goal have been proposed [29, 7, 49, 48, 18, 15, 51], however none provide the level of improvement necessary for scalable interactive BlobTree modeling. In retrospect, it is clear that an order-of-magnitude improvement is necessary (and will be achieved in this work).

Direct control over the implicit surface has also been noted as a significant problem for interactive implicit modeling. This is particularly true in the BlobTree modeling framework, where the designer only has control over the position and orientation of the various primitives, and over a set of parameters. Only indirect control over the surface is possible, by manipulating these parameters. The effects of such abstract parameters can be quite difficult to predict. A second goal of this research is to develop techniques which provide more intuitive, direct control over the implicit surface. In the BlobTree framework, direct control can most easily be realized at the primitive level. Hence, the goal is narrowed down to developing a BlobTree primitive

which provides direct control over the surface.

Of course, these two goals are only pieces of a general puzzle. At a more conceptual level, the purpose of this work is to enable the construction of an interactive BlobTree modeling system. Without such a system, the BlobTree framework cannot be reasonably compared to existing interactive modeling systems. Hence, the third goal is to design such a system, with an emphasis on interaction. This system will be built upon the new techniques developed to address the interactive visualization and surface manipulation problems. To achieve this goal, some sacrifices must be made - particularly with respect to interactive visualization, where maintaining both high visual accuracy and real-time feedback is extremely challenging.

## 1.2 Contributions

In the following chapters, several contributions are made. In Chapter 2, an introduction to implicit surfaces and volumes is provided. Mathematical properties related to shape modeling are discussed, such as composition operators, continuity, and normalization. The sections on perceptual discontinuities, normalization images, and the scale problem have not appeared previously in the literature.

Chapter 3 is closely tied to Chapter 2. Various implicit surface modeling techniques are compared based on the properties identified in Chapter 2. The goal of this chapter is two-fold. First, combined with Chapter 2 it provides a modest survey of the state-of-the-art in implicit surfaces. While the taxonomy developed can undoubtedly be refined, no comparable classification of implicit surface methods is available. Second, the choice of the BlobTree hierarchical implicit modeling framework as the basis for an interactive system is justified.

The interactive visualization problem is addressed in Chapter 4. A novel technique called Hierarchical Spatial Caching is developed which significantly reduces the cost of evaluating the scalar functions that define hierarchical implicit BlobTree models. This new technique combines aspects of existing traversal cache methods with spatial approximation of scalar fields. Acceleration structures called *cache nodes* are placed directly into hierarchical model tree. Dynamically-generated volume datasets are proposed as a means for realizing cache nodes. A thorough discussion of implementation issues is provided, as well as various profiling results which show an order-of-magnitude improvement in visualization time. This speed-up is sufficient to provide interactive visual feedback for moderately complex implicit models. Finally, an extensive analysis of Adaptive Distance Fields (ADFs) is carried out. ADFs have been suggested as a more suitable data structure for implementing Hierarchical Spatial Caching, however the analysis in this chapter identifies several outstanding issues which make ADFs unsuitable for spatial caching. This chapter includes extensive additional material beyond the published version [97].

Chapter 5 describes a new technique for generating implicit sweep surfaces. The advance here is the development of a smooth $C^2$ approximation to the distance field for an arbitrary set of closed 2D curves. This smoothed distance field is used to generate a sweep template scalar field. The main benefits of the resulting 3D sweep surfaces is that they support direct manipulation of the sweep contour, and are compatible with the BlobTree. Other improvements include the integration of sharp creases into the sweep template, and three new sweep endcap styles. Several

applications are also discussed. This chapter includes extra material omitted from the published version [96].

The techniques developed in Chapters 4 and 5 have been implemented in a prototype interactive BlobTree modeling system called ShapeShop. This proof-of-concept system is described in Chapter 6. ShapeShop includes both traditional and novel sketch-based modeling interaction styles. A brief overview of the interaction techniques available in ShapeShop is followed by a implementation details on interactive visualization and the BlobTree architecture used in the system. In the interests of brevity, this chapter focuses on interactive implicit modeling issues, and largely avoids the sketch-based modeling aspects which have been published elsewhere [99].

Chapter 7 provides 3D modeling results, primarily in the form of a gallery of hierarchical implicit models constructed using ShapeShop. These models demonstrate the versatility and power of ShapeShop. A final assessment and summary of the thesis is also included here.

## 1.3 Scope

In the following chapters (particularly Chapters 2 and 3), readers familiar with implicit surfaces may notice that discussion of a variety of issues is conspicuously absent. In particular, there will be little reference to computational aspects of various implicit surface schemes, such as efficiency or accuracy. This is intentional - computational limitations rarely persist. For example, implicit modeling techniques have long been dismissed as being "too slow" for interactive modeling, a statement which will be debunked in Chapter 4. The analytic properties considered in Chapter 2 are fundamental; their limitations cannot be mitigated with clever algorithms.

The scope of this thesis is also limited to shape modeling. Implicit surfaces have been applied in a variety of other problem domains, such as animation [124, 40, 120, 33], morphing [54, 16], and physical simulation [32, 41]. Numerous difficult challenges exist which will be completely ignored, such as accurate rendering of implicit surfaces [39], or surface parameterization and texture mapping [94]. Rather than attempt to provide a brief overview of such a wide-ranging field, this thesis will be limited to issues specifically related to the interactive construction of static solid models.

## 1.4 Summary

Guaranteed validity, functional representation, infinite-scale modeling, and non-linear procedural editing are significant benefits of implicit modeling. It is true that implicit modeling is not a "drop-in" replacement for current B-Rep modeling techniques; designers would have to adopt some new interaction styles. However, it is by no means certain that these new interaction styles would be less efficient or intuitive than the current state-of-the-art in B-Rep modeling. The primary goal of this research is to provide a framework in which interfaces for interactive hierarchical implicit modeling can be explored.

In the following chapters, several steps will be made towards this goal. First, the existing state-of-the-art in shape modeling with implicit surfaces will be analyzed. From the myriad methods available, the BlobTree hierarchical modeling system will be identified as having the

most desirable properties for interactive use. The largest drawback of BlobTrees - interactive visualization - will be addressed with the development of a hierarchical spatial caching technique. To improve shape control, sweep surfaces which permit direct specification of the sweep contour will then be developed. Finally, these new techniques will be applied in ShapeShop, a prototype interactive BlobTree modeling system.

# Chapter 2

# Shape Modeling with Implicit Surfaces

*Beginning with mathematical definitions of implicit surfaces and volumes, the major concepts in implicit modeling are introduced - CSG, blending, and general composition operators, bounded primitives, $C^0$ and $C^1$ continuity, and normalization. Novel analyses are provided for the* perceptual discontinuity *and* scaling *problems. Two new tools, the* normalization metric *and* normalization image*, are introduced for analyzing normalization error.*

## 2.1 Implicit Surfaces

Consider a function $f$ that, when applied to a point $\mathbf{p} \in \mathbb{E}^3$, produces a scalar value $f(\mathbf{p}) \in \mathbb{R}$. A surface $\mathcal{S} \subset \mathbb{E}^3$ can be defined by the equality

$$f(\mathbf{p}) = v \tag{2.1}$$

where $v$ is any scalar value in $\mathbb{R}$. This surface $\mathcal{S}$ is an *iso-contour* of the *scalar field* produced by $f(\mathbf{p})$, and $v$ is the *iso-value* that produces $\mathcal{S}$. In computer graphics, $\mathcal{S}$ is commonly known as an *implicit surface*. Functions $f$ will be referred to as *fields*, and specific values $f(\mathbf{p})$ will be called *field values*.

Note that by replacing $\mathbf{p}$ with a 2D point, Equation 2.1 can also be used to define 2D implicit curves. For clarity, some figures and examples will be shown in 2D.

By the above definition, most common surface representations used in computer graphics are implicit surfaces, because these representations all incorporate the notion of classifying points as being *on* or *off* the surface. A point is *on* a triangle mesh if it lies in any of the planar triangles that define the mesh. A point is *off* a NURBS surface if it cannot be produced by summation of the defining B-spline basis functions. Pathological cases, such as non-planar 3D polygons and fractal surfaces, would seem to be exceptions, however to be useful for shape modeling some ad-hoc binary condition must be invented.

This binary classification can be used to create a *binary implicit surface* by defining $f(\mathbf{p})$ such that it is 0 when $\mathbf{p} \in \mathcal{S}$ and 1 otherwise. Although rarely identified as such, this type of implicit surface is used extensively in boolean operations between meshes. However, for higher-order parametric surfaces such as NURBS surfaces, the only way to determine whether or not $\mathbf{p} \in \mathcal{S}$ is exhautive search of the parameter space - not an appealing option.

Another type of implicit surface is the *distance field*, defined with respect to some geometric entity $\mathsf{T}$:

$$f_{\mathsf{T}}(\mathbf{p}) = \min_{\mathbf{q} \in \mathsf{T}} |\mathbf{q} - \mathbf{p}| \tag{2.2}$$

Intuitively, $f_{\mathsf{T}}(\mathbf{p})$ is the shortest distance from $\mathbf{p}$ to $\mathsf{T}$. Hence, when $\mathbf{p}$ lies on $\mathsf{T}$, $f_{\mathsf{T}}(\mathbf{p}) = 0$ and the same surface is created as in the binary implicit surface. Otherwise, a non-zero distance is returned. $\mathsf{T}$ can be any geometric entity embedded in 3D - a point, curve, surface, or solid.

One challenge when analyzing implicit surfaces is visualizing the underlying scalar fields. A common technique is to regularly sample $f$ on a 2D planar slice through the field and map the values to grayscale, creating a *field image* (Figure 2.1a). Another useful visualization can be created by applying a sin function to the values of $f$ before mapping to grayscale. This creates a *contour diagram* (Figure 2.1d).

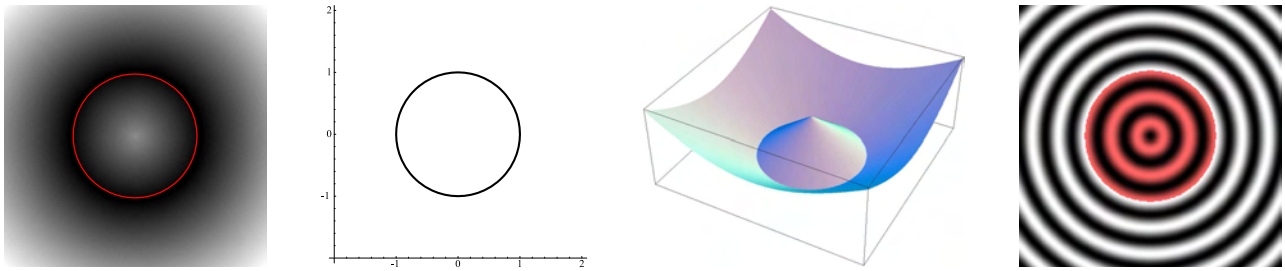



Figure 2.1: *A 2D implicit circle defined by the distance field* $f = \sqrt{x^2 + y^2} - 1$. *A* $2 \times 2$ *region of the infinite distance field* $f$ *is visualized in (a) by sampling* $f$ *at each pixel and mapping the value to grayscale. The circle lies on the zero iso-contour* $f = 0$, *highlighted in red in (a) and shown explicity in (b). The field* $f$ *is plotted as a standard height map in (c). In (d), a contour diagram is created by applying* $(1 + \sin(k * f))/2$ *to the value at each pixel before mapping to grayscale. The area inside the zero iso-contour,* $f < 0$, *is hilighted in red.*

The surfaces defined so far have all had an iso-value $v$ of 0. This need not always be the case, however the description and implementation of many algorithms can be simplified by assuming that the surface lies on the 0 iso-contour. For example, points on the surface are often referred to as *zeroes* of $f(\mathbf{p})$. Implicit surfaces defined with non-zero $v$ can be rewritten in this form as $f(\mathbf{p}) - v = 0$.

Non-zero iso-values can be used with distance fields to define *offset surfaces*, where $f_{\mathsf{T}}(\mathbf{p}) = v$ and $v > 0$. Here $v$ is the distance from the offset surface to $\mathsf{T}$. Note that if $\mathsf{T}$ is a closed surface, then $f_{\mathsf{T}}(\mathbf{p}) = v$ defines two new surfaces - one "inside" the old surface, and another "outside". If $\mathsf{T}$ has no interior, as is the case for points, curves, open surfaces, and solids, then only one offset surface is defined. In this case, $f_{\mathsf{T}}(\mathbf{p}) = v$ is referred to as a *skeletal primitive*.

## 2.2 Implicit Volumes

If an implicit surface $\mathcal{S}$ is closed, then it divides space into 3 sets of points - a finite interior volume $\mathcal{V}$, the surface $\mathcal{S}$, and an infinite exterior volume $\bar{\mathcal{V}}$ [88, 103, 104, 79]. For simplicity, it will be assumed that $\mathcal{S} \subset \mathcal{V}$. This set of points can be defined as

$$\{\mathcal{V} : f(\mathbf{p}) \leq v\} \tag{2.3}$$

where $v$ is the iso-value used to define $\mathcal{S}$. This inequality defines an *implicit volume*. Note that the sign of the inequality is dependent on $f(\mathbf{p})$, and can always be flipped to define $\bar{\mathcal{V}}$. This

definition provides a trivial point containment test [104] - the value of $f(\mathbf{p})$ determines where $\mathbf{p}$ is inside or outside the surface.

Not all implicit surfaces are necessarily implicit volumes. For example, the distance fields of the previous section do not define implicit volumes, as Equation 2.3 is true both inside and outside the surface. Self-intersecting implicit surfaces also cannot define implicit volumes, as the notion of inside and outside is undefined in the self-intersecting regions. However, if $\mathsf{T}$ is a closed, non-self-intersecting surface, then a *signed distance field* can be defined where $f_{\mathsf{T}}(\mathbf{p}) < 0$ if $\mathbf{p}$ lies inside $\mathsf{T}$, and $f_{\mathsf{T}}(\mathbf{p}) > 0$ outside.

In practice, creating a signed distance field for a non-trivial surface can be very difficult. The surface must support both a *distance query* and a *point containment* query to define a signed distance field, and hence an implicit volume. One of the advantages of skeletal primitives, which always define implicit volumes, is that they only require the underlying skeletal elements to support a distance query [124, 120].

## 2.3 Solid Modeling

Early 3D modeling systems [88, 85, 86] involved the construction of complex 3D shapes using simpler volumes such as spheres, cubes, and cylinders. These simple volumes were composed using *boolean operations* such as *union* ($\cup$), *intersection* ($\cap$), and *subtraction* or *difference* ($\setminus$). This constructive style of modeling is known as *Solid Modeling*, or *Constructive Solid Geometry* (CSG).

Boolean CSG operations are essentially set operations on 3D points. For example, the union of two 3D volumes is the set of points inside both volumes. In 1973, Ricci [88] introduced a very simple approach to performing set operations between implicit volumes, based on functional composition of the underlying scalar functions. The union of two implicit volumes can be defined as

$$(f_1 \cup f_2)(\mathbf{p}) = \min\left(f_1(\mathbf{p}), f_2(\mathbf{p})\right) \tag{2.4}$$

This composition *operator* produces a new scalar field defining a new implicit volume. Intersection and difference can be computed using similar methods.

The power of Ricci's operators is that they are *closed* under the space of all possible implicit volumes, meaning that an application of an operator simply produces another scalar field defining another implicit volume. This new field can be composed with other fields, again using Ricci's operators. Equation 2.4 will always produce the exact union of two implicit volumes, regardless of how complex they are. Compared with the difficulties involved in applying boolean CSG operations to B-rep surfaces, solid modeling with implicit volumes is incredibly simple. Examples of Ricci's CSG operators are shown in Figure 2.2.

## 2.4 Blending

Solid modeling is not limited to CSG. Another useful class of operation is the construction of smooth transitions between two surfaces. These transitions are often known as *blends*. One standard type of blend is the "rolling-ball" blend, where the blend surface is defined by sweeping


Figure 2.2: *Ricci CSG Operators Union (a), Intersection (b) and Difference (c), applied to circles in 2D and spheres in 3D.*

a sphere along a path such that it just touches both surfaces for all points of the path (essentially "rolling" the sphere around the joint).

Rolling-ball blend surfaces are difficult to compute. However, functional blend operators similar to Ricci's CSG operators can be defined for implicit volumes. Ricci's blend operator ($\uplus$), defined as

$$(f_1 \uplus f_2)(\mathbf{p}) = \left(f_1(\mathbf{p})^{-s} + f_2(\mathbf{p})^{-s}\right)^{\frac{1}{-s}} \tag{2.5}$$

produces a new blended volume (Figure 2.3). The parameter $s$ controls the smoothness of the blend (as $s \to \infty$, $\uplus \to \cup$). This blend operator has the same attributes as the CSG operators, namely that it is independent of surface complexity and that it produces a blended volume which can be treated as any other implicit volume.

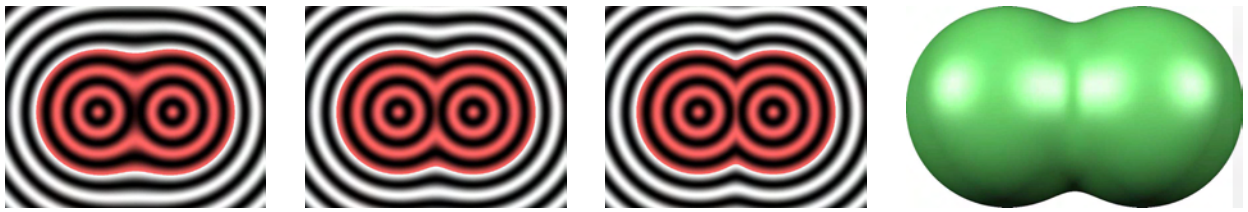



Figure 2.3: *Ricci Blending Operator applied to two circles with blending parameter* 6 *(a),* 12 *(b), and* 24 *(c). A 3D example is shown in (d).*

Blinn [24] introduced another type of implicit volume which was based on the concept of *equipotential surfaces* in molecular physics. Blinn was trying to visualize molecular structures using electron density clouds. The electron density cloud is defined by summing the electron density fields of the individual atoms in a molecule. These fields are defined by applying a *potential function* to a distance field generated from a point $\mathbf{p}$. The specific potential function was $e^{-rd^2}$, where $d$ is the Euclidean distance to $\mathbf{p}$. This field has the value 1 at $\mathbf{p}$ and smoothly decreases to 0 at $\infty$. The iso-surface is defined by some non-zero $v$. When the fields generated by all the points are summed, a smooth blend surface is created.

Blinn's blobby molecules were essentially implicit skeletal primitives (Section 2.1). His Gaussian potential function can be applied to any skeletal primitive defined by a distance field. These primitives can be blended using the *additive blend* operator, $+$:

$$(f_1 + f_2)(\mathbf{p}) = f_1(\mathbf{p}) + f_2(\mathbf{p}) \tag{2.6}$$

Note that this operator is simply a specific case of Ricci's blend, with $s = 1$.

## 2.5 Bounded Fields

While Blinn's Gaussian skeletal primitives were successful at modeling electron density clouds, they are less useful for interactive modeling. The issue is that the field function $e^{-rd^2}$ has *infinite support*, meaning that its value is non-zero everywhere in space. Since the surface is defined by summation of all the point primitives, moving any one of them will change the entire surface. For this reason, the primitives are said to have *global influence*.

Global influence is very un-intuitive for interactive modeling, and particularly for implicit modeling, where the underlying scalar fields are not visible to the designer. When a primitive is changed, the expected behavior is that it will only affect the local area. If a primitive has global influence, then changing it locally can affect distant portions of the surface, causing much confusion and frustration.

Nishimura [74] and Wyvill et al [124] introduced new polynomial field functions that had finite or *compact* support, meaning that the field value is uniformly 0 outside some finite distance from the skeletal primitive. Essentially, the non-zero field values are *bounded* within some geometric region, and for this reason the skeletal primitives are often said to have *bounded fields*.

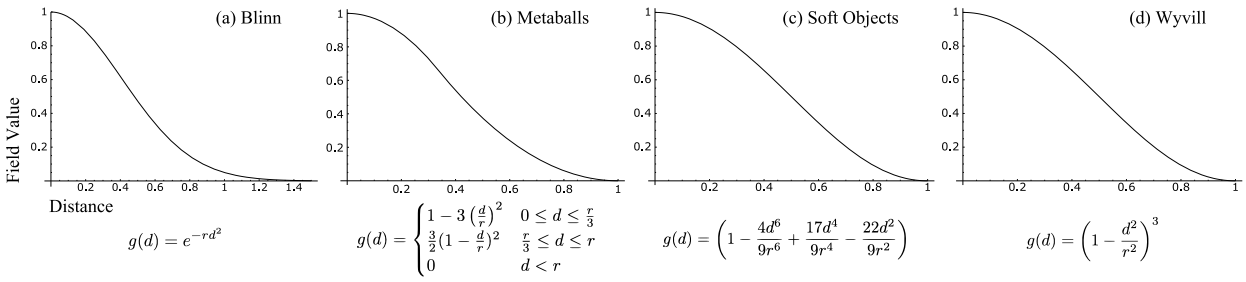



Figure 2.4: *Potential functions. (a) Blinn's Gaussian or "blobby" function, (b) Nishimura's "metaball" function, (c) Wyvill et al's "soft objects" function, and (d) the Wyvill function.*

The bounded fields used by Nishimura's *Metaballs* and Wyvill et al's *Soft Objects* are specific cases of a general type of skeletal primitive, defined by composition of a potential function $g$ and a distance field:

$$f(\mathbf{p}) = g \circ d_{\mathsf{T}}(\mathbf{p}) \tag{2.7}$$

This separation is very useful for stating guarantees about scalar fields. If $g$ has compact support, and $\mathsf{T}$ is finite, then $f$ will necessarily be bounded. Analysis of some properties of $f$ is also simplified - for any convex skeleton, the blending properties of $f$ are completely determined by $g$. The resulting bounded fields have local influence and hence preserve a sort of "principle of least surprise" that greatly improves the usability of constructive implicit modeling.

Most distance field composition operators can be rewritten to work on bounded fields. For instance, the Ricci union (Equation 2.4) operator can be converted simply by using max instead of min (and vice-versa for intersection). The Ricci blend (Equation 2.5) is converted by changing

the sign on the blending parameter. CSG difference is more problematic. For bounded fields, one way to define the difference of $f_1 - f_2$ is $\min(f_1, 2\,v - f_2)$, where $v$ is the iso-value. This operator is problematic because it can produce fields with non-zero values far from the surface. An example is shown in Figure 2.5e, where one skeletal point primitive has been subtracted from another. The ring of positive field values in the right side of the image can produce undesirable blending artifacts (See Figure 3.1) and may prevent gradient walks from converging on the surface. Difference operators based on R-functions (modified to support bounded fields) can reduce the magnitude of these external values, but not eliminate them completely [51].

Clearly, operators acting on bounded fields should produce fields that are also bounded. However, this is a relatively loose requirement - a "bounded field" that stops just short of infinity is still bounded, but cannot really be said to provide local influence. A reasonable constraint (which holds for all the operators described thus far) is that the output bounds be contained within the union of the input bounds.


Figure 2.5: *Field images of bounded skeletal point primitive (a,b), bounded Bezier curve primitive (c), Ricci blend of point primitives (d), and CSG Difference of point primitives (e).*

## 2.6 Continuity and Manifolds

Many properties of an implicit surface emerge from the mathematical properties of the underlying scalar field $f$. One critical property is that of *continuity*. While continuity comes in many forms, the most basic level of scalar field continuity is $C^0$ continuity. In mathematical terms, a $C^0$-continuous field $f$ is a field where as the distance between two 3D points $\mathbf{p}$ and $\mathbf{q}$ decreases to zero, so does the difference between their field values $f(\mathbf{p}) - f(\mathbf{q})$.

$C^0$ field continuity is one of the basic pre-requisites for a variety of other mathematical statements that can be made about $f$. One desirable property for solid modeling is that any iso-surface of $f$ be a *closed simple manifold*. In terms of 2D surfaces embedded in three dimensions, a surface is a manifold if some infinitely-small neighbourhood around any point on the surface "looks" like a disc [89]. If a manifold has no boundaries (edges where the surface lies only on one side, such as the edge of a triangle), then it is said to be *closed*. Closed manifolds which have no self-intersections are said to be *simple*. In 3 dimensions, a closed simple manifold has a well-defined interior and exterior, and hence a well-defined volume. This is nececssary for solid modeling, as only surfaces with well-defined volumes can be considered solids (Section 2.2). To

simplify exposition, a surface will be said to *be manifold* when it is a closed simple manifold. Note that the iso-surface may include multiple un-connected surfaces, in this case each surface is analyzed independently.

Scalar fields can be classified with respect to continuity and manifold properties. Binary fields are not $C^0$; the field values "jump" between 0 and 1 but do not make smooth transitions. Distance fields are $C^0$, however the manifold properties of the zero iso-contour are determined entirely by the geometric skeleton. The skeleton may not be closed, may have self intersections, or may not even be a surface. In any of these cases, the zero iso-contour is not manifold[1].

Offset surfaces of distance fields are manifold, despite the degenerate interior boundaries that can occur at certain iso-values. For example, a degenerate point occurs at the center of the distance field of a sphere when the iso-value is equal to the sphere radius. These interior iso-surfaces are infinitely thin and can be removed using a topological process known as *regularization* [85] which removes all extraneous lower-dimensional artifacts. Regularization is based on classifying points as either *boundary*[2] or *interior*, where interior points are those points inside the volume. Hence, regularization cannot repair volumes with self-intersections because the notion of "inside" is no longer mathematically well-defined.

Since offset surfaces of distance fields are manifold, all non-zero iso-contours of skeletal implicit primitives are manifold. Unlike distance fields, the zero iso-contour is not used in constructive modeling with bounded skeletal primitives. Hence, for modeling purposes, skeletal primitives always produce manifold iso-surfaces. This is a key distinction between skeletal primitives and distance fields.

Since a constructive modeling system will involve functionally combining implicit surfaces, the mathematical properties of composition operators must also be analyzed. Consider a general binary operator $g(f_1, f_2)$. Iso-surfaces of $g(f_1, f_2)$ can only be manifold if they are $C^0$, so it is essential that $g$ preserve $C^0$ continuity. As noted by [17], $g$ defines a 2D scalar field which maps from $\mathbb{E}^2$ to $\mathbb{R}$. If $f_1$, $f_2$, and $g$ are all $C^0$, then the scalar field produced by $g(f_1, f_2)$ is guaranteed to be $C^0$. While $C^0$ continuity is a necessary condition for manifold iso-surfaces, it is not sufficient. If $f_1 = v$ and $f_2 = v$ are manifold, then for most of the standard operators, $g(f_1, f_2) = v$ is manifold. However, the necessary conditions for $g$ to preserve manifold iso-contours are unknown in computer graphics.

Based on these mathematical properties, certain statements can be made about constructive modeling with implicit surfaces. One key property is that if the primitives in use are $C^0$ and have manifold iso-surfaces, and all operators in use are $C^0$ and preserve manifold iso-surfaces, then surfaces created with the system will be $C^0$ and manifold. Using this set of primitives and operators, it is *impossible* to create an implicit model that does not define a volume. This is very useful for interactive volume modeling, as it means that the designer cannot "break" the system and produce an invalid model.

---

[1] Again, "manifold" is used here as a shorthand here for "closed simple manifold"

[2] The term boundary is used here in a point-set-topology sense [115], boundary points are members of the closures of both the interior and exterior sets of points. For a valid implicit volume, boundary points are defined by $f(\mathbf{p}) = v$ and interior points by $f(\mathbf{p}) < v$.

## 2.7 $C^1$ Continuity and The Gradient

$C^0$ continuity, which ensures that there are no "jumps" in a function, is the most basic form of continuity. Higher-order continuity is defined in terms of derivatives of functions. For example, if the derivative of a one-dimensional scalar function is continuous, then the scalar function has first derivative or $C^1$ continuity.

In the case of a 3D scalar field $f$, the first derivative is a vector function known as the *gradient*, written $\nabla f$ and defined as:

$$\nabla f(\mathbf{p}) = \left\{ \frac{\partial f(\mathbf{p})}{\partial x}, \frac{\partial f(\mathbf{p})}{\partial y}, \frac{\partial f(\mathbf{p})}{\partial z} \right\} \tag{2.8}$$

If $\nabla f$ is defined at all points, and the three one-dimensional partial derivatives are each $C^0$, then $f$ is $C^1$.

Surfaces have a related but different notion of $C^1$ continuity. Informally, $C^1$ surface continuity requires that the *surface normal* vary smoothly over the surface. The surface normal at $\mathbf{p}$ is the unit vector perpendicular to the surface. If no unique surface normal can be defined, such as on the edge of a cube, then the surface is not $C^1$ along the edge. Note that self-intersections are also not $C^1$, so a closed $C^1$ surface is necessarily manifold 2.6. For points on an implicit surface $f(\mathbf{p}) = v$, the surface normal can be computed by normalizing the gradient vector $\nabla f$.

Surface continuity has direct implications for computer graphics. Shading on a surface at a point $\mathbf{p}$ is largely controlled by the surface normal. If the surface is not $C^1$, the gradient (and hence the normal) can change direction significantly at the discontinuity. At the $C^1$ discontinuity, a shading artifact will appear when the surface is rendered (Figure 2.6b). Hence, $C^1$ continuity is very desirable for modeling smooth surfaces.

If $f$ is $C^1$, then any iso-surface of $f$ is $C^1$. Unsigned distance fields are never $C^1$. This is easily shown in 1D, the "distance function" for a point at the origin is simply the absolute value function, which is not $C^1$. Hence, the distance field for any 3D skeleton will not be $C^1$ at the skeleton. Signed distance fields are $C^1$ if the skeleton is convex. If the skeleton is non-convex, then some points in space are equidistant to multiple points on the skeleton. There is no unique solution to Equation 2.2, and $C^1$ discontinuities occur in both signed and unsigned fields. For example, a $C^1$ discontinuity occurs at the center of the distance field for a sphere.

Skeletal implicit primitives are created by applying a potential function to an unsigned distance field (Equation 2.7). Although the distance field is never $C^1$ at the skeleton, these discontinuities can be removed by using a suitable potential function [9]. Specifically, if the first derivative of the 1D potential function is 0 at the origin, then the skeletal primitive becomes $C^1$ on the skeleton. However, $C^1$ discontinuities in the distance field due to non-convex skeletons are still present in the skeletal primitive field. Note that if the potential function is bounded, it must also have a zero first derivative when it reaches 0, and must be $C^1$ in the non-zero interval, for any skeletal primitive to be $C^1$.

As with $C^0$ continuity, no discussion is complete without considering the continuity of operators. Again, if $f_1$ and $f_2$ are $C^1$, and $g$ is $C^1$, then $g(f_1, f_2)$ is necessarily $C^1$. However, analysis of operators is complicated by the fact that it is sometimes desirable to create a $C^1$ discontinuity. This case occurs whenever a crease in the surface is desired. For example, a cube is not $C^1$

because tangent discontinuities occur at each edge. To create creases using $C^1$ primitives, the operator must introduce $C^1$ discontinuities, and hence cannot be $C^1$ itself.

A common type of operator which must create creases are CSG operators, such as union and intersection. However, the manner in which $C^1$ discontinuities are introduced into the field is quite important. For example, the $min()$ union operator (Equation 2.4) creates $C^1$ discontinuities at all points where $f_1(\mathbf{p}) = f_2(\mathbf{p})$. When applied to two spheres, the discontinuities produced by this union operator result in a crease on the surface (Figure 2.6a), which is the desired result. However, the discontinuities extend into the field outside of the surface, which is not visible in this image. If a blend is then applied to the result of the union, the $C^1$-discontinuous plane in the field produces a shading discontinuity (Figure 2.6b). To avoid this problem, CSG operators have been developed [17] which are $C^1$ at all points except those where $f_1(\mathbf{p}) = f_2(\mathbf{p}) = v$. Hence, creases are only introduced at the $v$ iso-surface, and the shading discontinuity in the blend surface is removed (Figure 2.6c).

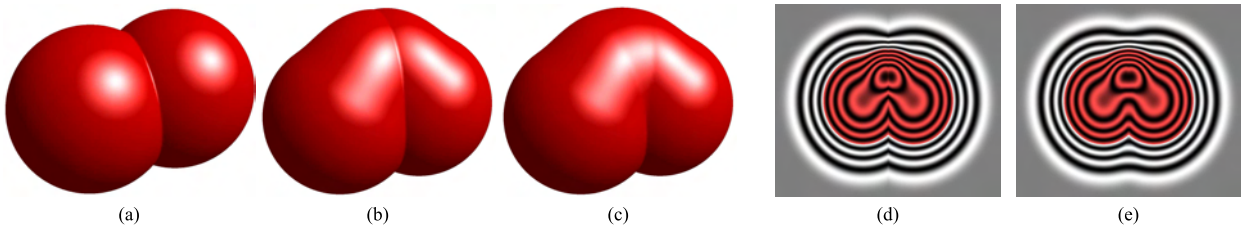



Figure 2.6: *Point blended to two union of two spheres (a) using $C^0$ Ricci union operator (b) and $C^1$ Barthe union operator (c). The Ricci union has a $C^1$ discontinuity plane which shows up as a crease through the middle of the contour diagram (d). The Barthe contour diagram (e) is smooth except at the desired surface crease.*

## 2.8 Higher Order Continuity and Perceptual Discontinuities

Higher-order continuity is important in many situations. For instance, $C^2$ continuity, also known as *curvature* continuity, is a requirement in many engineering applications. The notion of continuity is generalized as $C^n$ continuity, with $C^\infty$ being a desirable goal. For instance, Blinn's Gaussian implicits are $C^\infty$.

While many skeletal implicit primitives have higher-order continuity, formulating continuity-preserving operators is more difficult. Particularly challenging is the construction of CSG operators, which must introduce $C^0$ creases but also maintain higher-order continuity away from the creases. For example, Barthe's operators [17] achieve only $C^1$ continuity.

However, with regards to *perceived* surface smoothness, continuity is necessary but not sufficient. Continuity is a mathematical property of functions. A surface may be $C^\infty$ continuous, but appear to have a crease, because of a very high-frequency region in the underlying scalar field. An example of this situation is shown in Figure 2.7d, where two surfaces are joined with a very sharp blend, and then blended with another. Although there appears to be a crease in the final surface, like the $C^1$ discontinuity in Figure 2.6b, the underlying field is in fact $C^2$.

Unfortunately these *perceptual* discontinuities are largely determined by subjective human judgement. For example, the apparent crease in Figure 2.7d may disappear when examined close-up. This viewer-dependence hinders the construction of mathematical tools for analyzing perceptual discontinuities. Still, there are situations where such high-frequency surface variations are likely to occur. Perceptual discontinuites will be a factor in the analysis of the implicit surface approximation schemes in Chapters 4 and 5.

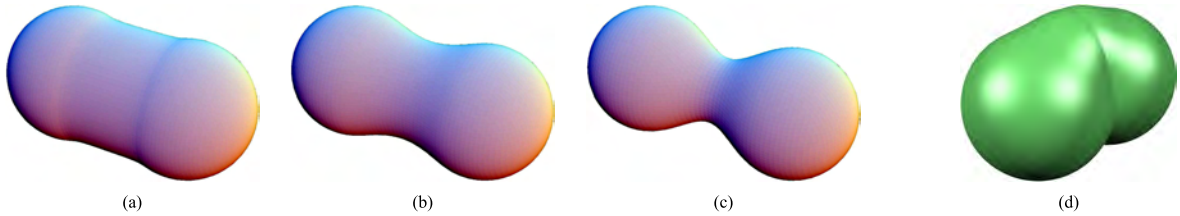



Figure 2.7: *Additive blending of two point primitives defined by potential functions* $(1 - d^2)^n$, *where* $n = 2$ *(a),* $n = 3$ *(b), and* $n = 4$ *(c). The blend becomes smoother as continuity increases. In (d), a smaller point primitive is blended to two other primitives which have been blended using Equation 2.5 with a large blending parameter. Although the field is* $C^2$, *there appears to be a discontinuity because the underlying field changes very quickly.*

## 2.9 Field Normalization and Surface Convergence

The gradient $\nabla f$ of a scalar field encodes two important pieces of information - the maximum rate-of-change of $f$, and the direction in which this maximum occurs. A desirable property of fields useful for implicit modeling is that $\nabla f$ always points "towards" the surface. If this is the case, then it is possible to reach the surface from any point by taking tiny steps in the direction of the gradient[3].

Distance fields have two properties related to the gradient which are very useful. First, in a distance field, $\nabla f$ always points towards the nearest point on the surface. Second, distance fields are *normalized* [103, 22]. A normalized field is one in which the gradient has a magnitude of 1 everywhere in the field ($\|\nabla f(\mathbf{p})\| = 1$). Given these two properties and a point $\mathbf{p}$ anywhere in space, the nearest point on the surface can be found directly - it is $\mathbf{p} + f(\mathbf{p})\nabla f(\mathbf{p})$. In fact, distance fields are not strictly normalized, because $\nabla f$ is undefined at $C^1$ discontinuities (points on the distance surface, and points equi-distant from the skeleton). In these cases none of the nearest points can be found, since the gradient is undefined. For convenience, these discontinuities will be disregarded when discussing normalization.

Normalization is a tenuous property and is very difficult to maintain, especially when combining scalar fields with $C^1$ (or greater) composition operators. Bounded fields by definition cannot be normalized, since $\nabla f$ is 0 outside the bounds. However, even inside the bounds, $f$ can only be normalized if the potential function is linear. The smoother potential functions necessary to create $C^1$ primitives (Section 2.7) rule out normalization.

[3]In some fields, the gradient always points *away* from the surface

Even if $f$ is not normalized, it is still possible to use the gradient to converge on the surface. To see how, consider a distance field scaled by 2, $f_2 = 2f_\mathsf{T}$. In this case, $\|\nabla f_2\| = 2$. Essentially, $\|\nabla f\|$ is a measure of the local scaling of the field values, so the nearest point to $\mathbf{p}$ on the surface is

$$\mathbf{p} + \frac{f_2(\mathbf{p})\nabla f_2(\mathbf{p})}{\|\nabla f_2(\mathbf{p})\|} \tag{2.9}$$

Unfortunately, on bounded fields such as those in Figure 2.5, evaluating this equation will not produce a point on the iso-surface. One consequence of normalization is that all higher-order derivatives are 0. In non-normalized fields where the higher-order derivatives are non-zero (such as in smooth bounded fields), Equation 2.9 only approximates the correct distance. However, repeated application of Equation 2.9 will eventually produce a point on the surface under some relatively weak conditions [4]. This iteration is known as *surface convergence*. Note that if the surface iso-value $v$ is non-zero, then the convergence iteration is instead:

$$\mathbf{p} + \frac{(f(\mathbf{p}) - v)\nabla f(\mathbf{p})}{\|\nabla f(\mathbf{p})\|} \tag{2.10}$$

These convergence iterations assume that $\nabla f(\mathbf{p})$ is non-zero. One drawback of bounded fields is that the gradient is zero in most of space. Hence, Equation 2.10 will only converge on the surface if the start point has a non-zero field value. In fields with infinite support, convergence occurs from any point in space.

Surface convergence is a critical tool for visualizing implicit surfaces, and in this domain it is desirable that surface convergence occur as quickly as possible. Higher normalization error in non-linear fields generally leads to a larger number of iterations of Equation 2.10 necessary to reach a given error tolerance, as each step is more likely to under-shoot or over-shoot the surface. Normalization also has other benefits [22]. For example, in a normalized field, an offset surface created by modifying the iso-value is equivalent to a distance-based offset surface. Also, normalized fields have a consistent "blending radius" which makes the results of blending operations more consistent and predictable. Tools for evaluating normalization are described in the next section.

### 2.9.1 Analyzing Field Normalization Error

Strict normalization is generally not attainable for the scalar fields used in implicit modeling. Even if the fields of primitives are normalized, few operators preserve normalization, particularly blending operators. However, since normalization is a key property for predictable implicit modeling, it is useful to be able to compare the normalization error in different fields. Normalization error at a point $\mathbf{p}$ is defined as $|1 - \|\nabla f(\mathbf{p})\||$, and a *normalization metric* over a field $f$[5] can then be defined as:

$$norm_f = \max_{\mathbf{p}} \left(|1 - \|\nabla f(\mathbf{p})\||\right) \tag{2.11}$$

[4] The convergence step is essentially a Newton iteration, which is known to be relatively stabe [84] but is only guaranteed to find the nearest local minimum. Hence, it is necessary that the field be monotonic along the path to the surface.

[5] In bounded fields, only the normalization error at points where $f \neq 0$ should be considered.

In a normalized field, $norm_f = 0$. However, unles $f$ is described by a simple equation, analytically computing $norm_f$ is not possible. The only alternative is to discretely approximate $norm_f$ by sampling over some domain. Besides the maximum, various other statistical measures can be applied to the samples to further analyze the normalization error. If the sampling is regular, a visual analysis tool called a *normalization image* can be constructed in the same way as field images (Section 2.1). Normalization images permit a more detailed evaluation of the normalization error in a particular scalar field. Some examples are shown in Figure 2.8.


Figure 2.8: *Normalization images for (a) bounded skeletal point primitive, (b) CSG difference of bounded point primitives, and (c) Ricci blend of point primitives. The error scale runs from* 0 *(black) to* $\geq 1$ *(white). The normalization image for the blended circles from Figure 2.2b is shown in (d). Since they are distance fields, the result is normalized everywhere except in the blending region.*

Normalization error is also visible in contour images. If $f$ is normalized, the spacing between contours is perfectly regular. As the normalization error increases, contour spacing changes, and can be irregular over the field. The normalization error in a bounded point primitive is clearly visible when comparing the contour image to that of a circle's distance field, as in Figure 2.5b.

## 2.10 The Scaling Problem

A fundamental issue with skeletal primitive modeling is the *scaling problem*. Consider a single point primitive, generated by the composition of some potential function with the distance field of a single point. The question is, how can the size of this point primitive be altered, or *scaled*?

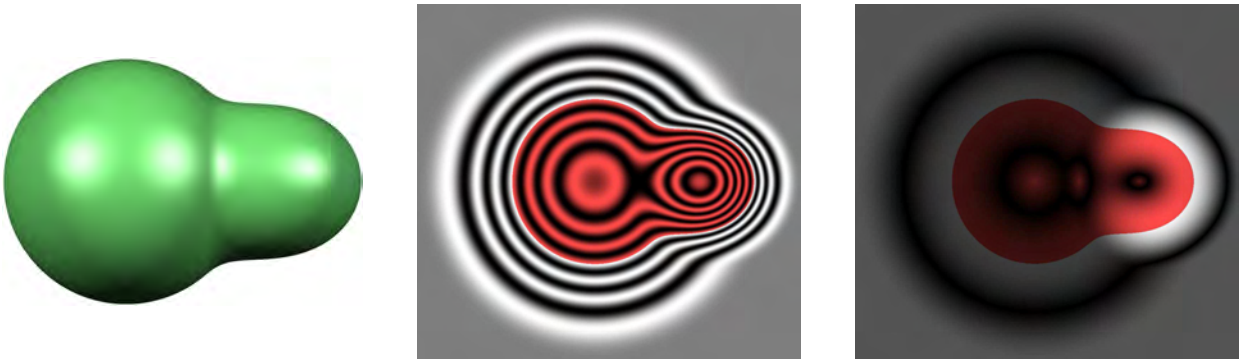
If the iso-surface is defined as $f(\mathbf{p}) = v$, then one option is to change $v$. However, consider a more complex case - two points are blended together. How can one be scaled? If $v$ is modified, both will be scaled. The only solution to scaling primitives is to modify $f$.

At this point, it is useful to introduce a specific potential function. The following potential function, $g_w$, will be used for all skeletal primitives that follow:

$$g_w(d) = \left(1 - \left(\frac{d^2}{r^2}\right)\right)^3 \tag{2.12}$$

where $r$ is the "radius" of the field and the input distance $d$ is clamped to the range $[0, r]$ [123]. Generally, an iso-value of 0.5 is used with this function. The function is $C^6$ continuous and does

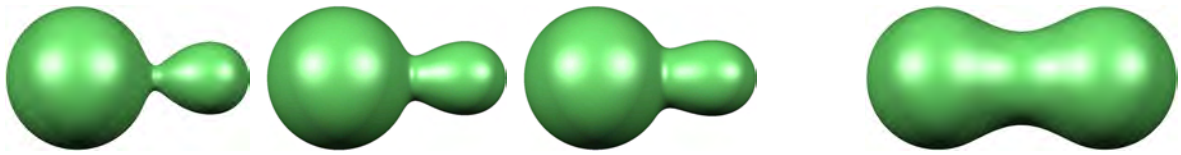
have zero-tangents at either end. A plot of $g_w$ is shown in Figure 2.4d, and an additive blend of two point primitives with $r = 1$ is shown in Figure 2.10d.



(a) (b) (c)

Figure 2.9: *The scaling problem. To create a blend between two point primitives of different radius (a), the potential function must be scaled. The resulting iso-contours of the smaller primitive are more closely spaced (b) due to higher normalization error (c). The normalization error increases as the primitive becomes smaller.*

By modifying $r$, the desired scaling result in Figure 2.9 is achieved, however the region of the field containing the scaled primitive has significantly more normalization error than the region of the un-scaled field. The difference in normalization error has many undesirable effects. An immediately visible problem is that it results in variable blending behavior when primitives with different scaling factors are blended (Figure 2.10). Another issue is that the convergence iteration (Equation 2.10) will require more steps to reach the same level of accuracy. This is the *scaling problem.*



(a) (b) (c) (d)

Figure 2.10: *The sequence of images in (a)-(c) show how the scaling problem creates blends which are less smooth than when the primitives all have the same scale (d).*

There are several possible approaches to mitigating the scaling problem. The first is to modify the potential function. Consider the plots in Figure 2.11a, showing $g_w$ with radius 1 and 0.5. At values $> 0.5$, the scaled function can be improved by "flattening" the 1D function, such that the value at $g_w(0)$ is less than 1 (Figure 2.11b). However, The scaled function *must* decrease from 0.5 to 0. To improve this region, the function must extend further along the x axis (Figure 2.11c). However, this "long tail" causes the field to extend further out in space from the iso-surface, increasing the support region of the primitive and breaking the principle of local support (Section 2.5). Blending with the resulting field can be very un-intuitive.

There is currently no known solution to the scaling problem. There have been attempts to mitigate the problem [122, 17], however these works generally only improve the blending surface and do not actually reduce normalization error. When the resulting fields are used in another blend, the issues re-appear. It may be that the scaling problem is a fundamental restriction on skeletal implicit modeling. Any solution will likely involve global iso-surface-preserving non-linear field transformations, an area which has received little attention to date. The normalization attempts described by [22] may be a reasonable starting point.

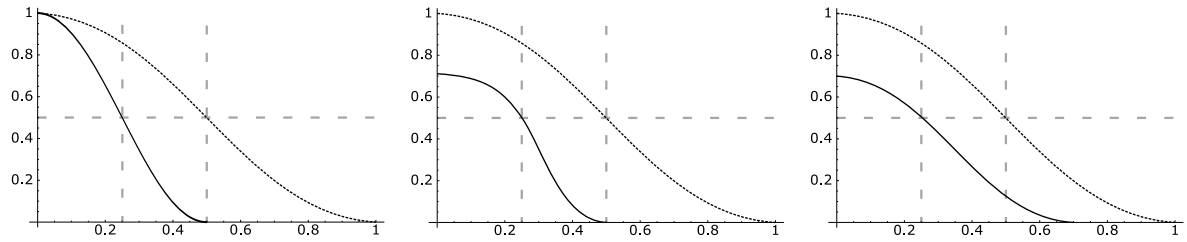



Figure 2.11: *The scaled potential function (solid line) in (a) has significantly higher normalization error than the un-scaled potential function (dashed line). Reducing the height of the scaled function (b) improves normalization error on the interior of the primitive, but does not help on the exterior. Extending the field radius (c) reduces exterior normalization error, but at the cost of local influence.*

## 2.11 Chapter Summary

Although the basic definition of an implicit surface is seemingly quite simple, a wide range of non-trivial details are quickly encountered. In this chapter, the major concepts relating to shape modeling with implicit surfaces and volumes have been introduced. Solid modeling operations such as CSG and blending permit the composition of arbitrarily complex shapes. Issues related to bounded fields, $C^0$ and $C^1$ continuity, and field normalization have been discussed. Recognition of perceptual discontinuities, analysis of the scaling problem, and the new tools for analyzing normalization error have not appeared in the literature previously.

# Chapter 3

# A Taxonomy of Implicit Surfaces

*A series of functional properties, introduced in the previous chapter, are chosen for use in a classification of different implicit modeling systems. Major hierarchical implicit modeling frameworks and other recent implicit surface modeling schemes are classified based on these properties. This taxonomy is used to select a method for use in an interactive modeling system. The hierarchical BlobTree implicit modeling framework is chosen.*

## 3.1 Introduction

As noted in the previous chapter, any scalar function in $\mathbb{E}^3$ inevitably defines some implicit surface. However, depending on the task, certain functions are "more useful" than others. Unfortunately there is little consensus on which types of implicit surfaces are appropriate for interactive modeling. In the following sections, an attempt is made to classify a variety of existing implicit surface techniques, based on their mathematical properties. The goal of this taxonomy is to support comparisons between different methods, with an eye towards selecting a implicit modeling framework that will best support interactive modeling.

As with any taxonomy, the properties which are chosen to differentiate different implicit surface schemes are somewhat at the whim of those creating the taxonomy. Likewise, the grouping of different implicit surface techniques into the considered categories is somewhat arbitrary, but necessary to make the task tractable. Even more problematic is the fluidity of implicit surfaces. Attributes of a binary nature, such as whether or not a function is bounded (Section 2.5), are quite rare. Many other important properties, such as normalization error considerations, do not permit such easy categorization. In these cases the statements about different techniques must be qualified with often imprecise terms.

In short, the following classification is proposed only as an initial attempt, and was designed with interactive modeling in mind. For the task of selecting a technique for interactive modeling with implicit surfaces, it has served its purpose. These biases must be considered when applying this taxonomy to other domains.

Another caveat is that the properties mentioned below are largely based on mathematical aspects, rather than algorithmic issues. This limitation of scope has been enforced due to the complexity of classifying algorithmic properties. For example, some implicit modeling techniques have special properties which can be taken advantage of to provide faster visualization [7]. Depending on the application, these fast techniques may not produce the desired accuracy or visual properties. Which specific visualization problems should be selected as most important? Furthermore, any discussion of visualization speed is intimately tied to implementation quality and computing hardware. In most cases, source code is not available, and any attempts to normalize computation times between 10-year-old computers and state-of-the-art networked clusters are likely to be erroneous. In short, detailed analysis of these issues for a single implicit repre-

sentation scheme is non-trivial, and attempts to generalize across the range of implicit surface techniques considered in this chapter would undoubtedly fail. Hence, this taxonomy is limited to mathematical issues of shape representation, as outlined in Section 1.3.

## 3.2 Classification Properties

**Continuity** The notions of $C^0$ and $C^1$ continuity, as well as the generalization to $C^k$ continuity, were discussed in Sections 2.6 and 2.7. Field continuity is a critical property for determining when an implicit surface technique is appropriate for a given application. For constructive implicit modeling frameworks, continuity of Boolean CSG (Section 2.3) and blending operators (Section 2.4) will also be considered.

**Field Support** As discussed in Section 2.5, some fields have finite support, meaning that their non-zero values are contained within some finite bounding region. This property is often desirable for computational reasons, however in interactive modeling a more critical implication is that of local influence. Of course, an unbounded field can always be explicitly bounded, so statements made about this property will generally refer to the support of the field as it is commonly used in the literature.

**Analytic Surfaces** Many types of implicit surfaces are defined based on a given set of 3D point samples $\{\mathbf{p}_i, f_i\}$, such that they either interpolate or approximate the values $f_i$ at $\mathbf{p}_i$. These techniques generally cannot represent simple analytic surfaces such as a sphere or cylinder. Techniques which support these types of basic surfaces will be said to be *analytic*.

**Creases** The modified definition of $C^1$ continuity given in Section 2.7 permits $C^1$ discontinuities on the surface to represent creases, which are critical in 3D modeling. This is a binary property for most implicit surface repesentations, however in some cases there is limited support for certain types of creases (such as the explicit creases created in MPU methods).

**Guaranteed Volumes** Most types of implicit surfaces are also capable of defining implicit volumes (Section 2.2). However, only some methods are *guaranteed* to generate surfaces without self-intersections, and hence well-defined volumes. Further, for constructive implicit modeling frameworks, this property must be preserved regardless of the operators and primitives involved (Section 2.6).

**Expected Volumes** For some types of bounded implicit surfaces defined based on point samples, internal iso-contours can occur which result in a well-defined but undesirable volume. These internal iso-contours are highly problematic in solid modeling contexts. Since the expected volume is somewhat subjective, for point-sampled surfaces the general requirement will be that all iso-contours lie on or very near to the initial samples. In constructive implicit modeling the operators must be considered; the resulting volume will be expected if all iso-contours are connected to one of the iso-contours of the input fields.

**Normalization Error** Normalization error was introduced in Section 2.9. In general, fields with low normalization error lead to well-behaved and efficient algorithms, particularly with respect to surface-convergence steps. Low normalization error also results in blending behavior that is more consistent, allowing designers to more easily manipulate blend surfaces. Normalization error is highly variable and currently cannot be distilled down into a simple categorization - each case must be analyzed independently.

## 3.3 Constructive Modeling Frameworks

Implicit surface modeling techniques generally take one of two paths. The first is to construct complex shapes by composing of simpler shapes (primitives) [88, 124, 120, 4, 106, 18, 58, 11]. The second is to construct complex shapes directly, generating a single scalar field from some given set of surface or volume samples [92, 35, 113, 34, 72, 114, 56, 73, 75, 87, 105]. Systems taking the first approach will be referred to as *constructive modeling framework*, while the second will be termed *global field* approaches. This distinction is somewhat artificial, as any global field technique can be used as a primitive in constructive systems [18]. However, there is a clear conceptual distinction, which is supported by the literature - works on global implicit representation rarely analyze the resulting fields for compatibility with constructive frameworks. For example, many efficient techniques for interpolating 3D point sets produce additional iso-contours [72, 75, 87]. While not a significant problem for their intended application (surface reconstruction from range scans), these extra iso-surfaces have serious implications for solid modeling.

### 3.3.1 Distance Fields, R-Functions, and F-Reps

Procedural modeling with distance fields is perhaps the oldest application of implicit volume modeling [88]. However, the introduction of *R-functions* [90] for shape modeling was a major advance [104]. R-functions provide a robust theoretical framework for boolean composition of real functions [103, 79], permitting the construction of $C^n$ CSG operators. These CSG operators can be used to create blending operators simply by adding a fixed offset to the result [79]. Although these blending functions are no longer technically R-functions [103], they have most of the desirable properties and can be mixed freely with R-functions to create complex hierarchical models. These R-function-based blending and CSG operators will be referred to as *R-operators*.

While any partition of the real line does in theory have a set of associated R-operators, the partition used most frequently in computer graphics is such that the zero iso-contour is taken as the surface, positive values inside, and negative values outside [103, 104, 79]. This implies that all fields have infinite support, and all operators necessarily have global influence. The term *F-Rep* is often used to refer to this class of implicit solid models [4].

As noted in Section 2.6, any surface can be represented as the zero iso-contour of a distance field. While this does mean that both analytic surfaces and creases are inherently supported, it is also possible to describe non-manifold surfaces with F-Reps. Hence, no volumetric guarantees can be made. R-operators exist for both signed and unsigned distance fields. In the unsigned case, volumes are not defined, and furthermore the composition of manifold surfaces can produce

non-manifold results. With signed fields, if all initial surfaces are manifold, volume guarantees can be made.

In general, composition with R-operators produces fields with very poor normalization, even if the initial fields are normalized [17, 22, 51]. Recent work has focused on improving normalization [17, 22], however the results are limited (see Section 5.2).

R-operators do not inherently require distance fields. Any distance-like fields (fields with the desired iso-contour at $v = 0$) can be composed. Hence, the F-Rep framework can be applied to a wide variety of implicit surfaces.

### 3.3.2 BlobTree Modeling

Skeletal primitives (Section 2.5) are defined as offset surfaces in a potential field, and have a nonzero iso-value. In addition, skeletal primitives with finite skeletons are bounded. The R-operators used in F-Rep systems cannot be applied to these fields, and hence an alternate set of blending operators has been developed. While Shapiro [103] states that a set of R-functions must exist for such fields, they have not been extensively studied in the literature[1]. Some recent attempts to adapt CSG R-operators have been made [16, 51], however these operators simply convert the bounded field to an unbounded F-Rep, apply standard R-operators, and explicitly bound the result.

The BlobTree [120] hierarchical modeling framework encapsulates techniques for constructive modeling with skeletal primitives. In general, skeletal primitives have variable continuity depending on the potential function, although low-degree polynomials ($k \leq 6$) are most common. Regardless, Boolean CSG operators which produce higher than $C^1$ fields are currently unknown [19]. Analytic surfaces can be represented, creases can be represented exactly, and strong volume guarantees can be made. However, existing CSG difference operators for bounded fields [88, 120, 16, 19, 51] leave non-zero values far from the surface, which can result in unexpected additional iso-surfaces after additive blending is applied (Figure 3.1).

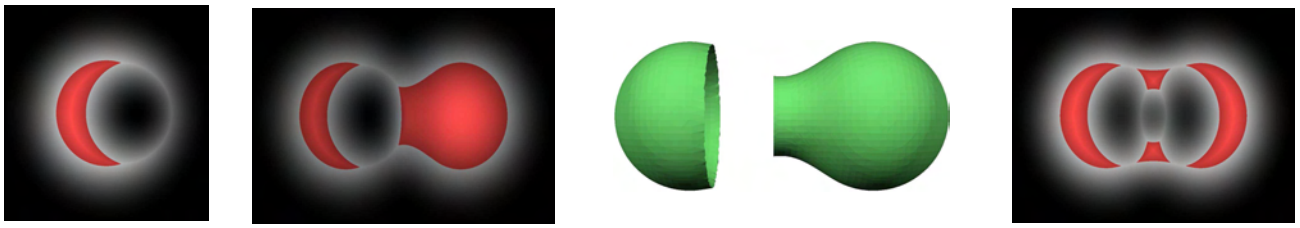



Figure 3.1: *A CSG Difference operation on two point primitives (a) leaves a ring of non-zero values far from the surface. When combined with additive blending, These values can cause unexpected behavior (b,c) and additional unwanted iso-contours (d).*

Significant scale differences beween skeletal primitives will result in high normalization error

[1]Ricci's min and max CSG operators [88] do appear to be R-functions, however they are rarely identifed as such. Also, CSG operators based on R-functions are described by Wvill et al [120], however it is unclear whether or not they are actually R-functions.

(Section 2.10). Repeated additive blending also increases normalization error, particularly for volumes that overlap significantly. Attempts to mitigate this problem using global field transformations have met with limited success [51].

The BlobTree framework does not inherently require that all primitives be skeletal primitives. The only condition is that values inside the surface be larger than the nonzero iso-value, and that the field drops off to 0 at some distance outside the surface. The sweep templates generated in Chapter 5 are one example of non-skeletal bounded fields.

### 3.3.3 Convolution Surfaces

Convolution surfaces, first introduced by Bloomenthal and Shoemake [28], are produced by convolving a geometric skeleton $\mathsf{S}$ with a *kernel* function $h$. Hence, the value at any position in space is defined by an integral over the skeleton:

$$f(\mathbf{p}) = \int_{\mathsf{S}} g(\mathbf{r})\, h(\mathbf{p} - \mathbf{r})\, d\mathbf{r} \tag{3.1}$$

Any compactly-supported function can be used as $h$, see [107] for a detailed analysis of different kernels.

Like skeletal primitives, convolution surfaces have bounded fields. Since they are compatible with the BlobTree modeling framework, they are in some sense a special case of skeletal primitives. However, composition of convolution surfaces is usually defined by composition of the underlying geometric skeletons, rather than functional composition. The reason for this is to avoid the bulges that tend to occur when composing multiple skeletal primitives with additive blending. The surface resulting from convolution of the combined skeleton does not have bulges, and the field is continuous even if the combined skeleton is non-convex. Volumes are preserved, and normalization error is entirely dependent on the convolution kernel. However, because they are always offset a fixed distance from a geometric skeleton, convolution surfaces cannot represent creases or non-smooth analytic surfaces.

## 3.4 Uniformly-Sampled Discrete Volume Datasets

Given a set of samples $f_i$ of some $f(\mathbf{p})$ distributed at uniform intervals, a scalar field $\widetilde{f}$ can be constructed which approximates $f$ [35]. The sets of samples are often referred to as *volume datasets*. The properties of the constructed scalar field hinge on the technique used to convert the $f_i$ to $\widetilde{f}$. Generally, a *reconstruction filter* is applied to the samples in some neighbourhood of $\mathbf{p}$ to produce a continuous field using interpolation. Although $C^k$ spline-based reconstruction filters can be used, $C^0$ to $C^2$ are most common [69, 71]. Field support is arbitrary - the sample set is assumed to be finite, so the value of $\widetilde{f}(\mathbf{p})$ outside of the sample set is application-defined. As with any sample-based scheme, analytic surfaces can only be approximated. Similarly, arbitrary creases cannot be represented. Volume and normalization properties are completely dependent on the sample set, no general statements can be made in these cases. Implicit surfaces created using volume datasets may be compatible with either the F-Rep or BlobTree modeling frameworks, and can be used to define Convolution Surfaces [101].

### 3.4.1 Adaptive Distance Fields

The implicit surfaces generated by the Adaptive Distance Field technique [49] are defined by sampling on an octree grid, rather than a regular grid. However, adaptively-sampled volume datasets are closely related to their uniformly-sampled counterparts in that adaptive sampling simply attempts to avoid storing all the samples that would be necessary at the highest sampling frequency. Hence, Adaptive Distance Fields have many properties similar to those listed above. In particular, creases still cannot be faithfully represented, although if the adaptive sampling is fine enough this may be difficult to discern visually [49]. One difference is continuity - standard reconstruction filters produce $C^0$ discontinuities if directly applied to irregularly-spaced samples (Section 4.9).

## 3.5 Point-Set Interpolation Schemes

A variety of implicit surface representation schemes have been developed which are rooted in the conceptual model of defining a surface based on a point cloud. Given a set of samples which are assumed to lie on the surface, the problem is to generate an implicit function such that some iso-contour passes through the sample points (usually the zero iso-contour). These techniques will be referred to as *point-set interpolation schemes*.

In some sense, these methods are similar to volume datasets, however there are two key differences. First, volume datasets are constructed from regularly spaced samples[2], while point-set schemes are designed to deal with unstructured point clouds. Second, volume datasets directly specify the field value $f_i$ at each sample. Point-set interpolation schemes generally take only sample positions as input, and assign field values automatically. As a result, there is much more variability in the individual methods, and they must be examined independently. Several recent methods are explored in the following sections. Two generalizations which can be made are that none of the techniques can represent analytic surfaces, and that normalization is entirely dependent on the given set of samples.

All of the following techniques generate iso-surfaces at $f = 0$, and hence can be functionally composed within the F-Rep framework (Section 3.3.1). However, modeling systems based on these techniques generally combine the sample sets instead. This approach has been used with some success in several interactive implicit modeling systems [65, 13].

### 3.5.1 Variational Implicit Surfaces

Variational implicit surfaces [92, 113, 34, 114] interpolate or approximate a set of point samples using a weighted sum of globally-supported basis functions. The resulting scalar field $f$ is sometimes referred to as a *Radial Basis Function* or *RBF* [34] [3]. Variational implicit surfaces are $C^k$, with $k$ dependent on the choice of basis function, although the $C^2$ thin-plate spline is most commonly used [114, 34]. This basis function is unbounded, hence so is the variational

[2] Octrees do define a regularly-spaced sampling, they simply avoid storing redundant samples.

[3] The term RBF refers to the radial symmetry of the basis function applied at each sample point, and hence referring to the entire $f$ as an RBF is a misnomer. In addition, the basis function need not be radially symmetric [43].

implicit surface. Since the field is globably $C^2$, creases cannot be defined. Anisotropic basis functions [43] can be used to produce fields which change more rapidly and may appear to have creases, however the surface is still smooth when inspected at the appropriate scale. The smooth field implies that self-intersections do not occur, and hence volumes are always well-defined. The thin-plate spline guarantees that global curvature is minimized [44], which intuitively seems to imply that internal zero iso-contours will not occur. However, no formal proof could be located[4].

Variational inteprolation has many properties which are desirable for 3D modeling, however controlling the resulting surfaces is very difficult. In Chapter 5, variational interpolation will be used to generate free-form skeletal primitives.

### 3.5.2 Compactly-Supported Variational Implicit Surfaces

Variational implicit surfaces based on compactly-supported radial basis functions (CS-RBFs) were proposed as a technique to reduce the computational cost of variational interpolation techniques [72]. Each CS-RBF only influences a local region, so computing $f(\mathbf{p})$ only requires evaluation of basis functions with some small neighbourhood of $\mathbf{p}$. As with the globally-supported counterpart, the resulting field is $C^k$, creases are not supported, and self-intersections cannot occur. The local support of each basis function results in a bounded global field, however this also guarantees that additional iso-contours will be present, as noted by various authors [75, 87].

### 3.5.3 Multi-level Partition of Unity Implicits

Multi-Level Partition of Unity (MPU) Implicit Surfaces [75] are defined by a set of point constraints with normals. Similar to compactly-supported variational implicit surfaces, the basic theory behind MPU implicits is to subdivide space into regions and construct a local implicit approximation of the surface in each region. To evaluate $f(\mathbf{p})$, overlapping basis functions are blended using partition-of-unity weights generated using $C^k$ functions. However, to optimize construction an adaptive octree-based spatial subdivision technique is used. Several basis functions have been applied, [75] uses $C^1$ quadrics while [87] uses local $C^k$ radial basis functions These basis functions produce unbounded fields, hence the global field is unbounded.

Certain types of creases can be represented on a per-cell basis by fitting multiple quadric surfaces and combing them using functional CSG operators. However, cells with creases must be detected based on the deviation of the point set normals. This procedure is far from robust, limiting [75] to edges (two surfaces) and corners (three surfaces)). However, [87] notes that the local creases are not preserved by the partition-of-unity blending functions. The quadric representation guarantees that volumes are defined, and the octree-based decomposition reduces the possibility of unwanted internal iso-surfaces, but there is no guarantee that they are voided completely.

[4]The author admits only limited familiarity with the related mathematical literature, so such a proof may indeed exist.

### 3.5.4 Implicit Moving Least-Squares

The Implicit moving least-squares (IMLS) technique [105] is closely related to MPU methods. The major improvement is the addition of value constraints over entire triangles, allowing entire triangle meshes to be exactly interpolated. Interpolation can also be relaxed, creating smoother approximations to the original mesh. Normal constraints are also improved, to reduce ringing artifacts. As with MPU implicits, various $C^k$ basis and weighting functions can be used. Since adjacent triangles can be interpolated exactly, IMLS can represent creases, but only if exact interpolation is used - approximating implicit surfaces are always smooth. If the initial mesh is self-intersecting, the IMLS surface will reproduce these self-intersections, and possibly introduce others. Since the basis functions have global support, unwanted internal iso-contours are unlikely to occur, however it is unclear if this is guaranteed.

## 3.6 Classification Summary

In the previous sections, a variety of implicit surface modeling techniques have been analyzed with respect to the properties described in Section 3.2. The results of this analysis are summarized in Figure 3.2. The columns in the table correspond to the properties listed in Section 3.2, and the rows to the various implicit modeling schemes. In an attempt to be as conservative as possible, any properties about which there is some debate are marked as having "limited support". In these cases, the analyses in the previous sections should be consulted.

The *Continuity* column lists the maximum continuity that can be achieved, including blending and CSG operations for the constructive frameworks. The *Bounded*, *Analytic Surfaces*, *Creases*, and *Guaranteed Volumes* columns are essentially either-or properties, although some techniques have limited support. The *Expected Volumes* column indicates what little is known about this property. Likewise, the *Normalization Error* column provides only a relative ranking among the constructive techniques - very little can be said about discrete volumes and point set interpolation.

## 3.7 Selecting a Modeling Framework

Ideally, an interactive implicit modeling system would support any implicit surface modeling technique. However, certain incompatibilities, such as bounded *vs* unbounded fields, prevent the composition of different types of fields. To construct an internally consistent modeling system, a specific modeling technique must be selected. The purpose of the classification developed in this chapter was to assist with such a choice.

Some desirable properties are already known. Guaranteed validity, ie the impossibility of creating self-intersecting surfaces which do not define a volume, is clearly beneficial. The ability to represent analytic surfaces is useful in engineering contexts. The local influence provided by bounded fields leads to more intuitive results during interaction with the model. Already, the possible choices have been narrowed down to two modeling frameworks - BlobTree modeling and Convolution Surfaces.

| | Continuity | Bounded | Analytic Surfaces | Creases | Guaranteed Volumes | Expected Volumes | Normalization Error |
|---|---|---|---|---|---|---|---|
| F-Reps / Signed Fields | $C^k$ | | ● | ● | | ○ | High |
| F-Reps / Unsigned Fields | $C^k$ | | ● | ● | | | High |
| BlobTree / Bounded Fields | $C^1$ | ● | ● | ● | ● | ○ | Moderate |
| Convolution Surfaces | $C^k$ | ● | ○ | | ● | ● | Constant |
| Volume Datasets | $C^0$–$C^2$ | ○ | | | | | Variable |
| Adaptive Distance Fields | N/A | ○ | | | | | Variable |
| Variational Implicits (VIS) | $C^k$ | | | | ● | ◐ | Variable |
| Locally-Supported VIS | $C^k$ | ● | | | ● | | Variable |
| MPU / MPU-RBF | $C^k$ | | | ○ | ● | | Variable |
| IMLS | $C^k$ | | | ● | | ◐ | Variable |

● = Supported ○ = Limited support ◐ = Suspected support

Figure 3.2: *Implicit Surface Classification Quick Reference Chart*

Convolution surfaces do have some compelling benefits. In particular, they have the most controllable normalization error of all the techniques described above. Unfortunately the skeletal-composition approach used in convolution surface modeling can only represent smooth surfaces. While this is clearly a promising direction for future work, one must currently step outside of the convolution framework into a more general modeling system, such as the BlobTree, to perform basic operations such as boolean CSG.

Hence, the BlobTree remains as the only implicit modeling framework which supports the desired properties. The main drawback of the BlobTree is that only $C^1$ CSG operators are currently known. However, this is an area of active research, and smoother methods may be developed in the future. Similarly, the problematic non-monotonic field generated by CSG difference operations is likely avoidable. Normalization error does seem to be an inherent problem (Section 2.10), however unlike in the R-operator case, the problem is localized by the bounds of the relevant fields. Additional control over normalization error may be possible using bounded blending operations [51].

## 3.8 Chapter Summary

Classifying mathematical functions is a difficult task, perhaps best left to mathematicians. Unfortunately, mathematicians seem to have largely neglected issues relating specifically to interactive computer graphics systems. Hence, in this section a set of functional properties have been isolated and used to classify a set of recent implicit surface modeling techniques. Both hierarchical modeling frameworks and schemes based on discrete samples have been classified. The result is a taxonomy of schemes which has been used to select a specific system, the BlobTree hierarchical modeling framework, for use in an interactive system.

A final note must be made to emphasize that this classification is only a preliminary attempt. Like the animation-specific comparative study presented in [33], it is biased towards its intended application - 3D modeling. The taxonomy presented here is not intended to be the final word on the subject, but rather a tool for recognizing the current limitations of implicit volume modeling.

# Chapter 4

# Hierarchical Spatial Caching

*Issues relating to interactive visualization of BlobTrees are introduced. An analysis of existing caching schemes leads to a novel new approach called Hierarchical Spatial Caching. Conceptual and practical implementation issues are discussed. Profiling results show an order-of-magnitude improvement in visualization times. Adaptive Distance Fields are analyzed for use in this caching scheme, and found to be unsuitable.*

*Some material in this chapter is taken from the publication "Interactive Implicit Modeling with Hierarchical Spatial Caching" by Schmidt, Wyvill, and Galin [97]. The material appearing here is due to Schmidt. The Medusa model was provided by Galin.*

## 4.1 The Interactive Visualization Problem

One of the critical constraints on interative tools for hierarchical constructive implicit modeling is the cost of visualizing the model. Although volume visualization techniques are becoming more common, traditional 3D modeling interfaces are based on the notion of manipulating surfaces. Unfortunately, the surface $\mathcal{S}$ defined by $f(\mathbf{p}) = v$ is in some sense unknown. Unlike a parametric surface patch, there is no function to evaluate which outputs points on the surface. Without an explicit surface function, visualization algorithms must resort to *spatial searches* to find the surface. These methods generally produce a discrete sampling of the surface, this sampling can then be rendered interactively.

A variety of spatial search algorithms for sampling the implicit surface have been developed. Most approaches produce triangle meshes [124, 109, 53, 5, 64, 93, 57] or point clouds [117, 56, 70, 110]. Regardless of the sampling technique, the fundamental operation in all of these algorithms is finding points on (or near) the surface by evaluating $f(\mathbf{p})$. Most algorithms rely on some geometric approach to find points "near" the surface, and then refine these guesses, converging on the surface using repeated evaluations of $f(\mathbf{p})$.

When visualizing complex implicit surfaces, these convergence steps (Section 2.9) consume the majority of the computational effort. As an example, a profiling test was performed on a relatively simple model (100 blended randomly-positioned point primitives) using a standard polygonization algorithm [26]. The $f(\mathbf{p})$ function was programmed using hand-optimized and tuned SIMD assembly code, to maximize performance. In this case, at moderate triangle resolutions the evaluation of $f(\mathbf{p})$ consumed 95% of the CPU time. At production-quality triangle resolutions the cost of $f(\mathbf{p})$ increased to 99%. Similar results have been observed with other visualization algorithms.

Clearly, the cost of visualizing an implicit surface with spatial search algorithms is directly dependent on the cost of evaluating $f(\mathbf{p})$. Recent visualization algorithms [70] often report times of multiple seconds or minutes to sample complex implicit surfaces. Unfortunately, a key

requirement of interactive modeling is that the surface visualization respond in real-time to the designer's actions. To support surface resampling at interactive rates, the cost of evalauting $f(\mathbf{p})$ must be reduced.

## 4.2 Problem Analysis and Solution Overview

If $f(\mathbf{p})$ is assumed to be an arbitrary "black box" function, there is essentially nothing that can be done to reduce it's evaluation cost. Hence, some narrowing of scope is necessary, and for the rest of this chapter $f(\mathbf{p})$ will be assumed to be a scalar field defined by a BlobTree implicit model.

BlobTree models (Section 3.3.2) are procedural models, defined hierarchically by a tree of nodes. At the leaves of the tree are *primitives*, the building-block implicit volumes. These are generally simple implicit volumes [120], although any arbitrarily complex implicit volume can be used as a primitive. The internal tree nodes are *operators*, which combine their child nodes into more complex shapes. Since primitives are simply scalar fields, and operators simply combine scalar fields to produce a new scalar field, the result of an operator can be considered to be a new primitive. This equivalence allows complex models to be incrementally constructed. A simple BlobTree example is shown in Figure 4.1. For the purpose of this chapter, the functions defining primitives and operators will be considered "black boxes" - no knowledge of their internal structure will be assumed.

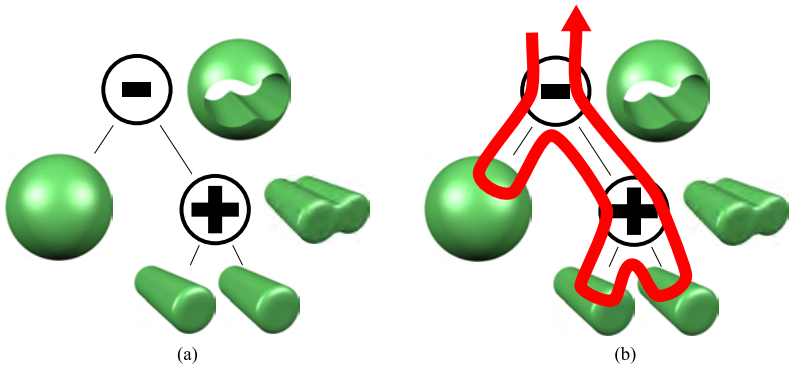



Figure 4.1: *The final BlobTree model in (a) is created by joining two cylinder primitives with a blend operator, and subtracting them from a sphere primitive with a CSG difference operator. In (b), the recursive path of a BlobTree evaluation is shown by the red arrow. All the nodes in the tree are evaluated.*

Consider the evaluation of a BlobTree $f(\mathbf{p})$ at a point $\mathbf{p}$. The field value is computed by recursively descending the BlobTree (Figure 4.1b). At interior nodes (composition operators), the descent may be pruned based on the bounding box of the operator. Eventually, leaf nodes are evaluated, and then their values are incrementally combined by operators as the recursion unwinds. Finally, a scalar value is produced at the root of the tree.

Although algorithmically simple, visualizing BlobTree models using this naive approach is incredibly expensive. Adding a node to a BlobTree increases the cost of evaluating $f(\mathbf{p})$ by some

factor $m$, where $m$ is the cost of evaluating the new node. Assuming that the surface is sampled with $k$ points, and $n$ convergence steps are taken to refine each sample position, then adding adding a new primitive increases the cost of sampling the surface by $m \cdot n \cdot k$ operations. If these factors are assumed to be constant, then the visualization time doubles as the number of nodes in the tree doubles.

While linear growth is generally considered quite good from a theoretical standpoint, in practice the resulting explosion in computational cost prohibits usable interactive BlobTree modeling systems. Simple models can be manipulated in real-time, however as the timings in Section 4.8.2 clearly show, existing methods are several orders of magnitude too slow to support high-quality interactive visualization of even moderately complex models.

Since primitives and operators are assumed to be "black boxes", there are few opportunities for optimization in the basic algorithm. Discarding the "black-box" formalism is undesirable. The power of BlobTree modeling is largely derived from it's generality, of which the "black-box" notion is a critical component. Hence, the only avenue for reducing evaluation cost is to evaluate fewer tree nodes.

In the following sections, a method will be developed for replacing sub-trees of a BlobTree model with dynamically-computed scalar field approximations. These approximate scalar fields can be evaluated in constant time, reducing the cost of evaluating the approximated subtree from $O(N)$ to $O(1)$. To avoid unnecessary pre-computation overhead, the approximate scalar fields are generated using dynamic lazy evaluation. This *Hierarchical Spatial Caching* technique is both a caching and an approximation scheme. In a variety of tests, use of hierarchical spatial caching has been found to produce an order of magnitude reduction in visualization time over existing techniques.

## 4.3 Previous Caching and Approximation Schemes

A common approach to reduce the cost of visualizing an implicit surface is to employ caching schemes. Caching techniques generally try to exploit coherence in the global state of the BlobTree. Caching can happen at many levels, from very coarse caching at the level of inter-frame temporal coherence, to very fine caching of node-specific variables during during a BlobTree traversal.

*Local update* schemes [62, 117, 42, 53, 5, 14] cache surface samples, exploiting temporal coherence to reduce the number of evaluations of $f(\mathbf{p})$. When the BlobTree is modified, local support ensures that the changes in $f(\mathbf{p})$ are bounded within some region $\mathcal{B}$. The surface outside $\mathcal{B}$ is guaranteed to be unmodified, and hence the samples outside $\mathcal{B}$ need not be recomputed. Local update schemes do improve interactivity, and can be used with most other caching schemes. However, they do not fundamentally reduce the cost of evaluating $f(\mathbf{p})$, so large or complex update regions are still problematic.

BlobTree traversals can also be cached, to some extent. One approach is to generate a *cache tree* [48, 51]. A cache tree mirrors the BlobTree, containing a cache node for each tree node. Temporary variables and field values computed during the last tree traversal are stored in each cache node. The cached values are only valid if the tree is traversed multiple times at a point $\mathbf{p}$.

Cache trees provide a speed-up mainly in situations where the field value and gradient are both computed at $\mathbf{p}$. If $\mathbf{p}$ changes between each evaluation, cache trees actually make evaluation of $f(\mathbf{p})$ more expensive. A similar approach involves pre-computing values that will be constant along a 3D ray [15].

Another take on caching traversals is to subdivide space into voxel "bins" and *prune* the BlobTree at each bin [48]. The BlobTree is traversed for each voxel, and a new tree is constructed which only contains nodes that intersect the current bin. These pruned trees are used instead of the original tree to evaluate $f(\mathbf{p})$. This scheme is based on the observation that for a complex BlobTree containing nodes with many children, a large number of redundant bounding box tests may be performed. The pruning step is too expensive to perform interactively, and must be repeated whenever the tree is modified. Similar speed-ups could likely be achieved with less pre-computation by dynamically generating bounding box hierarchies at nodes with many children. However, like cache trees, while performance is definitely improved, the gains are not sufficient to provide interactivity.

The caching schemes mentioned so far are essentially optimization techniques. Visualization time is reduced by avoiding redundant computation, but the cost of visualizing a BlobTree is not fundamentally reduced. To achieve this, approximation schemes have been developed which represent the implicit surface using a BlobTree which has fewer nodes, or nodes that are less expensive to evaluate. For example, [29] suggests replacing expensive spline primitives with a set of blended points primitives. This concept has been generalized as *level-of-detail* or LOD implicits [15]. Curve and surface primitives based on subdivision skeletons [12, 58] provide a simple framework for LOD primitives. However, LOD primitives are problematic for interactive modeling because lower levels of detail do not accurately reflect the final model. Even if the surface of the LOD primitive is relatively accurate at lower detail levels, the underlying field may be different, producing a different surface after blending. LOD primitives also only reduce computational cost at the leaves of the tree, the techniques do not generalize to approximating the result of composition nodes.

Caching and approximation is combined in *spatial caching* schemes. A basic spatial cache can be created by sampling $f(\mathbf{p})$ on a uniform grid. The field value at any $\mathbf{p}$ can then be approximated by applying a reconstruction filter, such as tri-linear interpolation, to a set of samples near $\mathbf{p}$. This type of spatial cache can support level-of-detail by reducing or increasing the number of samples stored in the cache. General spatial caches simply approximate the scalar field $f(\mathbf{p})$, instead of attempting to explicitly approximate the implicit surface, and hence can be applied to both primitives and composition nodes.

Barthe [18] describes a general constructive modeling system based on quadratic interpolation of uniformly-sampled grids (which are essentially spatial caches). The system allows a designer to interactively combine two shapes, creating a new sampled grid. The new grid can again be combined with another, resulting in a binary tree of composition operations. To conserve memory, the tree of grids is not stored, hence the user can only manipulate the root node (or append a new node). Frisken [49] describes a similar system but replaces the uniform grids with adaptively-sampled distance fields (ADFs) to improve the representation of small details. Again, tree editing is only supported at the root node - the the notion of procedural modeling is essentially discarded. Without the hiearchical framework, benefits of the BlobTree such as

animation and non-linear editing are lost.

Akleman [7] describes a related type of spatial caching for ray-tracing certain classes of implicit surfaces known as *ray-quadric* surfaces. These primitives, when evaluated along a ray, reduce to quadratic functions which can be easily solved. The function coefficients can be cached, permitting quick ray-surface intersection tests. Ray quadrics can also be functionally composed, using a somewhat-cumbersome interface based on manipulation of 3D prisms [7]. However, individual ray-quadrics are limited to "star-shapes" and the caching techniques do not generalize for use in general constructive modeling systems.

The use of spatial caches in these works demonstrates the computational benefits of approximating analytic scalar fields with discrete sampling. However, none of these systems provided fully interactive BlobTree-style modeling. Akelman's [7] caching system does not generalize to black-box hierarchical modeling. The memory and processing limitations inherent in Barthe's [18] approach prevent non-linear editing of the model tree, and hence many of the benefits of hiearchical modeling are lost. Kizamu [49] suffers from a similar limitation. In some sense, because the underlying analytic functions are discarded, it is debatable that the use of spatial approximation in [18] and [49] should even be called caching. Regardless, while the desired level of interactivity was not achieved, the results are promising and suggest that further investigation is warranted.

## 4.4 Hierarchical Spatial Caching

To address the shortcomings of previous applications of spatial approximation to hierarchical implicit modeling, the *Hierarchical Spatial Caching* method was developed. The underlying idea in Hierarchical Spatial Caching is is to dynamically insert spatial caches into the BlobTree. These spatial caches are re-cast as *cache nodes* which can be placed above any other internal node in the BlobTree. The samples of $f$ stored in the caches are dynamically computed as they are needed. The hierarchical caches are also dynamically invalidated (and recomputed) as the designer modifies the BlobTree. The main components of Hierarchical Spatial Caching are described in the following paragraphs.

In previous works, spatial caches have been used at the primitives and at the root of the hierarchical model tree. Primitive caches do not reduce the cost of evaluating complex trees. In the BlobTree, where primitives are relatively inexpensive to evaluate [120], primitive caches provide little benefit. Global caches do reduce the cost of evaluating $f$. However, consider that during interactive manipulation, some region of $f$ is being modified and hence the cache in that region must be invalidated and recomputed. Similar to *local update* schemes, if the modified region is large or expensive to evaluate, the global cache fails to preserve interactivity. In particular, global caching provides marginal benefit if a local update scheme is used for surface visualization.

In contrast to primitive and global caches, the approach taken in hierarchical spatial caching is to insert spatial caches *inside* the BlobTree. As noted above, there is marginal benefit to caching individual primitives or the root node, so caches are only placed above internal composition operator nodes. The spatial cache is then formalized in the BlobTree framework as a *cache node*. A caching node $\mathcal{C}$ is a unary operator with a single subtree $\mathcal{T}$. Each caching node stores a set

of exact potential field values determined by evaluating its subtree $\mathcal{T}$ . When evaluating the potential field of $\mathcal{T}$ at a point $\mathbf{p}$ inside the bounding box of $\mathcal{T}$, an approximation $f_{\mathcal{C}}(\mathbf{p})$ to the exact **potential** field value $f_{\mathcal{T}}(\mathbf{p})$ is reconstructed from the cached potential values. The subtree $\mathcal{T}$ is not traversed, reducing the cost of evaluating it from $O(N)$ to $O(1)$ (Figure 4.2).

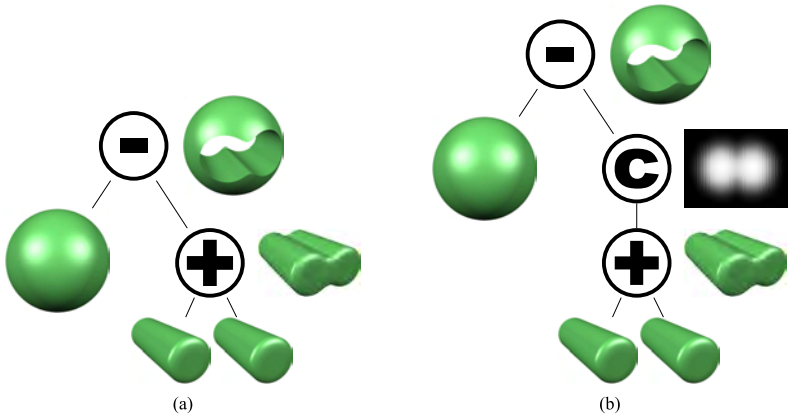



Figure 4.2: *In (a), two cylinder primitives are blended, and then subtracted from a sphere. In (b), a cache node is inserted above the blend node, reducing the cost of evaluating the blend sub-tree.*

Previous applications of spatial approximation to constructive implicit modeling have entirely replaced the analytic functions with discretely-sampled grids [18, 49]. In this situation, all grid samples must be stored and updated at all times. However, in the case of cache nodes, the underlying analytic BlobTree functions are not discarded. Hence, cache samples can be computed on-demand, and there is no reason to store fully evaluated grids. When a cache node is to be evaluated at a point $\mathbf{p}$, the cache data structure is inspected to determine if the samples required to reconstruct $f_{\mathcal{T}}(\mathbf{p})$ are available. If some are not, they are first computed, and then $f_{\mathcal{C}}(\mathbf{p})$ is evaluated. In short, cache nodes are filled using *lazy evaluation* (Figure 4.3).

Lazy evaluation is a significant benefit, as full evaluation of high-resolution grids is computationally intensive. In addition, if surface-tracking visualization algorithms are used, only cache samples near the surface are necessary. In this case most of the samples in a fully evaluated grid will never be used - particularly if they will be invalidated in the next frame as the user drags a primitive across the screen.

Since Hierarchical Spatial Caching involves replacing analytic scalar functions with discrete samplings, it is an approximation scheme. However, because cache nodes are dynamically initialized based on evaluations of $f(\mathbf{p})$, and subtree traversals are dynamically truncated, cache nodes can also be considered traversal caches. The novelty of Hierarchical Spatial Caching is the combination of spatial approximation with traversal caching. Individually, these techniques have limited ability to reduce the cost of interactive visualization for a complex BlobTree. When combined they can produce an order-of-magnitude reduction in visualization time.

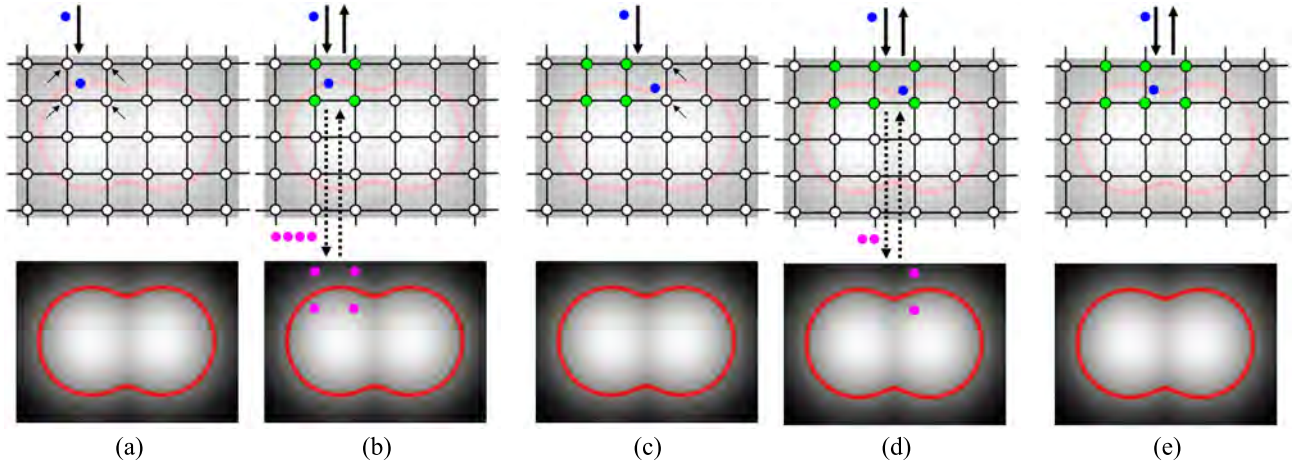



Figure 4.3: *Lazy evaluation process. In (a), the field values necessary to reconstruct the value at the incoming query point (in blue) are unavailable. The child field must be evaluated 4 times, once for each cache value (b). In (c), two cache values are missing and must be evaluated in the child field (d). Finally, in (e) all cache values are available. The incoming value query can be directly approximated in $O(1)$ time, no $O(N)$ evaluations of the child field are necessary.*

## 4.5 Hierarchical Spatial Caching Implementation Issues

A number of design decisions must be made when implementing hierarchical spatial caching. The choice of sampling scheme is paramount to the performance and accuracy of any spatial approximation scheme, however several other implementation details hold regardless of the particular sampling scheme.

The first issue relates to dynamic cache invalidation. During tree traversals in a standard BlobTree implementation, child nodes are evaluated by their parents. Child nodes never initiate events - there is no upward communication. However, to implement dynamic cache invalidation, there must be some mechanism to tell a parent node when it's child has been modified. Each node on the path from the modified child to the root node must be notified, so that any cache nodes on the path have a chance to discard now-invalid cache values.

There are essentially two option for implementing this parent notification. One option is to use an external notification scheme - an "oracle", which knows about all the nodes in the model tree and can explicitly notify them. This scheme preserves the top-down organization of the BlobTree - children do not need to know their parents. Also, the oracle can accumulate invalidation events and then only notify each node once, which can be more efficient if multiple children change simultaneously. The second option is to use a parent-notification scheme, where invalidation events are passed up the model tree from child to parent. This scheme preserves the desirable BlobTree properties of encapsulation and local knowledge, at the cost of possible repeated notifications. There is no clear winner between these options. The oracle introduces significant implementation complexity, however the upward-notification approach may be less efficient.

The actual process of invalidating a cache node is somewhat dependent on the particular

spatial caching scheme. However, in general invalidation occurs based on some bounding region. When initiating an invalidation event, the modified child should compute a bounding volume for the updated region. Note that when a change to the child modifies the bounding volume, both the old and new volumes must be invalidated. While two events can be sent, it can be more efficient to join the bounding volumes and send a single event. Of course, the smaller the bounding volume the better, however this must be weighed against the cost of computing a more accurate bounding volume. The simplest approach is to take the union of the "before" and "after" bounding boxes of the modified node, and invalidate this entire region. However, some relatively simple optimizations can be made, particularly at composition nodes. For example, when changing the blending parameter on a Ricci blend (Section 2.3), only the intersection of the child bounding boxes must be invalidated.

Another issue concerns the amount of coupling between the cache node and it's subtree. In the previous section, the cache node $\mathcal{C}$ was described as simply another type of BlobTree node. While this is a useful abstraction that fits cleanly into the BlobTree conceptual framework, it does not necessarily lead to the most efficient implementation. Consider the spatial cache of an additive blend operator (Section 2.4) applied to two fields, one a simple primitive $f_A$ and another a complex sub-tree $f_B$. In this case, each value in the cache is the sum of the two child field values, $f_A + f_B$. If the simple primitive is modified, two cache update strategies are possible. The one described so far would entail clearing all the cached values in the bounding box of $f_A$, and then re-computing $f_A + f_B$ as necessary. The second option is to add the value $-f_A$ to each cached value, and then only re-compute $f_A$ as needed and add it to the cached values. The re-evaluation of expensive-to-compute values of $f_B$ is avoided.

This "removal" of sub-tree values from the cache, or *partial-invalidation*, can be applied to any operator for which the contribution of a particular child can be computed based soley on the field value of that child. These operators in general will be called *separable*. Some analysis of separable operators is provided by Akleman [8][1]

Combining lazy-evaluation with partial-invalidation introduces another complication. Some book-keeping is necessary to keep track of which cache values contain $f_A + f_B$, and which only contain $f_B$. In the case of uniform grids, a separate grid can be maintained, where each grid cell contains a set of bit flags, one for each child of the composition operation. This additional overhead, combined with the necessity of re-computing values of $f_A$ to subtract them from existing cached values, makes partial-invalidation impractical unless the overhead is lower than the cost of re-computing $f_B$.

To support partial-invalidation, cache nodes now must be aware of whether or not they are caching a separable operator, whether or not partial-invalidation is warranted, and if so, the book-keeping bit-masks for each child of the operator. Cache nodes that support partial-invalidation are no longer traditional black-box BlobTree nodes. In fact, since the child of a cache node will always be an operator (Section 4.4), it can be more efficient to discard with the notion of a separate "cache node" entirely, and simply attach a spatial cache to each operator. This type of direct spatial caching at operators can be considered *tightly-coupled* spatial caching, as opposed to the *loosely-coupled* abstract cache node of Section 4.4 (Figure 4.4).

[1] In this work, separable operators are referred to as "constant time updateable" operators.

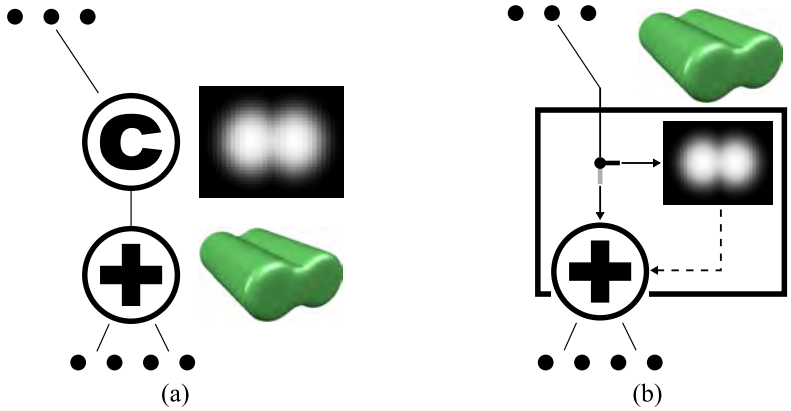



Figure 4.4: *Loosely-coupled cache node (a) and tightly-coupled caching blend node (b).*

Finally, a critical issue is positioning of caches in the model tree. In theory, a cache node could be placed above each composition operator. However, this introduces two complications. First, cache nodes necessarily introduce approximation error, which would accumulate in each successive cache. The caches at the top of a deep model tree would have excessive error. Second, when a leaf node in the tree is modified, invalidation and lazy evaluation of all the intervening caches becomes incredibly expensive, negating the benefits of hierarchical caching. Clearly, caches should be used sparingly, where they will provide the greatest benefit. However, determining these locations is difficult. The simplest option is to leave cache placement to the model designer, however this is clearly undesirable.

An alternative is to take an algorithmic approach to cache management, determining placement of cache nodes automatically. Static optimization techniques based on properties such as tree depth and node complexity may not produce results which are optimal for interactive manipulation. Instead, cache management algorithms should take into account the current actions of the user, minimizing the number of caches along the path to the node being manipulated. This may necessitate dynamic tree rewriting, where an equivalent BlobTree is generated in which the node being modified is segmented from any operators which are cached. Extensive investigation of such techniques has not been carried out, however an initial attempt is described in Section 4.7.5.

## 4.6 Sampling and Reconstruction

Two choices must be made to implement a spatial caching node which approximates a scalar field $f$. First, a sampling scheme must be chosen. The alternatives are uniform sampling, as used by [18, 7], or some form of adaptive sampling, such as octree-ADFs [49]. For most sampling schemes, multiple *reconstruction filters* are available which produce an approximation to $f$ from the given samples.

Ideally, adaptive sampling schemes provide more accurate approximation and use less memory. Octree-ADFs have emerged as a leading approach for approximating distance fields [49]. However, several un-addressed issues with the ADF method, and adaptive sampling schemes in

general, make them unsuitable for use in Hierarchical Spatial Caching. These issues are discussed in more detail in Section 4.9. Due to these shortcomings, uniform sampling seems to be the most feasible approach to implementing cache nodes.

A variety of reconstruction filters compatible with uniform sampling have been developed, and are thoroughly analyzed in [69]. The most popular filter is the *tri-linear* filter, which is the three-dimensional extension of basic linear interpolation. The tri-linear filter is simply a linear interpolation between the 8 nearest sample values. The technique is described in detail in Appendix A. The key limitation of this filter is that it is only $C^0$ continuous - gradient discontinuities occur along the edges between each sample.

As discussed in Section 2.7, $C^1$ continuity is desirable to avoid shading artifacts. A cubic $C^2$ filter is analyzed in [69], however this filter is very costly because reconstruction at a point $\mathbf{p}$ requires 64 neighbouring samples and 21 cubic polynomial evaluations. To reduce this cost, [71] introduced a $C^1$ *tri-quadratic* filter which requires only 27 neighbours and 13 quadratic polynomial evaluations. These quadratic polynomials do not pass through the original sample values, resulting in the reconstructed scalar field being smoothed out. In particular, small features can be lost. Hence, unlike the interpolating tri-linear filter, the tri-quadratic filter is an *approximating* filter.

As is clear from Figure 4.5, the $C^1$ tri-quadratic filter provides superior visual results when compared to the $C^0$ tri-linear filter. However, because the tri-quadratic filter requires over 3 times as many neighbour samples and significantly more computation, it is more expensive to evaluate. In practice, it was found that tri-quadratic reconstruction was approximately twice as expensive as tri-linear reconstruction, when integrated into hierarchical spatial caching.

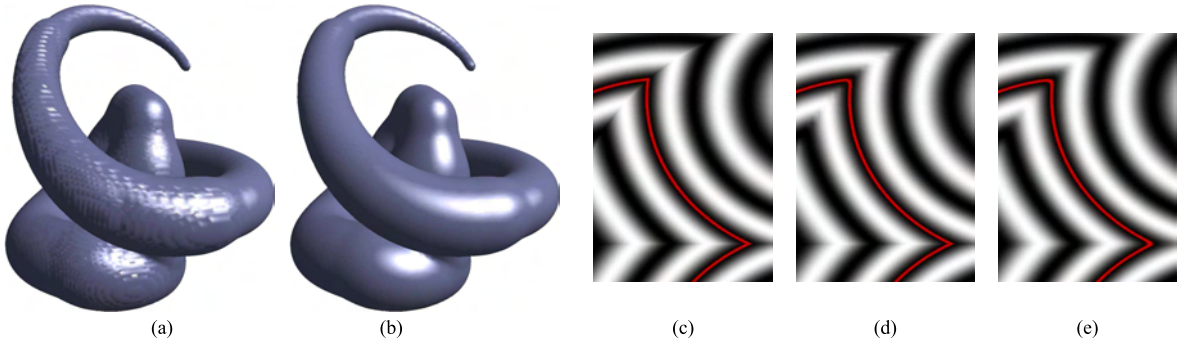



Figure 4.5: *Tri-Linear (a) and Tri-Quadratic (b) gradients. Close-up of (c) un-cached sharp features, (d) tri-linear interpolation, and (e) tri-quadratic approximation.*

While this extra cost is undesirable, two observations can be made. First, for many visualization algorithms, significantly more evaluations of $f$ are performed than of $\nabla f$. Second, the $C^1$ continuity necessary to produce smooth shading is only is only required in $\nabla f$. Hence, a third reconstruction option is to use the cheaper tri-linear filter to reconstruct $f$, and the $C^1$ tri-quadratic filter to reconstruct $\nabla f$. In practice, this approach was only 10% more expensive than a pure tri-linear approach.

Using different filters for $f$ and $\nabla f$ necessarily introduces a discrepancy between surface vertices and surface normals. However, if the uniform sampling resolution is reasonably high,

and the polygonization resolution is on the order of the grid resolution, then these differences are not visually apparent. A more serious problem may occur for surface-convergence algorithms which assume that $\nabla f$ points towards the surface. This may not be true with mixed filters, so a pure tri-quadratic approach would be necessary.

One final issue with both the tri-linear and tri-quadratic filters is that it is not possible to reconstruct sharp edges. As shown in Figure 4.5, the tri-linear filter smooths out creases except when the crease lies on the edge between two samples. The tri-quadratic filter smooths out all creases. In fact, all the uniform-sampling reconstruction filters analyzed by [69] fail to reproduce sharp edges. Similar problems are encountered with adaptive sampling. A claim has been made that ADFs can reconstruct sharp edges [49], however this is technically not accurate. Due to very high sampling resolution, ADF surfaces appear to have creases when viewed from a distance. However, when viewed from closer distances the same problems as shown in Figure 4.5d will appear, since ADFs use tri-linear reconstruction.

## 4.7 A Sample Spatial Caching Implementation

To analyze the performance benefits of hierarchical spatial caching, a sample C++ implementation was created. As discussed in Section 4.6, uniform grids appear to currently be the most effective approach to dynamic spatial caching, and are used in the test system. By default, tri-linear reconstruction is used for field values and tri-quadratic reconstruction for field gradients, although these options can be toggled interactively.

In terms of the design options described in Section 4.5, the test implementation utilizes tightly-coupled caches attached to composition operator nodes. Parent-notification based on axis-aligned bounding boxes is used for cache invalidation, rather than an external oracle. Although far from optimal, the union of "before" and "after" bounding boxes is used for invalidation in all cases.

An automatic cache placement algorithm is applied (described below), however caching at an operator node can be directly controlled by the designer. Automatic cache placement is not controlled directly by the operator nodes, rather a central cache manager object determines whether the cache at an operator should be enabled. This manager can also perform some rudimentary automatic model tree optimizations, such as removing redundant composition operators.

A variety of specific optimizations and enhancements have been implemented to improve the results of hierarchical spatial caching. These improvements are discussed in the following sections.

### 4.7.1 Dynamic Grid Resolution

Reconstruction accuracy is entirely dependent on the uniform grid resolution. To reduce under- and over-sampling, the test implementation defines the initial cache resolution based on the axis-aligned bounding box of the subtree $\mathcal{T}$ and a user-defined grid resolution $r$. Let $s$ denote the longest side of the bounding box of $\mathcal{T}$. The size of each grid cell is then defined as

$$c = s/r$$

If the bounding box of $\mathcal{T}$ changes (due to some interactive operation), a new cell size $c'$ is computed. If the ratio $c'/c$ is greater than 2 or less than 0.5, the current cache is destroyed, and the cache cell size set to $c'$. This prevents cache resolution both from getting too low, which would result in a coarse approximation of the surface, as well as from getting too high and wasting memory. Based on empirical observations, an default value of $r = 128$ provided reasonable cache accuracy without excessive update overhead.

No attempt is made to transfer the existing cache values to the new cache. In the case of under-sampling this would clearly be undesirable. It may be beneficial in over-sampling situations, however these seem to be fairly rare in practice, so this optimization was not implemented. Instead, the lazy evaluation scheme automatically re-populates the cache as required.

### 4.7.2 Cache Coordinate System

Some of the most common operations in interactive modeling are the basic affine transformations - translation, rotation, and scaling. In a naive implementation, these transformations would be applied to the child of a caching node, causing an expensive cache invalidation. A more efficient approach is to apply the transformation to the cache itself. In the test implementation, each cache has a local coordinate system. Affine transformations applied to the operator node are applied to the associated cache coordinate system as well. This avoids any unnecessary invalidation when transforming the operator node.

Note that avoiding affine-transformation invalidation is more complicated when using loosely-coupled cache nodes (Section 4.5). In that case, transformation nodes are required, and transformations applied to the child of a cache should always be pushed above the cache node. Ensuring this behavior again requires a global view of the BlobTree.

### 4.7.3 Blocked Memory Allocation

High-resolution uniform 3D grids can require significant amounts of memory. A fully-evaluated $128^3$ cache of single-precision floating point values requires 8MB of memory. Current hardware limitations prevent storing a significant number of fixed caches at this resolution. In addition, the volume represented by a single cache can expand, making a fixed grid data structure undesirable because it must then be re-allocated.

To reduce memory usage and avoid expensive re-allocation, uniform 3D grids in the test implementation were allocated using a blocked memory scheme. The uniform grid is divided into $k \times k \times k$ blocks of voxels, where $k$ is the *block resolution*. A voxel block is allocated only when one of the voxels it contains is needed to compute a field value. The blocks are quickly identified using a 30-bit hash, with 10 bits per integer grid axis coordinate [118][2]. This approach saves memory because continuation methods for polygonizing implicit surfaces [26] are designed to follow the surface, hence the required voxels will also be near the surface. Blocks further from the surface may remain completely untouched, and hence unallocated. As $k$ decreases, fewer and fewer blocks need be allocated, further reducing memory usage.

[2]This method limits the total grid size to $(1024 \cdot k)^3$ voxels with 32-bit integers.

However, $k$ also affects the cost of sampling and cache invalidation. Sampling is more expensive because the required neighbours of a particular cell may lie in other blocks, which then need to be looked up with a relatively expensive hash-table access (compared to simply adding a constant to the current index). The smaller the block size, the more likely this is to occur. Also, invalidation becomes more expensive because the invalidated cells must be processed block-by-block. As $k$ decreases it becomes increasingly expensive to find and invalidate all the affected blocks. On the test system, good results were found with a grid resolution of $k = 8$. Reducing $k$ to 4 cut average memory usage by approximately 20%, but increased update times by approximately 25%.

Block size also influences low-level hardware efficiency. Current workstation processors contain several levels of hardware memory cache to reduce memory access latency. The traditional static allocation of a large 3D uniform grid causes frequent cache misses, particularly when accessing adjacent voxels along the z axis. This blocked allocation scheme potentially increases processor cache coherency if the block size fits in a cache line, although this effect is difficult to isolate.

Memory requirements can be reduced even further by using an encoding scheme, at the expense of some reconstruction accuracy. The range of potential field values that can occur in our system is small. Memory usage can be reduced $50 - 75\%$ percent by encoding floating point values as one or two-byte integers. This technique is slightly more computationally expensive and decreases accuracy, so it is not used in the test implementation.

### 4.7.4 Last-Access Caching

As noted in the previous section, looking up the required neighbour values to evaluate a reconstruction filter at a particular grid cell can be a significant cost. The blocked memory scheme only increases this cost. To offset this expense, a *last-access* cache is employed. During reconstruction filter evaluation, the cached neighbour values are saved. If the next filter evaluation occurs in the same cell, the saved values can be used directly, avoiding the relatively expensive neighbour cell lookups.

While it may seem unlikely that the last-access cache would provide much benefit, during convergence iterations 2.9 the same cell is often accessed multiple times. The speed-up provided by last-access caching has not been analyzed, however it introduces negligible overhead so there is really no reason not to use it.

### 4.7.5 Dynamic Cache Placement

As noted previously, the efficacy of hierarchical spatial caching is dependent on the positioning of caches in the model tree. The automatic cache-placement algorithm used in the test implementation is relatively simple. Placement decisions are based on two assumptions - that primitives are quick to evaluate, and that most interactive operations take place near the root of the tree.

First, caches are only allocated to operator nodes that have at least two children. Most operators have no effect on a single child, and the child will either be an operator with a cache, or a primitive. Based on the fast-primitive assumption, caches are also not allocated if all the children of an operator are primitives. Second, caches are only allocated to nodes within $n$ levels

from the root node, where $n$ is a small integer ($n = 3$ in the test implementation). Caches that fall below this threshold as the model tree changes are discarded.

These rudimentary placement rules prove to be quite effective in practice. In particular, for the prototype system described here, a quick response is seen for interactive operations performed near the root of the tree. Operations closer to the tree leaves are slower, however in many cases this can be mitigated by re-organizing the model tree to favor breadth over depth. For example, the sketch-based interface 6 simply appends nodes to the top of the tree. If a body is drawn first, and then an arm, the body is a child of the arm and is slow to manipulate. If the tree is manually re-organized, overall interactivity is greatly increased. This leads to the conclusion that interfaces which assist in the creation of well-organized model trees may be more effective at increasing caching efficiency than smarter cache-placement algorihtms.

## 4.8 Results and Analysis

Hierarchical Spatial Caching has been evaluated in the context of the ShapeShop modeling system (Chapter 6). To analyze performance, polygonization times with and without caching nodes have been compared. The polygonizer in use is an an optimized version of the implicit surface polygonizer described in [26] with the optional cubical decomposition enabled. When computing a mesh vertex on a cube edge, 10 bisections were performed to locate the implicit surface.

The software was compiled using the Microsoft Visual Studio 2003 C++ compiller in Release mode with default optimization flags. All timings reported below were performed on a laptop with an Intel 1.6Ghz Mobile Pentium 4 processor and 512MB of RAM.

Detailed profiling has been carried out using the complex hierarchical Medusa model shown in Figure 4.6. The model is composed of 9490 point skeletal elements segmented into 7 major components. Each major component is modeled as a blend of point skeletal elements distributed along a set of splines. All the points along each individual spline are grouped together into a single optimized blend node to avoid excessive tree traversal costs. The point counts of the 7 components are shown in Figure 4.6.

A caching node with a resolution of $128^3$ voxels[3] was placed above each major component in the model tree. Potential field reconstruction accuracy was evaluated by comparing reconstructed and exact field values at triangle mesh vertices. Using this method, the mean tri-linear reconstruction error for the Medusa model with $128^3$ voxel caches is estimated to be approximately 3%, computed by comparing the real and approximate field values at mesh vertices. The error is concentrated in high-frequency regions, particularly the head. While a global measure of reconstruction accuracy has not been devised, it is suspected that the global error will be similar to the near-surface error because the cache spacing is uniform. Also, note that the modeling technique of blending many points along spline curves used to create the medusa model produces very high normalization error (Section 2.9), resulting in higher approximation error.

While the results discussed in this section focus on the Medusa model, the reductions in

[3] $128^3$ was chosen because powers-of-two produce the most efficient memory alignment, and it was the lowest value which produced a smooth approximation of all the Medusa components, see Figure 4.9.

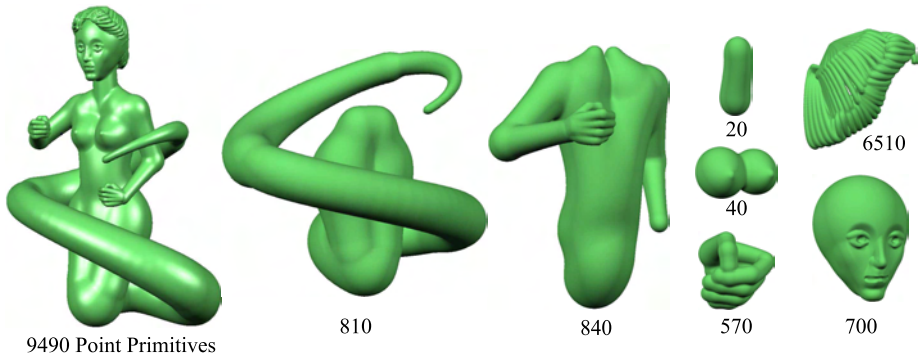



Figure 4.6: *Medusa model composed of 9490 point primitives (left) grouped into 7 major components - lower body, upper body, neck, chest, left hand, hair, and head. Note that the right hand is part of the upper body component, while the left hand is separated.*

computation time have been found to apply to found to apply to hierarchical implicit modeling in general. Comparable measurements have been observed with a wide variety of other simple and complex models. The Medusa model is focused on here because it is the most computationally expensive model that was available and has a wide range of feature scales.

### 4.8.1 Static Polygonization Time

Static polygonization times for the Medusa model with and without caching nodes are compared in Table 4.1. All caches were cleared before each polygonization, and all reported times are an average of 5 tests. In all cases the cached polygonization contained less than 1% more triangles that the non-cached polygonization.

| Cubes | Cache | No Cache | Ratio |
|---|---|---|---|
| $32^3$ | 5.77 | 4.90 | 0.8× |
| $64^3$ | 10.34 | 14.36 | 1.4× |
| $128^3$ | 17.40 | 51.97 | 3× |
| $256^3$ | 29.23 | 199.37 | 6.5× |
| $512^3$ | 49.83 | 809.66 | 16× |

Table 4.1: *Comparison of cached and non-cached polygonization times (in seconds) for Medusa model at different polygonization resolutions.*

Polygonization time without caching increases by approximately a factor of 4 when resolution doubles. Polygonization time with caching is initially slower than without because 8 voxels are required to compute a single tri-linear interpolation. Subsequently, polygonization time reduces as the cache is populated. The ratio between cached times at consective resolutions decreases

because more cached voxels are re-used. At $512^3$ polygonizer resolution, approximately 33% of all cache voxels have been evaluated.

Since caches are cleared before each polygonization, it may seem un-intuitive that such a large gain would be made. However, as previously noted, the convergence iterations used to refine surface vertex positions (Section 2.9) frequently involve multiple evaluations in a single cache cell. With caching, these convergence steps require only a few tri-linear interpolations, instead of evaluation of thousands of point primitives.

While static polygonization time does not provide an indicative measure of interactive performance, fast static polygonization can still be useful in interactive modeling contexts. A standard interactive visualization technique is *iterative refinement*, where the designer interacts with a low-resolution visualization which is automatically refined during idle moments [29]. The example in Figure 4.7 shows that Hierarchical Spatial Caching is capable of providing significantly faster iterative refinement for implicit modeling.

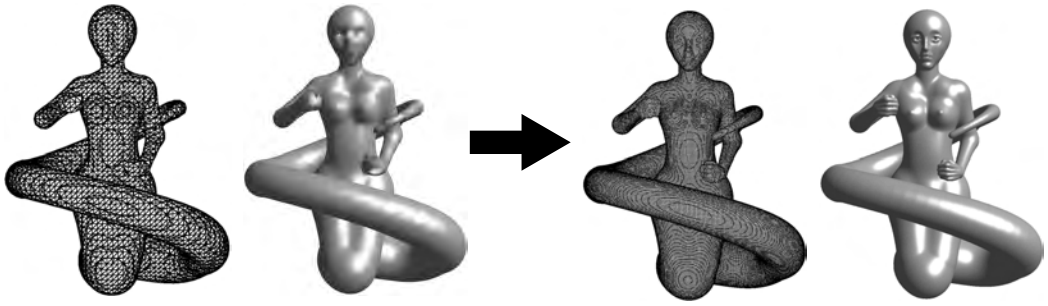


Figure 4.7: *Since the cache is already partially evaluated, iterative refinement of the low-resolution mesh on the left (computed at a polygonizer resolution of* $64^3$*) takes approximately 6 seconds. The high-resolution refined mesh on the right (computed at* $256^3$ *cube resolution) takes over 200 seconds to generate without hierachical spatial caching.*

### 4.8.2 Interactive Polygonization Time

While caching nodes can significantly reduce static polygonization time, the critical benefit becomes apparent during interactive manipulation of BlobTree model components. Update times measured during a a basic interaction test on the Medusa model are compared in Figure 4.8. The test involved simulated translation of the Medusa head component. The head was moved 25 steps towards the tail, then 25 steps back to the original position, as shown in the figure. Along this path the head intersects all other components of the model. An initial polygonization was computed before running each test, hence the cache begins partially evaluated. Polygonizer resolution was fixed at $60^3$ cubes, based on the initial model bounding box. Note that this is significantly lower than the resolution shown in Figure 4.8, see Figure 4.7 for a mesh of comparable resolution.

Polygonization time is relatively constant for the non-cached cases, which is expected. With caching nodes the polygonization time rapidly drops over the first few frames. Many of the caching node voxels for the head component are being evaluated over these frames. During the

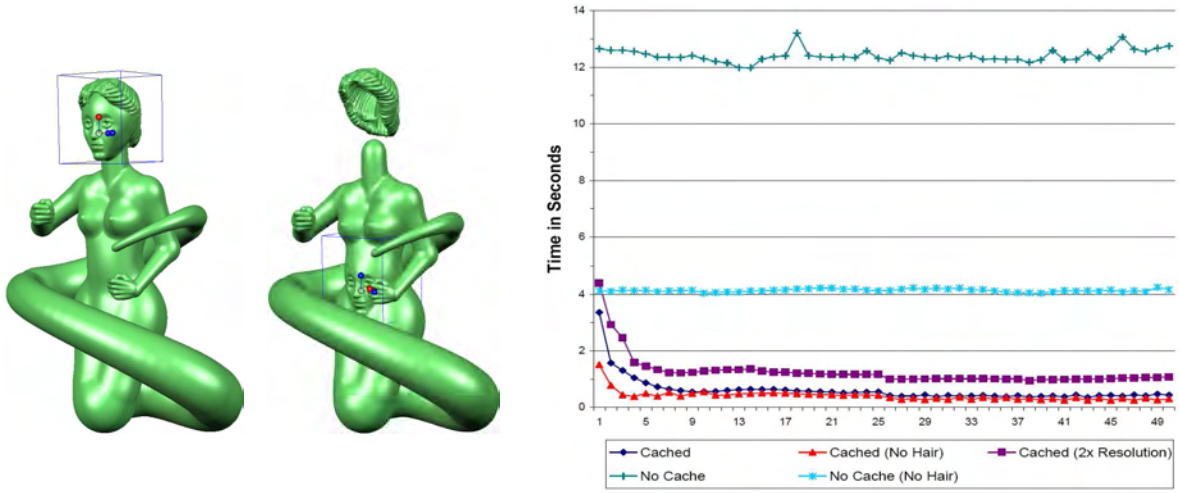



Figure 4.8: *Update time with various cache and model configurations for the head-translation test shown on the right. The animation frame is plotted on the X axis - frame 25 corresponds to the right image, frames 1 and 50 to the left. Hierarchical Spatial Caching is an order of magnitude faster in equivalent tests.*

rest of the downward path small numbers of potential field values are cached for other components as the head intersects them. This results in a relatively stable polygonization time. At frame 25 the upward path begins. The average stable polygonization time drops at this point because all the necessary field values have been cached on the downward pass.

Tests were performed with and without the hair component. The first few frames are computed more quickly without the hair component, however the stable polygonization time is essentially identical in both cases. This comparison shows that after the caches are populated, polygonization time is insensitive to the underlying model complexity and depends primarily on surface complexity. Finally, polygonizer resolution is doubled to $120^3$ cubes. The cached polygonization converges to a stable time of approximately 1.25 seconds per frame. Non-cached timings were over 50 seconds per frame.

### 4.8.3 Local Update Polygonization

A common technique for improving interactive visualization time in implicit modeling systems is to only re-polygonize the model in areas that have been updated. Several tests with and without caching nodes have been performed to analyze Hierarchical Spatial Caching in local update contexts. A modified version of the cubical decomposition polygonizer [26] was developed which only re-computes the surface inside a bounding region. At any frame, this update region is limited to the bounding box of model components that have changed. Because the same resolution is used as in the full polygonization case (Section 4.8.2), the cubes involved are a subset of those used for full polygonization. Hence, the same triangles are generated and polygonization timings can be directly compared between the two cases.

The interactive assembly task simulated described in Section 4.8.2 was performed using local

updates. The average timing improvements are shown in Table 4.2. Results from another test are also shown, in which only the hair component was loaded. A small point primitive was then dragged through the hair interactively. The bounding box of the point primitive covered less than 5% of the total hair volume.

| Test | Cubes | Ratio |
|---|---|---|
| Head Translation | $60^3$ | 7× |
| Head Translation | $120^3$ | 12× |
| Point / Hair Test | $60^3$ | 30× |
| Point / Hair Test | $120^3$ | 47× |

Table 4.2: *Comparison of cached and non-cached polygonization times for local-update polygonization tests.* Cubes *column refers to polygonizer resolution,* Improvement *column shows number of times speed-up with caching nodes.*

### 4.8.4 Approximation Error

The test results listed in the sections above were all based on caches with a constant grid resolution of 128 (as described in Section 4.7.1). While this has been found to provide reasonable performance, clearly it is not optimal. Model components that are small or simple relative to the global model scale, such as the left hand and neck of the medusa model, do not require as high of cache resolutions to achieve an equivalent level of approximation accuracy (Figure 4.9). It is desirable to minimize cache resolution because it reduces the number of evaluations of the child field that are necessary, particularly during cache invalidation.

Perhaps a more pressing issue, however, is that of undersampling. For example, the right hand of the medusa model, which is cached as part of the upper body component, is always undersampled in Figure 4.9. In the current framework, the only way to resolve undersampling is to increase the uniform grid resolution, which has the side effect of increasing the number of child field evaluations necessary to populate the cache. However, since the grid is uniform, the effect is not local to the undersampled region - the entire cache becomes more expensive to evaluate.

In the Medusa example there is another solution. If the right hand were to be seperated, both it and the body component could be adequately represented at lower cache resolutions. While the designer could conceivably re-organize the model to reduce undersampling, an automatic scheme would be preferable. One possibility is to measure the approximation error at surface samples, such as the mesh vertices. This produces a rough map of approximation error on the surface. A visualization of such a map is shown in Figure 4.10. Using this information, undersampled regions can be detected.

Unfortunately, even if undersampling can be detected, it is unclear how it can be resolved. In the Medusa example, bounding-volume heuristics could be used to automatically re-organize the model tree by separating the right hand. However, this may not always be possible - a “black

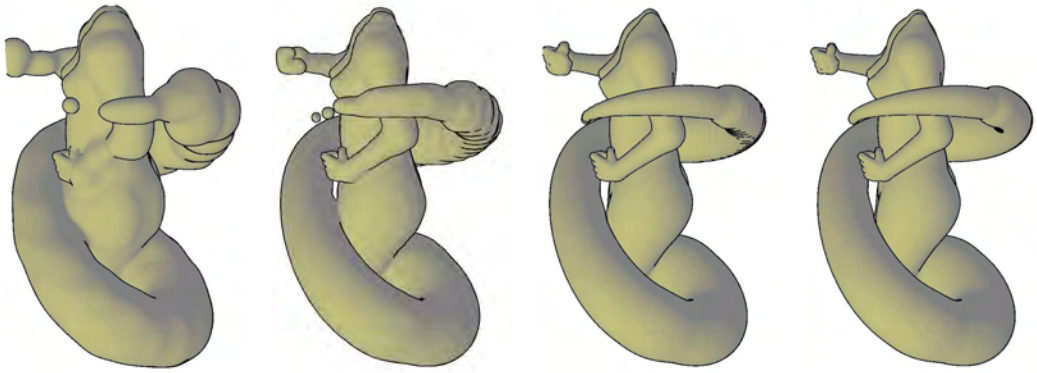



Figure 4.9: *Medusa model with component cache resolutions of (left to right)* 16, 32, 64, *and* 128. *Even at very low cache resolution, the left hand component is represented adequately. In contrast, the right hand (which is part of the much larger upper body component) is undersampled even at* 128 *resolution. Tri-quadratic value sampling is used at lower resolutions to increase surface smoothness.*

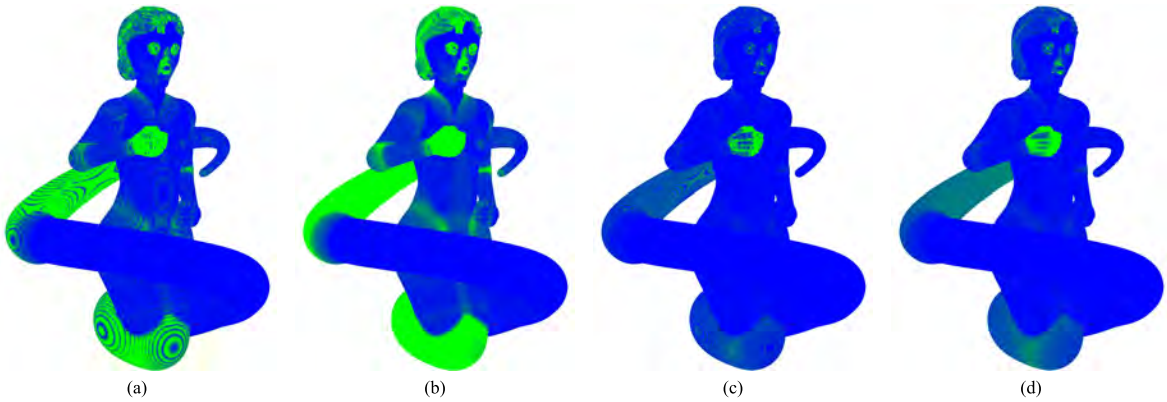



Figure 4.10: *Approximation error on the Medusa model. Mesh vertices are colored based on the absolute difference between the approximated field value and the accurate field value. The color scale runs from blue (low error) to green (high error). The images shown are (a)* $128^3$ *tri-linear reconstruction, (b)* $128^3$ *tri-quadratic, (c)* $256^3$ *tri-linear, and (d)* $256^3$ *tri-quadratic. Since (b) and (d) use approximation rather than interpolation, they have higher error than (a) and (c), which is clearly visible in these images.*

box" tree node may contain small details that cannot be segmented. Alternatives here include using a multi-resolution cache with locally adaptive resolution, or simply by-passing the cache in the undersampled region. Both these solutions are problematic. As described in Section 4.9.4, existing multi-resolution spatial caching schemes do not provide the $C^1$ continuity required for smooth surface visualization (Section 4.6). By-passing the cache introduces similar problems, and also may significantly reduce interactivity, since regions that are undersampled inconveniently correspond to the expensive, high-detail portions of the model. A suitable approach to automatically reducing undersampling has yet to be developed.

One final sampling issue involves sharp features. This problem was discussed in Section 4.6, a 3D example is shown in Figure 4.11. While grid-based reconstruction can appear to reconstruct sharp features if the sampling rate is high enough [49], the sharp edge is necessarily lost when more closely examined. Some polygonization algorithms attempt to resolve sharp features by searching for large changes in gradient direction at adjacent mesh vertices [121, 66, 76]. These algorithms are quite effective when used with analytic gradients, as seen in Figure 4.11a. However, they perform poorly with gradients reconstructed using tri-linear and tri-quadratic filters, because of regular $C^1$ discontinuities in the former, and gradient smoothing in the latter. A variety of related techniques have been developed for implicit surfaces based on point-set interpolation (Section 3.5). Currently, these methods are only designed to approximate creases in surfaces - it is unclear that they could be adapted to general scalar field approximation.


Figure 4.11: *Comparison of representation of a sharp edge without caching (a), with a cache resolution of* 128 *(b), and* 256 *(c). The Extended Marching Cubes polygnizer is used, producing a clear sharp edge in (a), but having no effect in (b) and (c) due to gradient smoothing at the crease.*

## 4.9 Spatial Caching with Adaptive Distance Fields

In the previous sections, spatial caching of a scalar field $f(\mathbf{p})$ has been implemented using uniform grids. Consider the case where $f(\mathbf{p})$ represents a large surface which has very small details. To accurately represent the details, $f(\mathbf{p})$ must be sampled very finely. This finely-sampled voxel grid will require a significant amount of memory to store, but a more immediate problem for Hierarchical Spatial Caching is that the cost of computing all the samples becomes prohibitive.

An improvement would be to adjust the sampling rate based on the size of the local details in the scalar field. This local refinement process is commonly known as *adaptive sampling*.

Recently, Adaptive Distance Fields, or ADFs, have been proposed as a solution for adaptive sampling and approximation of distance fields [49]. In short, the uniform grid is replaced with a hierarchy of voxels known as an *octree*. For any particular cell containing the surface, an error function is computed by comparing tri-linearly reconstructed values in the cell with real evaluations of $f(\mathbf{p})$. If the error is too high, the cell is subdivided into 8 child cells. Subdivision is repeated until the maximum error in all leaf cells is below some tolerance.

ADFs have been applied successfully to interactive modeling [83]. Frisken et al claim that their ADF techniques can be applied to general scalar field approximation, and hence should be usable in Hierarchical Spatial Caching. However, a detailed study has shown that this is not the case.

### 4.9.1 Octree Subdivision

In [49],the following test is proposed to determine when to subdivide an octree cell. First, only octree cells containing the surface are considered for subdivision. The value of $f(\mathbf{p})$ is computed at 19 points - 12 samples at the middle of each cell edge, 6 at the center of each face, and one at the center of the cell. These real values are compared with the approximate values (computed by tri-linear reconstruction based on the samples stored at the cell corners). The maximum error is then compared with a global error tolerance, and the cell is subdivided if the estimated error is too high.

One issue with this test is that only cells containing the surface are subdivided. The result is that the distance field approximation becomes much less accurate further away from the surface (Figure 4.12). This poor approximation does not affect most ADF applications because the far-field is only used to determine inside-outside tests. However, the far-field is an integral component of blending operators in BlobTree modeling. The poor far-field approximation causes blending artifacts and hence is unnacceptable. All cells must be considered for subdivision, resulting in a significant increase in the size of the final ADF (Figure 4.12) These extra cells make dynamic construction of ADFs even more costly.

### 4.9.2 Dynamic Construction

Applications of ADFs largely assume that the ADF is pre-computed off-line. Interactive applications interrogate the ADF data structure, and in some cases interactively combine two existing ADFs [83] into a new ADF, but entire ADFs are not generated dynamically. However, hierarchical spatial caching enables interactive BlobTree manipulation precisely because the spatial caches are created dynamically through lazy evaluation. Dynamic ADF construction has significant overhead.

First, the subdivision tests described in the previous section require 19 samples of both the underlying scalar field $f$ and the current ADF, to determine whether the approximation error in a cell is acceptable. Consider the case when the ADF octree is initially empty. Evaluating the ADF at a single point $\mathbf{p}$ will require dynamic expansion of the octree down to a sufficient level. Many 19-sample tests will be required. Compare this with evaluating a tri-linear cell

(a)

(b)

(c)

Figure 4.12: *The ADF generated using Frisken et al's technique of only subdividing surface cells (b) produces 1385 voxels, but poorly approximates the original distance field (a). Subdividing everywhere (c) is visually more accurate but produces 7833 voxels, an increase of over 550%.*

in a dynamic uniform grid. At most 8 samples of $f$ will be computed. In theory, the cost of octree expansion is amortized when it is truncated at higher tree levels (where a larger cell is adequate). However, during interactive manipulation at lower polygonization resolutions, it is often the case that only a few leaf cells in the update region will actually be necessary before they are invalidated. Hence, the octree expansion can introduce significant overhead.

The ADF error test also introduces extra overhead at leaf cells. Expansion continues until a leaf cell passes the error test, which requires 19 samples of $f$ and the ADF. However, unlike during octree expansion, at a leaf cell these samples are not stored - they are discarded. This is a considerable expense. If the leaf cell is re-used many times than this expense amortized, but during interactive manipulation any leaf cells in the update region will be destroyed and re-generated each frame. Again, this is a significant overhead.

These construction costs are significant problems for any dynamic adaptive approximation scheme to be used in Hierarchical Spatial Caching. Approximation error is traditionally measured by sampling, however the primary means for increasing interactive visualization speed is to avoid computing many samples! It may be possible to formulate error tests based entirely on existing samples, by measuring values such as (approximated) curvature or gradient magnitude. This would mitigate the overhead at leaf nodes. The adaptive expansion overhead is more challenging. One possible approach is to replace the global octree with a uniform grid of cells which are themselves octrees. If the uniform grid resolution were adequate, only the octree-cells in higher-detail regions would be expanded beyond the root node. However, determining a suitable uniform octree-grid resolution is problematic. Note that this approach also solves the expansion problem.

One objection to these statements is that the Kizamu system [83] supports interactive editing of ADFs, by using the ADF of a "carving tool" to add or subtract from the exising ADF. However, the demonstrated examples in [83] generally show carving of small details on a large surface. It is not clear that large-scale editing operations are interactive. Second, Kizamu only supports binary CSG compositions. Visually, CSG operations are simply a composition of two surfaces, so Kizamu only updates the ADF within the visual bounding box of the tool. This is not acceptable in a BlobTree modeler, the entire support region of the tool's scalar field must be updated. Finally, as noted in section 4.9.1, ADF subdivision only occurs in cells containing the surface. The additional octree expansion (Section 4.9.1) required for accurate scalar field approximation is avoided.

### 4.9.3 Sampling Cost

The entire purpose of hierarchical spatial caching is to reduce the cost of sampling $f$. The more efficient a spatial cache is to sample, the more interactive the modeling system will be. Hence, it is important to consider the cost of sampling an ADF.

ADF's apply the tri-linear filter to reconstruct the distance field approximation at a point $\mathbf{p}$. This filter is ideal for the octree data structure, as only the 8 corner samples of the leaf cell containing $\mathbf{p}$ are required. However, finding the leaf cell requires $O(\log N)$ steps. Frisken et al [50] optimize this leaf search by replacing the standard recursive algorithm with an iterative algorithm, however $O(\log N)$ iterations are still required. After the leaf cell is found, the 8 corner vertices are located through indirect hash-table lookups - at best an amortized $O(1)$ cost.

In comparison, the blocked uniform grid scheme described in section 4.7.3 supports direct lookup of the bottom-left corner of the cell containing **p**. In the best case, all the necessary values are stored in a single contiguous block which (if properly sized) fits entirely in a CPU cache line. In the worst case, each necessary value is in a separate block, however these values are located directly by indexing into the separate blocks. The total cost is significantly lower than the $O(\log N)$ octree traversal and 8 hash table lookups necessary for an ADF.

### 4.9.4 Continuity

The issues with ADFs described thus far have all been related to computational constraints, which in the future may become less restrictive. However, a more fundmental problem with ADFs is that the scalar fields generated by ADFs are not $C^0$ continuous. As noted in Section 2.6, $C^0$ continuity is a requirement for consistent implicit modeling.

Consider the simplest case in 2D, where the neighbour of cell A is subdivided once (Figure 4.13). In cell A, the value at the midpoint of the edge between values $a$ and $b$ is obtained by linear interpolation. However, in the adjacent cells, the value at the midpoint of edge E is one of the stored "true" samples of $f$. If $f$ is not linear along this edge, a $C^0$ discontinuity will necessarily exist in some neighbourhood of E. For most $f$, the discontinuity will exist along the entire edge. These discontinuities are clearly visible in Figure 4.12. As explained in Section 2.6, $C^0$ discontinuities result in holes in the surface, and hence ADFs are incompatible with the basic scalar field requirements for implicit modeling.

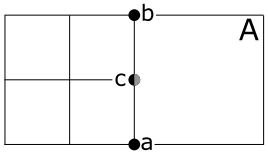



Figure 4.13: $C^0$ *discontinuty between octree cells.*

## 4.10 Chapter Summary

The interactive visualization problem has prohibited the use of hierarchical implicit modeling systems such as the BlobTree in interactive shape modeling tools. In this chapter, an approach has been developed for improving interactive visualization times. Hierarchical spatial caching combines aspects of existing caching schemes, namely traversal caching and spatial caching, to reduce the cost of evaluating internal branches of a BlobTree model from $O(N)$ to $O(1)$. This novel technique results in an order-of-magnitude improvement in visualization time. A variety of implementation options have been discussed, as well as several optimization techniques used in the testing implementation.

In an attempt to reduce approximation error, the octree-based Adaptive Distance Field technique was investigated for use in hierarchical spatial caching. Unfortunately, no viable approach was found. It is clear that interactive construction of ADFs has significant computational costs

which have yet to be addressed. Finally, and most conclusively, the lack of a $C^0$ reconstruction scheme prohibits the use of octree ADFs for representation of valid implicit volumes.

One possible way forward may be to move to multiple partition-of-unity (MPU) schemes, such as the MPU-RBF [87] approach. Currently, these methods are not designed for interactive construction, have a relatively high sampling cost, and an expensive error metric. If these issues can be addressed, MPU approximation schemes may be usable as a drop-in replacement for regular sampling in hierarchical spatial caching.

# Chapter 5

# Implicit Sweep Surfaces

*BlobTree-compatible implicit sweep objects are developed which support direct specification and manipulation of the surface with no topological limitations on the 2D sweep template. A 2D distance field approximation technique is described which is $C^2$ smooth except at sharp creases, and is easily converted to a bounded field. Several sweep endcap techniques are introduced, and several applications are considered.*

*Some material in this chapter is taken from the publication "Generalized Sweep Templates for Implicit Modeling" by Schmidt and Wyvill [96]. The material appearing here is due to Schmidt.*

## 5.1 Implicit Sweep Surfaces

Sweep surfaces, nearly ubiquitous in parametric modeling for the last 20 years [85], are a useful interactive modeling metaphor. However, sweep surface methods are very limited in the hierarchical implicit modeling domain [27]. Existing implicit sweep representations compatible with the BlobTree are limited to star-shape sweep profiles defined by offset curves [37]. Implicit sweep techniques that support the intuitive direct profile manipulation available in the parametric domain have not been previously developed for the BlobTree.

One of the key problems regarding implicit sweeps is the definition of a suitable 2D sweep template scalar field. In the parametric domain a sweep surface is generated by sweeping a 2D template curve $\mathcal{C}$ along a 3D trajectory $\mathcal{T}$. In the implicit domain, a 2D template scalar field $f_{\mathcal{C}}$ must be created which represents $\mathcal{C}$ and can be swept along $\mathcal{T}$. To create well-defined 3D volumes, $\mathcal{C}$ must be a closed contour. Also, $f_{\mathcal{C}}$ should be bounded and continuous if the scalar field created by sweeping $f_{\mathcal{C}}$ is to be used in the BlobTree. In particular, gradient discontinuities in the field away from the surface must be avoided to prevent blending artifacts.

In this chapter, a technique is described for generating 2D sweep template fields that can be used to create sweep volumes which have good blending properties and are compatible with the BlobTree (Section 5.3). First, a simple technique is described for converting distance fields to bounded fields, allowing any set of closed 2D contours, including nested "hole" contours, to be used to create an implicit sweep template. Then, existing variational curve-fitting techniques are extended to create a distance field approximation which is $C^2$ away from the surface, providing good blending properties, but preserves creases in the sweep template contour.

The approach to used to sweep the template scalar field largely follows classical methods (Section 5.4). One issue which has not been previously discussed is the generation of different sweep endcap styles. Three novel sweep endcap styles are described - flat endcaps with sharp and rounded edges, and a smooth variable-width endcap.

The goal of this sweep formulation is to create implicit sweep volumes compatible with the BlobTree. Within this framework, implicit sweeps can be composed with other implicit volumes

(including other implicit sweeps). These sweep primitives have been implemented in an interactive BlobTree modeling environment (Chapter 6). To reduce evaluation time in interactive contexts, the sweep template $f_{\mathcal{C}}$ is discretely approximated (Section 5.6).

Since $\mathcal{C}$ is directly approximated by $f_{\mathcal{C}}$, the resulting implicit sweep surfaces are nearly identical to parametric sweep surfaces created by sweeping $\mathcal{C}$ (Section 5.7). The implicit formulation transparently handles difficult degenerate cases, such as self-intersecting sweeps (Section 5.5), that are problematic in the parametric formulation. In addition, parallel contours represented implicitly can simply be interpolated to reconstruct 3D surfaces (Section 5.8).

In interactive contexts, the ability to directly manipulate the surface contour is much more efficient than the previous offset-contour methods. The interaction techniques developed for parametric sweeps can be used without modification. Combined with their good blending behavior, the implicit sweep primitives developed in this chapter are an intuitive and expressive free-form addition to BlobTree modeling.

## 5.2 Previous Approaches

Solid modeling by sweeping a 2D area (commonly referred to as a *sweep template*) along a 3D trajectory is a well known technique in computer graphics [85, 47]. Most early CAD systems [86] supported sweep surfaces as boundary representations (B-reps). These B-rep sweep solids lacked a robust mathematical foundation, self-intersecting sweeps were simply invalid. Recent work on the more general problem of sweeping a 3D volume has produced several general swept-volume envelope computation techniques which employ either the sweep-envelope differential equation [23] or Jacobian rank-deficiency conditions [3]. Applications of these techniques are explored in a recent survey [1]. These methods operate on closed-form boundary representations which are symbolically manipulated. The volume boundaries can be defined using closed-form implicit surfaces [2]. However, the resulting swept volume is still a boundary representation, and not an implicit volume defined such as in Equation 2.3. Sourin et al [108] describe a technique for sweeping implicit volumes. Evaluating the resulting function requires slow non-linear optimization algorithms.

There are essentially two approaches to creating implicit sweep surfaces. One method is to convert the 2D sweep template to a 2D scalar field, and then sweep this scalar field in 3D. Alternatively, a discretization of a 3D parametric sweep surface (such as a triangle mesh or point set) can be approximated with an implicit representation.

Creating an implicit representation of an arbitrary 3D surface is non-trivial. Some success in this domain has been achieved using *scattered-data interpolation* techniques, where an implicit surface is defined that passes through a set of 3D sample points. One approach, variational implicit surfaces [113, 92], involves solving a dense matrix and hence does not scale to more than a few thousand sample points - insufficient to adequately represent a complex sweep surface. The cost of solving the linear system can be greatly reduced using Fast Multipole Methods, permitting millions of sample points to be used [34]. Unfortunately the multipole expansions necessary to implement this technique were not provided in [34] and have not been published. In addition, these approximation techniques cannot represent sharp creases or points, since the

variational implicit surface is globally $C^2$ continous. Interpolation of point values is generalized in the concept of transfinite interpolation [91], which can interpolate arbitrary geometric curves and preserves discontinuities. However, this technique has only been demonstrated for a restricted set of closed-form 2D implicit contours.

Another class of scattered-data interpolation techniques have been developed that employ basis functions with local support [72]. The underlying linear systems are sparse and hence more efficient to solve. Partition of unity techniques such as MPU implicit surfaces [75] and the Partition of Unity Variational method [87] apply hierarchical approximation techniques to simplify surface fitting and can represent sharp features in certain configurations. However, these techniques are not guaranteed to define an implicit volume (Equation 2.3). Additional internal iso-surfaces may be produced as a result of blending local approximations with finite support [87]. Volume modeling operations such as CSG may expose these internal iso-surfaces. The Implicit Moving-Least-Squares (IMLS) technique [105] converts triangle meshes into implicit volumes and hence could be used to define implicit sweep surfaces. However, the implicit surface interpolates the tessellation - the differential properties of the sweep template are lost, local curvature is always zero (or infinite on the creases between each triangle), and an adequate tessellation level must be determined.

[102] and [116] approximate sweep surfaces with volume datasets. The sweep dataset is initialized by sampling a sweep template specified as a 2D image. Volume datasets cannot represent features smaller than a single voxel. Surface creases and sharp points are also lost in the conversion process. Adaptively-Sampled Distance Fields (ADFs) [49] attempt to deal with these issues, but contain $C^1$ discontinuities in the reconstructed scalar field.

Implicit sweeps can alternatively be defined by following the parametric approach of sweeping a 2D template. One key issue is the definition of a suitable implicit template. A 2D scalar field must be defined that represents the closed 2D template contours. This scalar field is then swept along the sweep trajectory.

[37] describes implicit sweep primitives with bounded sweep template fields. Profile curves defined in polar coordinates are used to create an anisotropic distance field. The sweep surface is an offset surface from the swept profile curve, prohibiting direct surface specification. Profile curves are limited to "star-shapes" by the polar definition. [55] describes implicit generalized cylinders which have a similar limitation.

Another approach is to approximate the contour with a polygon, and then construct an implicit representation of the polygon [80, 17]. These methods require that non-convex polygons be carefully decomposed into pieces that can be properly combined using CSG intersection operations. This decomposition is non-trivial. The properties of the resulting scalar field are dependent on the tessellation level (Figure 5.1d). These issues can be mitigated using explicit normalization techniques [22]. A side-effect of this normalization is that offset iso-contours quickly converge to a circle (Figure 5.1e). While *Huygen's Principle* guarantees that all distance fields will converge to circular iso-contours in the limit, in this case it happens quickly and has an undesirable effect on blending.

A swept 2D implicit polygon results in a faceted 3D sweep surface. Curvature at the surface is not preserved, and gradient discontinuities exist along the sweep paths of each vertex in the 2D polygon. The 2D polygon can be smoothed at the vertices [81]. However, this "corner-

cutting" reduces the accuracy of the approximation as the surface no longer passes through the accurate surface samples (the polygon vertices). An implicit representation of 2D curves has been described [80], but is based on an underlying "carrier polygon" that has only been developed for cubic polynomial splines. Variational interpolation can also be applied to approximate a 2D contour [125]. The standard technique involves constraining the variational solution only near the contour, the rest of the field is unconstrained and hence determining bounds is very difficult (Figure 5.1c).

2D implicit curves can be represented using level set techniques [77]. These algorithms are based on 2D pixel discretizations, and hence have the same issues as volume datasets regarding small and sharp features. Skeletonization techniques such as Medial Axis Transform [30] can be used to compute distance fields which approximate curves but also contain $C^1$ discontinuities.

To summarize, several implicit sweep models have been proposed that support direct surface specification and manipulation. However, none of these models produce the bounded scalar fields necessary for interactive modeling of complex hierarchical implicit models [120, 97].

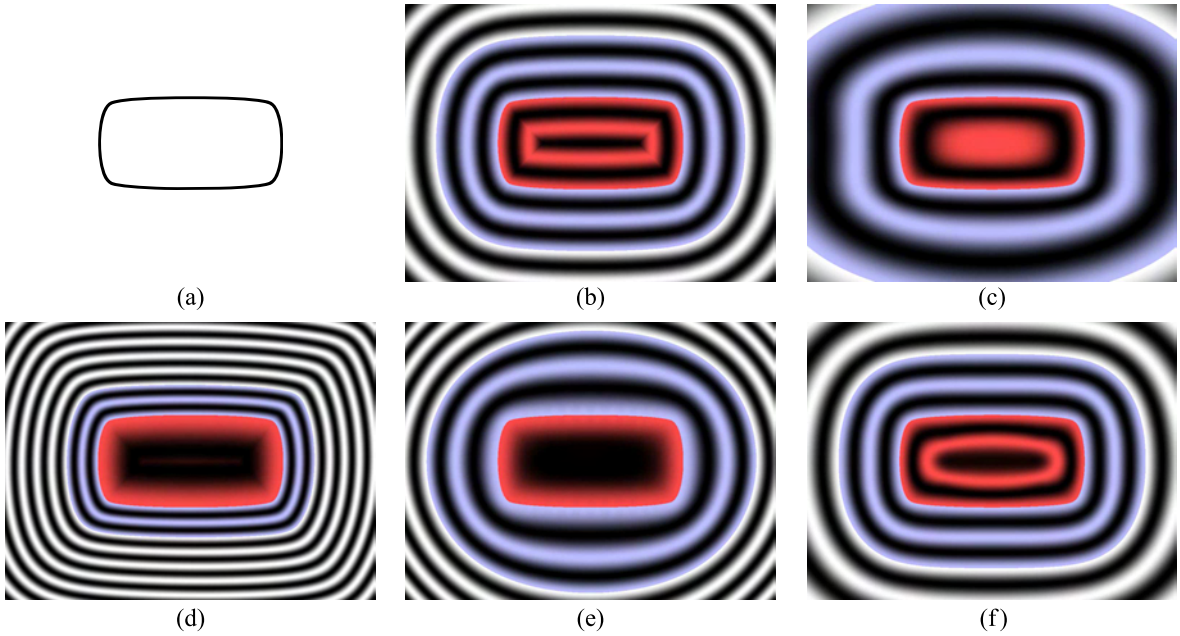



Figure 5.1: *Scalar fields generated from contour (a), with filter* $(1 + \sin(k \cdot f))/2$ *applied to the field before mapping to grayscale to emphasize iso-contours. Red highlight indicates region inside contour, blue highlight indicates bounded region after applying Equation 2.12. Exact distance field (b) has* $C^1$ *discontinuities inside contour. Variational approximation (c) using normal constraints provides a poor approximation to (b). Approximation with a* $C^1$ *implicit polygon (d) is sensitive to the tessellation of the curve, field values grow as tessellation level is increased. A normalized implicit polygon (e) is accurate near the contour, but the iso-contours quickly tend to a circular shape. The variational approximation with boundary constraints proposed in this chapter (f) closely approximates (b) inside the falloff region but is globally* $C^2$ *continuous.*

## 5.3 2D Implicit Sweep Templates

In the following section a method is developed for creating a 2D implicit sweep template which represents a closed 2D contour $\mathcal{C}$. First, a technique for converting the signed distance field $d_{\mathcal{C}}$ to a bounded 2D field is described. This simple method allows BlobTree-compatible implicit sweep surfaces to be generated from any set of closed 2D contours. Then, a method is developed for creating a smooth approximation $\widetilde{d_{\mathcal{C}}}$ to the exact distance field $d_{\mathcal{C}}$ which does not contain undesirable $C^1$ discontinuities. This approximation technique, which is based on existing work which approximates curves using variational interpolation [113], cannot reproduce sharp edges in $\mathcal{C}$. To mitigate this problem, a technique for re-introducing sharp edges into the sweep template is described, followed by a comparison of the distance field approximation with existing methods.

### 5.3.1 Bounding Distance Fields

The approach used to create skeletal primitives - applying $g_w$ (Equation 2.12) to a distance field - can be adapted to create the sweep template field $f_{\mathcal{C}}$. In the distance field $d_{\mathcal{C}}$ the curve lies on the zero iso-contour, $d_{\mathcal{C}} = 0$. If $g_w$ is applied directly to $d_{\mathcal{C}}$, the iso-contour defined by $g_w \circ d_{\mathcal{C}} = v_{iso} \neq 0$ will be offset from $\mathcal{C}$. As noted above, sweeps generated by offset iso-contours generally do not support direct manipulation and hence are difficult to control. To force this contour to lie on $\mathcal{C}$ the distance values must be shifted. First, $d_{\mathcal{C}}$ must be converted to a signed distance field, where points inside $\mathcal{C}$ have a negative distance value (Section 2.2). The shifted distance $d'_{\mathcal{C}}$ is then:

$$d'_{\mathcal{C}} = min(g_w^{-1}(v_{iso}) + d_{\mathcal{C}}, 0) \tag{5.1}$$

The bounded 2D sweep template field $f_{\mathcal{C}}$ is then defined as

$$f_{\mathcal{C}} = g_w \circ d'_{\mathcal{C}} \tag{5.2}$$

To give some sense of what this equation accomplishes, consider Figure 5.2. When a potential function is applied to the distance field of a standard skeletal primitive, such as the line in 5.2a, an offset curve is produced. For sweeps, this final curve is known, and the task is instead to find a skeleton which would generate the desired offset curve, as shown in 5.2b. However, this is not always possible. Curves with sharp edges, such as the rectangle in 5.2c, cannot be generated by a constant-offset curve from any skeleton, since constant-offset curves always round off sharp convex edges, as shown in 5.2d. Equation 5.1 avoids this difficulty by directly generating a scalar field with the desired iso-contour. Hence, a skeleton is implicitly defined by this equation - however, the input curve is not necessarily a constant-offset from this skeleton (as in Figure5.2c).

Since $\mathcal{C}$ may be non-convex, $d_{\mathcal{C}}$ can contain $C^1$ discontinuities (Figure 5.4a), and hence so will the bounded field $f_{\mathcal{C}}$ (Section 2.7). These discontinuities will be swept in 3D and produce undesirable blending artifacts (Figure 2.6a). To avoid this problem an approximation to $d_{\mathcal{C}}$ must be developed which has a smooth field away from the surface.

### 5.3.2 $C^2$ Distance Field Approximation

An implicit approximation to a 2D contour $\mathcal{C}$ can be created using variational interpolation [113]. A $C^2$ interpolating thin-plate spline is fitted to a set of constraint points placed at samples of $\mathcal{C}$.

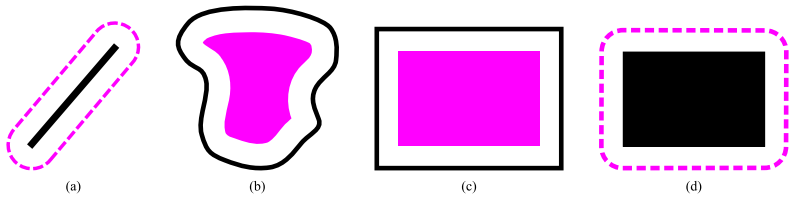



Figure 5.2: *Skeletal primitives, such as the line in (a), generate offset contours (shown in magenta). Equation 5.1 implicitly defines the "skeleton" area for an arbitrary curve, as shown in (b). However, the skeleton defined for the curve in (c) could not be used to directly re-produce the curve, as the offset contour smooths out convex corners.*

To adequately constrain the solution, *normal constraints* [34, 125] are added. Inner and outer normal constraints at a sample point are added at short distances along the normal to $\mathcal{C}$ at the on-curve sample (Figure 5.3a).

Normal constraints only ensure that the iso-contour passes through the sample points. The scalar field is unconstrained further from the surface, resulting in a poor approximation to the distance field (particularly in the case of non-convex $\mathcal{C}$, see Figure 5.4). This is especially problematic if the resulting function is to be bounded, since it is non-trivial to determine the bounding region of the non-zero field values without resorting to a time-consuming spatial search.

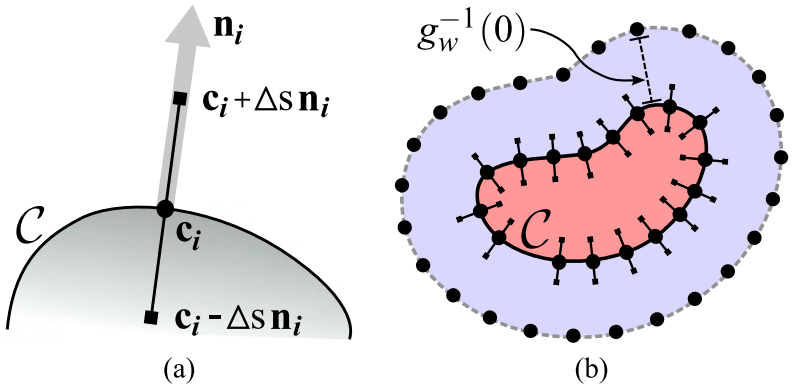



Figure 5.3: Normal constraints *(a) at a point* $\mathbf{c}_i$ *are added at short offset* $\Delta$ *s from the curve* $\mathcal{C}$*, along the curve normal* $\mathbf{n}_i$*.* Boundary constraints *(b) are placed at a constant distance from* $\mathcal{C}$ *to improve the distance field approximation and ensure that the field* $f_{\mathcal{C}}$ *is bounded within a known distance.*

Variational approximation produces an implicit curve, rather than polygon. Since only sample points are necessary, there are no restrictions on $\mathcal{C}$. However, normal constraints alone are insufficient to produce a variational solution which approximates a distance field. To constrain the variational solution in regions further from the curve, *boundary constraints* are added (Figure 5.3b).

Boundary constraint points are generated by sampling several iso-contours of the distance field. A reasonable distance field approximation can be obtained using one set of boundary

constraints along the iso-contour found at a distance of $g_w^{-1}(0)$. After applying $g_w$, this contour bounds the non-zero field values, and hence can be used to determine a bounding region. In the examples shown in this section, two additional iso-contour constraints have been used, one at $g_w^{-1}(0.5 * v_{iso})$, which is approximately half-way between $\mathcal{C}$ and the zero-contour, and another at $g_w^{-1}(1.5 * v_{iso})$, which lies inside $\mathcal{C}$. The purpose of these extra constraints is to further reduce error in the distance field approximation.

If $\mathcal{C}$ has high-frequency features, there may be higher error in the distance field approximation close to the curve. In interactive modeling with implicit sweeps, this error is not particularly critical. However, the situation can be identified automatically by comparing approximated distance values with exact distance values at small offsets from $\mathcal{C}$. If the measured error is unacceptable, several directions can be taken. The first is to increase sampling density along the existing iso-contour constraints. In practice, sampling each iso-contour constraint with between 50 and 100 uniformly-spaced sample points produced satisfactory results in all test cases. For curves with very small features, higher sampling rates, or non-uniform importance sampling, may be necessary. To further reduce approximation error, additional sets of iso-contour constraints close to $\mathcal{C}$ can be inserted.

Tracing iso-contours in the exact distance field $d_{\mathcal{C}}$ is non-trivial and computationally intensive. To accelerate computation, $d_{\mathcal{C}}$ is approximated on a discrete 2D grid using a *distance transform* [61] algorithm. Approximate iso-contours can then be efficiently extracted from the distance transform. The test implementation uses the CSSED algorithm which runs in linear time in the number of image pixels [38]. The distance transform is computed on a $512^2$ pixel image and then discrete iso-contours are traced in the image. The resolution here is not critical, similar results have been found with $128^2$ pixel images.

Alternatives to this iso-contour-sampling technique have been found to produce unsatisfactory results in limited testing. Since the variational solution minimizes global curvature, constraints based on either random or regular sampling can result in unexpected oscillations. In addition, regular sampling introduces many spurious constraint points and if the sampling frequency is too high the discontinuities in the distance field are closely approximated. When these high-frequency areas are blended, the perceptual discontinuities described in Section 2.8 can occur. In theory, this problem can also occur with the iso-contour sampling described above, since iso-contours in the distance transform approximate the non-$C^1$ iso-contours in the distance field. This problem can be avoided by replacing the distance transform with a fast-marching level set method that propagates at curvature-dependent speeds [77]. The discrete iso-contours produced by this algorithm smooth out the $C^0$ discontinuities in the distance field, minimizing the chance that perceptual discontinuities will be introduced.

Adding boundary constraints to the variational solution results in a much more accurate approximation to the distance field, particularly in the case of non-convex curves (Figure 5.4). The variational scalar field is a globally $C^2$ continuous approximation to $d_{\mathcal{C}}$.

### 5.3.3 Sharp Features

The variational approximation to $\mathcal{C}$ is globally $C^2$ continuous. This implies that sharp features, which are $C^1$ discontinuities in $\mathcal{C}$, are not preserved in the variational approximation. Significant

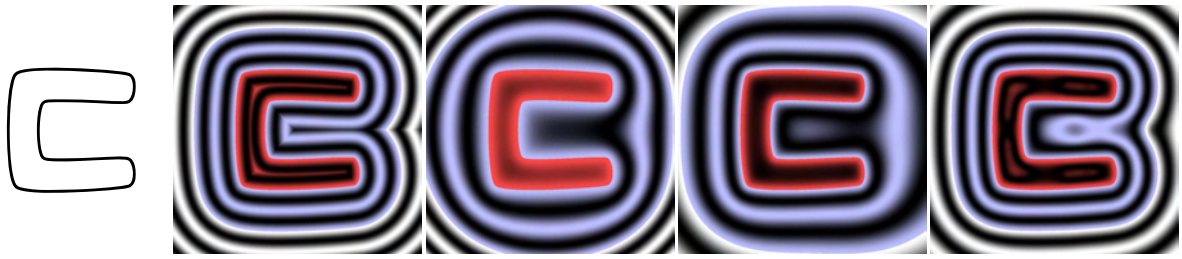



Figure 5.4: *Scalar fields generated using a non-convex curve (a). The exact distance field (b) has $C^1$ discontinuities inside and outside the curve. Approximation with a normalized implicit polygon (c) and variational approximation with normal constraints (d) provide poor approximations in concave region. Our approach (e) smoothly approximates the distance field away from the surface.*

smoothing can be observed near creases, even with very high sampling rates (Figure 5.5a). Crease approximation can be improved by using anisotropic basis functions [43] but these basis functions still do not create sharp $C^1$ discontinuities. Sharp features are preserved with implicit polygon techniques [22, 17]. To re-introduce sharp features into the constrained variational approach a constructive approach is taken which incorporates these existing techniques. The $C^2$ smooth distance field approximation described above is used for most of the field, however in a small disc around sharp features, a crease-preserving implicit polygon approximation $\widehat{d_{\mathcal{C}}}$ is used. To preserve continuity, these fields are blended in a transition region around each feature region.

A set of crease positions $\mathbf{c}_i$ are assumed to be known, as is a *feature radius* $r_i$. When evaluating $f_{\mathcal{C}}$ at a 2D point $\mathbf{u}$, the distance $k$ from $\mathbf{u}$ to the nearest feature point is computed. The blended distance field approximation $d^{\star}_{\mathcal{C}}$ is then defined as follows:

$$d^{\star}_{\mathcal{C}} = \begin{cases} \widetilde{d_{\mathcal{C}}} & \text{if } k \leq r \\ (1 - g_w(\frac{k}{r}))\widetilde{d_{\mathcal{C}}} + g_w(\frac{k}{r})\widehat{d_{\mathcal{C}}} & \text{if } r < k < 2r \\ \widehat{d_{\mathcal{C}}} & \text{if } k \geq 2r \end{cases} \tag{5.3}$$

See Figure 5.6 for a graphical representation of the blending regions. The use of $g_w$ here is not significant, any smooth interpolating function can be used. This method is similar to the bounded-blending techniques used to control implicit volume composition operators [82, 52]. An example is shown in Figure 5.5.

In the examples shown, the normalized implicit polygon approach [22] is used to define the crease-preserving field $\widehat{d_{\mathcal{C}}}$. The necessary equations are described in Appendix B. To locate crease points on $\mathcal{C}$, the rudimentary method of thresholding the angles between consecutive line segments in the polygonal approximation is used. This technique is not robust but works reasonably well for most creases.

This blending approach provides the benefits of boundary-constrained variational approximation while also locally preserving sharp features. Since this method provides a more accurate distance field approximation (Figure 5.1), it may be more appropriate than implicit polygon approaches even when $\mathcal{C}$ is actually a polygon. However, there are some drawbacks. In the blending

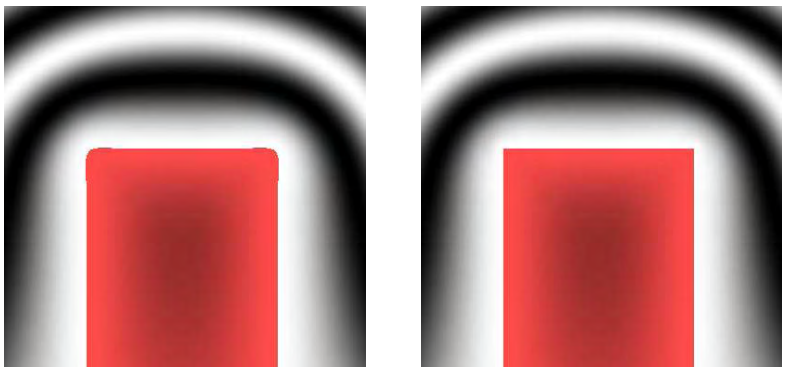



Figure 5.5: *The boundary-constrained variational method described in section 5.3.2 rounds out sharp edges (a) because it is globally $C^2$ continuous. Sharp features can be re-introduced (b) by blending between an implicit polygon and the variational field in the feature regions (see section 5.3.3).*

regions around feature points the scalar field may not be monotonic, since the two fields being combined can be increasing at different rates. As previously noted, curvature in the feature region is not preserved by the polygonal approximation.

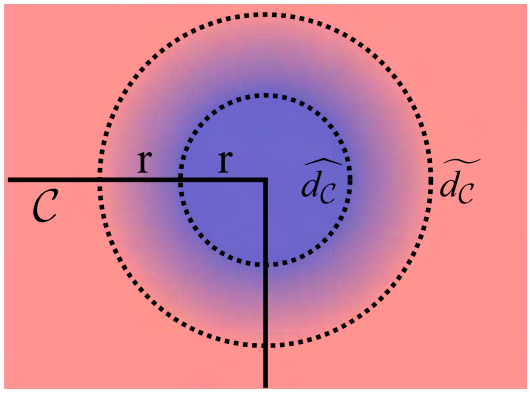



Figure 5.6: *A scalar field $\widehat{d_C}$ that preserves sharp features in $\mathcal{C}$ can be combined with a smooth approximation $\widetilde{d_C}$. Within a distance $r$ from the sharp feature, $\widehat{d_C}$ is used. Outside the radius $2r$, $\widetilde{d_C}$ is used. Between $r$ and $2r$ (dashed circles in diagram), $\widetilde{d_C}$ and $\widehat{d_C}$ are smoothly blended.*

### 5.3.4 Analysis

A variety of implicit representations for general 2D curves have been developed (Section 5.2). The scalar fields generated by some of these techniques are compared in Figures 5.1 and 5.4. In both of these examples it is clear that the technique described in this section more closely approximates a distance field than existing methods. Close approximation to a distance field is desirable because it allows easy computation of field bounds and results in more predictable

blending behavior. The technique used to generate Figure 5.1d, for example, is sensitive to the tessellation level of the curve. If the tessellation level is increased, the iso-contour spacing will change and hence blend surfaces will also change. This is unacceptable. The normalized polygon approach [22] does not have this problem; however, the iso-contours produced approach a circular shape very quickly (Figure 5.4c). This results in undesirable blending behavior (Figure 5.7). In both of these cases, the scalar field represents a polygon, not a curve. The curvature of the input contour $\mathcal{C}$ is lost, and hence the differential surface properties of the implicit sweep do not reflect those that would be computed with a parametric sweep. Although it is still an approximation, the variational sweep template produces a smooth curve and hence approximates the curvature of the accurate sweep surface.

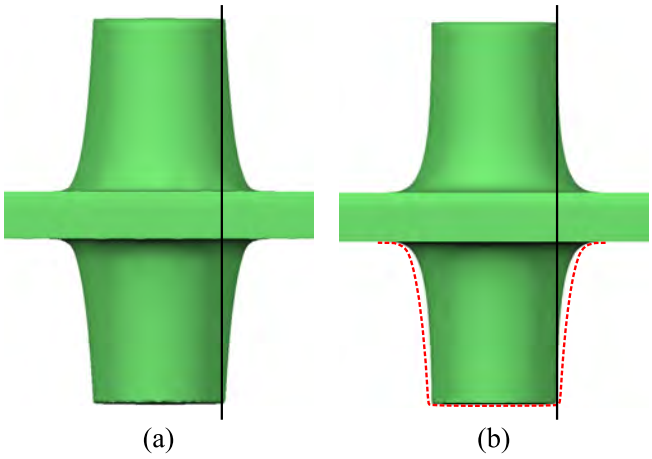


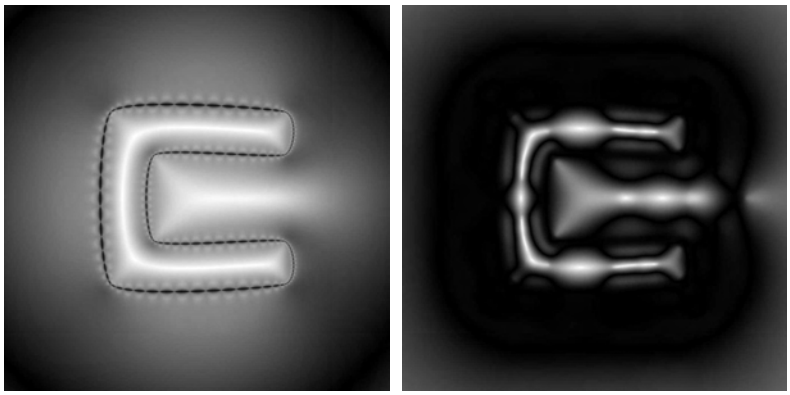

Figure 5.7: *Implicit sweep blended with cylinder. Sweep template is computed using normalized implicit polygon in (a), and boundary-constrained variational approximation in (b). Black line shows edge of non-blended cylinder, dashed red line in (b) shows outline of (a). Blending in (b) is more localized and has a softer transition.*

Defining "goodness" measures for distance field approximation is challenging. One test is to compare the approximated field values with the exact distance field values. However, since the whole point of distance field approximation is to avoid the discontinuities that occur in the exact distance field, zero error is in fact undesirable. Still, this test has been performed with a variety of convex and non-convex curves. On average, variational curves with boundary constraints reduce the mean error by a factor of 25 over the normalized implicit polygon approach.

As noted in Section 2.9, distance fields are normalized ($|\nabla f| = 1$ everywhere except along the discontinuity curves). Again, if the goal is to smooth discontinuities in a distance field, minimizing the gradient magnitude error $|1 - |\nabla f||$ is undesirable as it will produce high-frequency $C^2$ regions that are effectively perceptual discontinuities (Section 2.8). However, this is still a useful error metric because near-normalization is a desirable property. Variational curves with boundary constraints have a mean normalization error that is on average a factor of 4 lower than the implicit polygon approach.

Normalization images (Section 2.9.1) are a useful tool for visually inspecting the normalization error over a field. Two normalization images are shown in Figure 5.8. In both cases, the

normalization error is highest along what would be discontinuities in the exact distance field (inside $\mathcal{C}$, the discontinuities lie on the medial axis). In the implicit polygon approach (Figure 5.8a), the error is very low near $\mathcal{C}$ but increases rapidly away from the curve. The variational curve with boundary constraints has somewhat higher error near $\mathcal{C}$ but is much more accurate inside the bounded approximation region further from the curve.



(a) (b)

Figure 5.8: *Gradient images computed by mapping* $|1 - |\nabla f||$ *to grayscale (white is maximum error). Normalized implicit polygon (a) has low error near surface, but higher error than the boundary-constrained variational technique (b) in regions further from the surface.*

## 5.4 Sweep Primitives

An implicit sweep primitive is defined by a 2D *sweep template field* $f_M$ (Equation 5.2) and a 3D *sweep trajectory* $\mathcal{T}$, which is assumed to be a parametric curve defined by one parameter:

$$\mathcal{T}(s) = (t_x(s), t_y(s), t_z(s)), \qquad 0 \leq s \leq 1$$

The sweep surface $\mathcal{S}$ is defined by a 3D scalar field $f_{\mathcal{S}}$. Given a point $\mathcal{T}(s)$ on the sweep trajectory, a geometric transformation $\mathbf{F}(s)$ can be defined which maps 2D points $\mathbf{u}$ to 3D points $\mathbf{p}$. $\mathbf{F}(s)$ is assumed to be affine and maps the 2D origin to $\mathcal{T}(s)$, implying that points $\mathbf{F}(s) \cdot \mathbf{u}$ are co-planar (Figure 5.9).

In the following discussion, 0 is appended as the third coordinate when converting 2D point $\mathbf{u}$ to 3D point $\mathbf{p}$. Going from 3D to 2D, the third coordinate is simply dropped. Application of this conversion will be determined by context and not explicitly denoted.

A possible definition for $f_{\mathcal{S}}$ is:

$$f_{\mathcal{S}}(\mathbf{p}) = f_M(\mathbf{u}), \qquad \mathbf{F}(s) \cdot \mathbf{u} = \mathbf{p} \tag{5.4}$$

Unfortunately, given only a point $\mathbf{p}$ this equation cannot be directly evaluated because the parameter value $s$ and 2D point $\mathbf{u}$ are unknown. Since $\mathbf{F}(s)$ is affine the inverse mapping $\mathbf{F}^{-1}(s)$ can be computed:

$$f_{\mathcal{S}}(\mathbf{p}) = f_M(\mathbf{u}), \qquad \mathbf{u} = \mathbf{F}^{-1}(s) \cdot \mathbf{p} \tag{5.5}$$

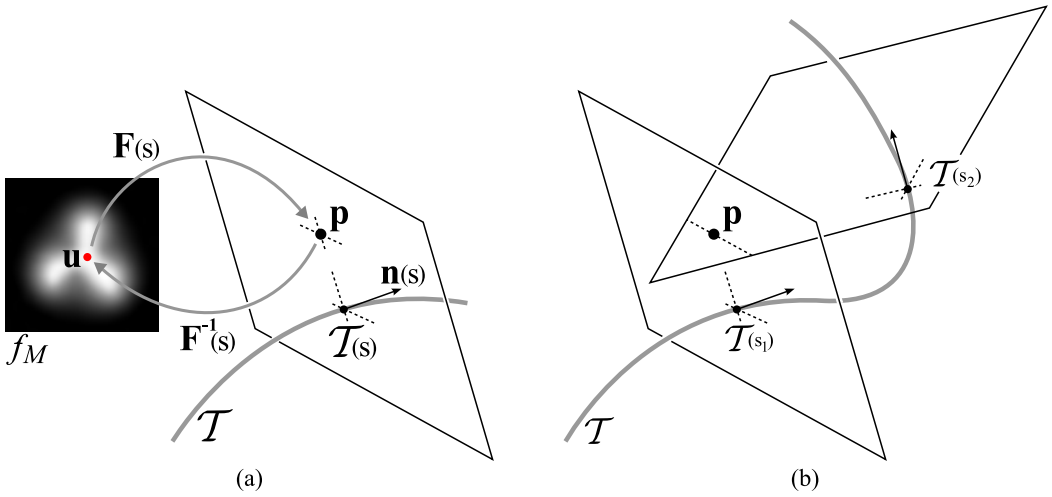



Figure 5.9: *Forward and inverse mapping from sweep template field $f_M$ to 3D space (a). Function $\mathbf{F}(s)$ maps 2D point $\mathbf{u}$ to 3D point $\mathbf{p}$ lying in plane defined by $\mathcal{T}(s)$ and $\mathbf{n}(s)$. Inverse map $\mathbf{F}^{-1}(s)$ maps from $\mathbf{p}$ to $\mathbf{u}$. In (b), point $\mathbf{p}$ lies on planes at points $\mathcal{T}(s_1)$ and $\mathcal{T}(s_2)$. Unique inverse mapping $\mathbf{F}^{-1}(s)$ does not exist.*

However, $s$ is still unknown, and not necessarily unique (Figure 5.9b).

$\mathbf{F}(s)$ transforms the 2D plane into some 3D plane passing through $\mathcal{T}(s)$ with normal $\mathbf{n}(s)$ (Figure 5.9). Since there may be more than one plane that passes through $\mathbf{p}$ (Figure 5.9b), a set of parameter values $\mathbb{S}(\mathbf{p}) = s_i$ is defined such that $\mathbf{p}$ lies in the plane at $s_i$:

$$\mathbb{S}(\mathbf{p}) = \{s_i \ : \ (\mathbf{p} - \mathcal{T}(s)) \cdot \mathbf{n}(s) = 0\} \tag{5.6}$$

To compute the field value $f_{\mathcal{S}}(\mathbf{p})$ Equation 5.5 is evaluated for each parameter value $s_i \in \mathbb{S}(\mathbf{p})$. The resulting field values are combined with a composition operator $\mathcal{G}$, such as a CSG union operator [88], to produce a single field value. The final definition:

$$f_{\mathcal{S}}(\mathbf{p}) = \mathcal{G}\left( \left\{ \ f_M(\ \mathbf{F}^{-1}(s_i) \cdot \mathbf{p}\ ) \quad : \quad s_i \in \mathbb{S}(\mathbf{p}) \right\} \right) \tag{5.7}$$

can be evaluated for any trajectory where the parameter set $\mathbb{S}(\mathbf{p})$ is computable.

### 5.4.1 Linear Trajectory

A simple and efficient sweep primitive is the linear sweep, or *extrusion*, where the sweep trajectory is a straight line between points $\mathbf{a}$ and $\mathbf{b}$. In this case $\mathbf{F}(s)$ is unique, $s = (\mathbf{p} - \mathbf{a}) \cdot \mathbf{n}$ where $\mathbf{n}$ is the unit vector $(\mathbf{b} - \mathbf{a})/\|\mathbf{b} - \mathbf{a}\|$. Given two mutually perpendicular vectors $\mathbf{k}_1$ and $\mathbf{k}_2$ which lie in the plane defined by $\mathbf{n}$, and origin $\mathbf{o}$, $\mathbf{F}^{-1}_{linear}(s)$ is defined as:

$$\mathbf{F}^{-1}_{linear}(s) = \mathbf{Rot}\begin{bmatrix}\mathbf{k}_1 & \mathbf{k}_2 & \mathbf{n}\end{bmatrix} \cdot \mathbf{Tr}\left[-((\mathbf{a} - \mathbf{o}) + s\mathbf{n})\right] \tag{5.8}$$

where $\mathbf{Rot}\begin{bmatrix}\mathbf{k}_1 & \mathbf{k}_2 & \mathbf{n}\end{bmatrix}$ is the homogeneous transformation matrix with upper left 3x3 submatrix $\begin{bmatrix}\mathbf{k}_1 & \mathbf{k}_2 & \mathbf{n}\end{bmatrix}^\top$ and $\mathbf{Tr}[-((\mathbf{a} - \mathbf{o}) + s\mathbf{n})]$ is a homogeneous translation matrix with translation

component $-((\mathbf{a}-\mathbf{o})+s\mathbf{n})$. Twisting and scaling can be introduced into the sweep surface by varying $\mathbf{k}_1$ and $\mathbf{k}_2$ along the trajectory.

The linear sweep scalar field $f_{linear}$ is then defined as

$$f_{linear}(\mathbf{p}) = f_M\left(\mathbf{F}^{-1}_{linear}(s)\cdot\mathbf{p}\right)$$

The function $f_{linear}$ defines an scalar field of infinite extent along $\mathbf{n}$ which must explicitly be bounded to cap the ends of the sweep surface. The field should extend for some distance $d_{endcap}$ beyond the sweep line endpoints to permit blending. It is desirable to have some control over the endcap shape. Existing implicit sweeps create endcaps by linearly interpolating the sweep template values to zero [37]. Three alternative endcap styles are now defined.

**Flat Endcap With Sharp Transition** An infinite linear sweep can be capped with flat endcaps by computing the Boolean intersection of the sweep with a rectangular box. Barthe's [19] $C^1$ intersection operator creates a sharp edge on the cap while still preserving $C^1$ continuity. To optimize computation, the rectangular box can be replaced with two infinite planes. Note that the intersection operator must be applied to the entire sweep to preserve continuity. Hence, between the infinite planes (in the sweep region) a field value of 1 is used. In Figure 5.10 a flat endcap created using this method is compared with one created using Ricci's intersection operator [88], which creates $C^1$ discontinuities

Figure 5.10: *Flat endcap with sharp transition. $C^1$ discontinuity on surface produces sharp crease (left) but does not propagate away from the surface with Barthe CSG intersection (middle). Ricci intersection produces $C^1$ discontinuity away from surface (right). Isocontour is marked in red.*

**Flat Endcap With Smooth Transition** An alternative to the sharp endcap above is to a flat endcap with a smooth transition region. Hybrid $C^1$ CSG/blending operators that smooth out the CSG transition zone but are otherwise identical to sharp CSG operators are defined by [19]. Intersection with infinite planes is again used to smoothly transition from the sweep surface to a flat endcap. The size of the transition zone is controllable, Figure 5.11 shows endcaps with varying parameter values. This operator converges to a $C^1$ field discontinuity as the transition zone size decreases.

**Blobby Endcap** A third style of endcap is related to the linear interpolation to 0 used in previous approaches. Essentially, the goal is to scale $f_{linear}$ down to zero at some distance

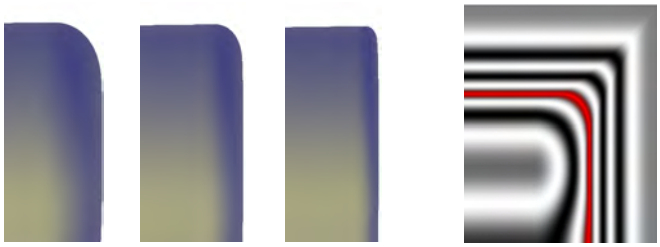


Figure 5.11: *Flat endcap style with smooth transition. Transition parameter varies from left to right* 30°, 35°, 40°. *Rightmost image shows scalar field for* 35° *parameter. Isocontour is marked in red.*

beyond the end of the sweep. Instead of linear interpolation, a smoother interpolating function is used - in this case, $g_w$ (Equation 2.12). This function is applied to the distance past the end of the sweep, and used as a scaling factor for the field value $f_{linear}$. A parameter $d_{endcap}$ determines the extent of the field beyond the end of the sweep line. If the parameter value at the line endpoint is $s_{max}$, the function in the endcap region is:

$$f_{endcap}(\mathbf{p}) = g_w\left(\frac{|s| - s_{max}}{d_{endcap}}\right) \cdot f_{linear}(\mathbf{p}) \tag{5.9}$$

Field continuity is preserved by the zero tangents of $g_w$. Because the value of $f_{linear}$ increases as the interior distance from $\mathcal{C}$ increases, the field takes "longer" to reach zero at points further from the $\mathcal{C}$. The result is an endcap with variable width (Figure 5.12). This variable width primitive with a free-form contour has been found to be extremely useful in interactive sketch-based modeling contexts (Section 6.4).

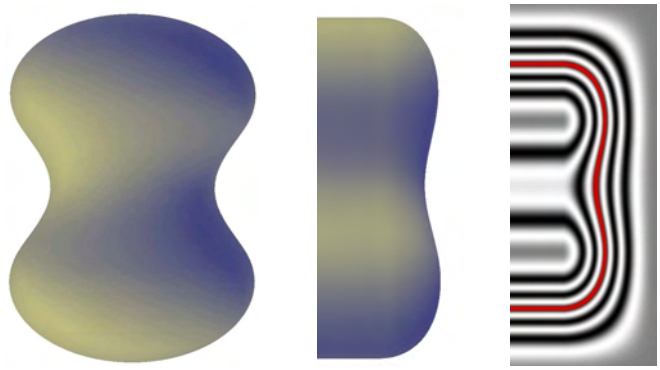


Figure 5.12: *Blobby endcap style. Sweep profile (left) determines width of endcap at each point (middle). Width is dependent on field value inside the profile (right). Isocontour is marked in red.*

### 5.4.2 Circular Trajectory

Circular sweep trajectories, described by a point $\mathbf{o}$ and a normalized axis $\mathbf{n}$, result in *surfaces of revolution.* Several restrictions are necessary to provide an implicit circular sweep definition that

is valid for any trajectory and sweep template. The sweep function, Equation 5.7, cannot be evaluated at points lying on the axis $\mathbf{n}$ because the set $\mathbb{S}(\mathbf{p})$ (Equation 5.6) is infinite. To define circular sweeps a simplifying assumption is made - that the template orientation and scaling is constant. Under this constraint the infinite parameter set maps to a single point $\mathbf{u}$.

Since twisting and scaling is disallowed, it is possible to compute point $\mathbf{u}$ in the sweep template corresponding to a point $\mathbf{p}$ without finding the angle $\theta$ on the circular trajectory. The 2D coordinates are found geometrically:

$$\mathbf{u}_y = (\mathbf{p} - \mathbf{o}) \cdot \mathbf{n}$$
$$\mathbf{u}_x = \|(\mathbf{p} - \mathbf{o}) - \mathbf{u}_y \mathbf{n}\|$$

The 2D coordinates corresponding to the opposing side of the circle are $(-\mathbf{u}_x, \mathbf{u}_u)$. These sample points result in two field values which can be composed with a $C^1$ union operator [19] to produce a gradient-continuous scalar field.

Note that $\mathbf{u}_x \geq 0$, implying that the portion of the sweep template to the left of the axis of revolution never sampled. This can be desirable as it is analogous to revolving an open curve around an axis between the curve endpoints. Unfortunately it also produces a $C^1$ discontinuity at points on $\mathbf{n}$ where $f_M$ is not continuous $C^1$ continuous across the $Y$ axis.

Open sweeps (sweeps along some arc segment of a circular curve) can be created by computing the sweep angle [116] and applying the linear-sweep endcap techniques. Also, if the axis of revolution lies outside the bounds of $f_M$, twisting can be safely applied, although for full circular sweeps the twist angle needs be a whole multiple of $2\pi$ to avoid $C^0$ discontinuities. A similar restriction applies for scaling.

### 5.4.3 Cubic Bezier Trajectory

As an example of a general trajectory, consider a 3D cubic Bezier curve $\mathcal{B}(s)$, $s \in [0, 1]$. Assume that the normal function $\mathbf{n}(s)$ (Equation 5.6) is the tangent vector $\mathcal{B}'(s)$. The following polynomial in $s$ must be solved to find the parameter value set $\mathbb{S}(\mathbf{p})$:

$$(\mathbf{p} - \mathcal{B}(s)) \cdot \mathcal{B}'(s) = 0 \tag{5.10}$$

This polynomial is degree 5 for a cubic Bezier curve, precluding analytic solution. Numerical root-finding techniques such as [100] must be used to solve for the roots $s$.

Once $\mathbb{S}(\mathbf{p})$ is computed, $\mathbf{F}^{-1}_{bezier}(s)$ can be evaluated:

$$\mathbf{F}^{-1}_{bezier}(s) = \mathbf{Rot}\begin{bmatrix}\mathbf{k}_1 & \mathbf{k}_2 & \mathcal{B}'(s)\end{bmatrix} \cdot \mathbf{Tr}\begin{bmatrix}-\mathcal{B}(s)\end{bmatrix}$$

Care needs be taken when defining the vectors $\mathbf{k}_1$ and $\mathbf{k}_2$. One option is to calculate the *Frenet frame* [47]:

$$\mathbf{k}_1 = \mathcal{B}'(s) \times \mathcal{B}''(s)$$
$$\mathbf{k}_2 = \mathbf{k}_1 \times \mathcal{B}'(s)$$

However $\mathcal{B}''(s) = \mathbf{0}$ in degenerate cases such as straight sections. The Frenet frame can also flip direction across points of inflection, creating $C^0$ discontinuities. An alternative is the rotation

minimizing frame of [25]. This frame is procedurally defined based on an initial frame at $\mathcal{B}(0)$ and cannot be calculated analytically. In order to generate a frame at any $s$, a table of rotation minimizing frames $\mathbf{F}_j$ can be calculated at points $s_j$ along the curve. $\mathbf{F}^{-1}_{bezier}(s)$ is then found by locating the nearest $s_j < s$ and computing a rotation minimizing frame from $\mathbf{F}_j$. This technique is known as Bishop framing [21].

The cap styles described in Section 5.4.1 are all applicable to arbitrary curved trajectories. Twisting and scaling can also be applied based on $s$ or an arc-length parameterization of $\mathcal{B}(s)$.

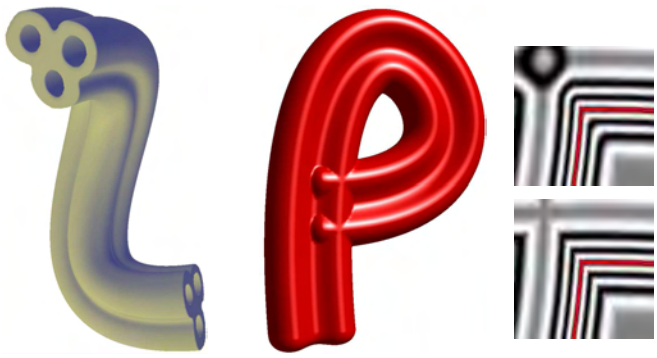


Figure 5.13: *Bezier curve sweeps. Holes in template are supported (left). Scalar field in self-intersecting cases (middle) is $C^1$ continuous using Barthe CSG union operator (right top). Ricci union operator produces $C^1$ discontinuities (right bottom).*

## 5.5 Self-Intersection

A key problem with sweep surfaces is that they are frequently self-intersecting (Figure 5.14). In the B-rep domain self-intersection is a critical problem [86], since the definition of 'inside' and 'outside' is often based on the sweep surface and hence is ambiguous inside self-intersections. These problems are not an issue in volumetric implicit modeling. Since a point in space has only one field value, there is no ambiguity as to whether or not it lies inside the volume.


Figure 5.14: *Self intersection can occur near high-curvature regions of the sweep trajectory (a) as well as in overlapping regions (b), (c). Cutaway mesh in (d) shows manifold surface.*

The behavior of volumetric implicit sweeps in self-intersecting cases is entirely determined by the operator $\mathcal{G}$ (Equation 5.7). $\mathcal{G}$ should be applied to all field values produced with the

solutions to Equation 5.10, as well as the endcap field values. The CSG Union operator of [88] can be used as $\mathcal{G}$. This produces a closed manifold surface but also gradient discontinuities in the scalar field (Figure 5.13). A $C^1$ continuous CSG union operator [19] can be used to preserve both the surface creases and gradient continuity in the field away from the surface. Using this formulation, self-intersections are automatically handled. The final implicit volume is always defined as the outermost surface of the sweep.

## 5.6 Discrete Sweep Templates

There are two costs associated with using the sweep formulations described above in an interactive system. The first cost is the initial solution of the thin-plate spline with boundary constraints (Section 5.3.2). While the resulting linear system (Appendix A) is an $O(N^3)$, the number of constraint points is usually less than one thousand. Using a SIMD-optimized LAPACK implementation, the dense symmetric matrix can be solved with double precision in a few seconds on modest hardware[1].

However, compared to the cost of evaluating $f_M$, the cost of finding the thin-plate spline solution is minimal. Each evaluation of $f_M$ requires an evaluation of equation B.1, which involves a log and *sqrt* call for each constraint point. With over a thousand constraint points, a single evaluation of $f_M$ becomes quite costly.

As noted in Chapter 4, bounded scalar fields can be approximated with discrete uniform sampling. In the sweep template case, $f_M$ only requires a 2D spatial cache. For most smooth sweeps, $128^2$ field images provides good results without requiring excessive sampling time. Since uniform sampling is used, error can only be reduced by globally increasing the field image resolution. To maintain interactivity, lazy evaluation (Section 4.4) can be employed.

As in the 3D case, bi-quadratic sampling can be applied to produce a $C^1$ approximation to $f_M$. However, the crease approximation technique (Section 5.3.3) is not so easily integrated. One option is to replace $\widetilde{d_{\mathcal{C}}}$ with the cached value, and blend with the functional implicit polygon. This is a reasonable approach, since the functional implicit polygon is much more efficient to evaluate, however it has not been implemented.

## 5.7 Parametric to Implicit Sweep Conversion

Many existing parametric modeling tools define sweep surfaces using a boundary representation. In this case the sweep template is the contour $\mathcal{C}$, usually defined parametrically, and the sweep trajectory is a 3D parametric curve $\mathcal{T}(s)$. As previously noted, the boundary representation has several limitations. Operations such as CSG, or even simple point-inclusion testing, require complex numerical techniques. Self-intersecting sweeps are difficult to handle [1] because the definition of "inside" is ambiguous. Algorithms for measuring physical properties, such as volume, often produce invalid results in these degenerate cases.

The implicit sweep representation described in this section easily handles degenerate conditions, CSG is well-supported [88, 120, 19], and point-inclusion testing is trivial using the implicit

[1] Test systems included a 1.4Ghz Intel Pentium 4 laptop and a 1.6 Ghz Intel Centrino laptop.

volume definition (Equation 2.3). Conversion from parametric sweeps to implicit sweeps is trivial because the existing parametric contour $\mathcal{C}$ and trajectory $\mathcal{T}$ (Section 5.4) can simply be re-used.

An important consideration when converting parametric sweeps to implicit sweeps is error introduced by the conversion. The 2D sweep template scalar field is generated using variational techniques that only approximate $\mathcal{C}$. However, by increasing the sampling rates for the various constraint curves the variational solution can be tightly constrained. An example of parametric conversion is shown in Figure 5.15. In this case the maximum surface deviation is less than 0.5%, which occurs at the high-curvature features.

Interactive tools for implicit modeling can take advantage of this coherence between parametric and implicit representations. The sweep surface can be quickly visualized during interactive manipulation using parametric techniques. When manipulation is complete, the implicit model is re-polygonized.

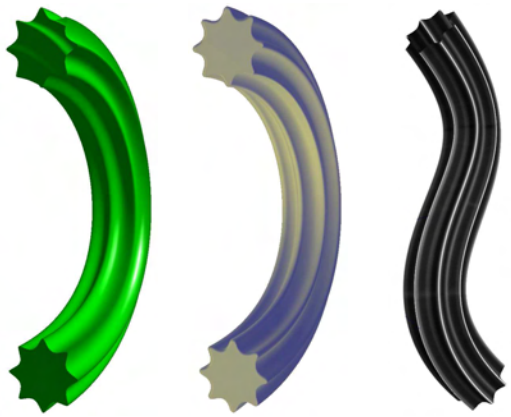


Figure 5.15: *Parametric sweep (left) and corresponding implicit sweep (middle). Parametric sweep vertices are colored by implicit approximation error (right), where white is the highest error. Error values have been linearly scaled by* 1000 *to improve visibility.*

## 5.8 Surface Reconstruction from Parallel Contours

It is possible to interpolate between the two sweep template fields $f_{M_1}$ and $f_{M_2}$ along a sweep with respect to a parameter $t \in [0, 1]$:

$$f_{M_{interp}}(t) = (1 - t)f_{M_1} + (t)f_{M_2}$$

A sweep surface with varying cross section can be created by sweeping the template $f_{M_{interp}}(s)$, where s is scaled to $[0, 1]$. The resulting surface smoothly transitions between the two templates regardless of topology. Interpolation between $n$ templates is accomplished by successively interpolating between pairs of templates. Given a set of contours $\mathcal{C}_i$, a closed surface can be reconstructed that passes through all the contours.

A variety of techniques for implicit surface reconstruction from parallel contours are available [92, 6, 34, 125]. Generally these methods produce complex field functions that are expensive

to evaluate. One benefit of the approach described above is that it is very efficient to evaluate, particularly when the sweep templates are approximated with discrete versions (Section 5.6). In Figure 5.16 a human L3 vertebra is reconstructed from 93 segmented CT slices. It takes approximately 6 minutes to create the set of high-resolution parallel sweep templates, however once complete the implicit object (which is simply a linear sweep) can be interactively edited using the system described in Chapter 6 (Figure 5.16c).

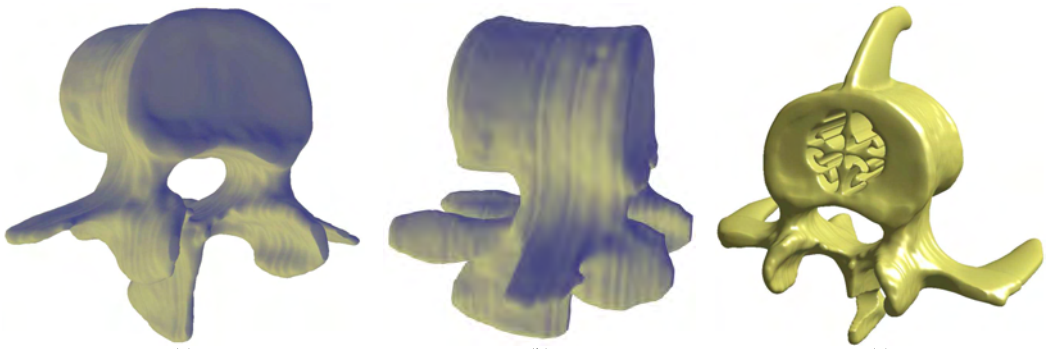



Figure 5.16: *Surface reconstruction from contours. Human L3 vertebra is reconstructed from 93 slices (a,b) and interactively modified in a BlobTree modeling system (c). The high-frequency ridges on the vertebra surface in (b) are due to manual segmentation errors.*

## 5.9 Chapter Summary

A technique has been described for generating 3D implicit sweep volumes that are compatible with the BlobTree hierarchical modeling system. The main contribution is the development of a method for converting a 2D sweep template contour $\mathcal{C}$ (or set of contours, including "holes") into a bounded, continuous 2D scalar field which is used as an implicit sweep template. First, *boundary constraints* were added to existing variational curve approximation techniques, resulting in a $C^2$ distance field approximation with low normalization error. A constructive field-blending approach was used to preserve sharp features in the sweep template. The approach to sweeping the 2D sweep template field largely follows existing methods, although the integration of various endcap styles into the implicit sweep formulation is a necessary detail that has not been previously described.

A key benefit of the sweep volumes described in this chapter is compatibility with BlobTree hierarchical implicit modeling. Existing implicit sweeps either lacked direct profile manipulation, or did not have BlobTree-compatible scalar field properties. The sweep formulation presented here combines the desirable attributes of these previous approaches. Particularly in an interactive context, these new sweeps provides an expressive, intuitive free-form primitive for the BlobTree. Complex shapes such as the set of pipes in Figure 5.17 can be quickly created by blending multiple sweep surfaces. More examples are shown in Chapter 7.

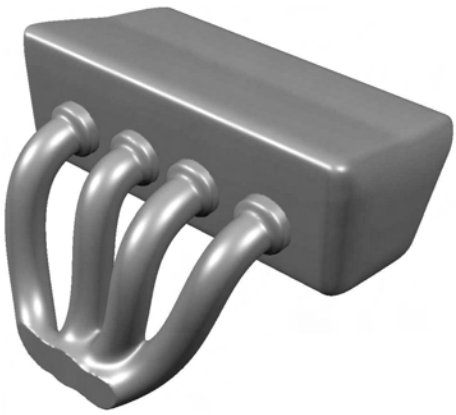


Figure 5.17: *An intake manifold modeled with Bezier curve sweeps.*

# Chapter 6

# ShapeShop: A Proof-of-Concept

*Existing interactive implicit modeling systems are reviewed. A new interactive BlobTree modeling system, known as ShapeShop, is introduced. Various aspects of the interface are reviewed, with a particular focus on sketch-based operations. Novel implementation challenges are discussed, followed by suggestions for future work.*

*Some material in this chapter is taken from the publication "ShapeShop: Sketch-Based Solid Modeling with BlobTrees" by Schmidt, Wyvill, Sousa, and Jorge [99]. The material appearing here is due to Schmidt.*

## 6.1 Interactive Implicit Modeling Systems

Interactive implicit-surface modeling has been a long sought-after goal in computer graphics. A variety of systems described in the literature have claimed success, however the interpretation of the term "interactive" varies considerably. For example, a variety of hierarchical implicit surface modeling environments have been developed, notable examples include the various incarnations of the BlobTree Modeling System [51] and the HyperFun system [4]. In the latter system, claims of "interactivity" mean only that the underlying skeleton can be manipulated interactively - the surface is not visualized in real-time. Similarly, the convolution-surface modeling system described by Sherstyuk [106] supports interactive editing of the convolution skeleton, and provides compelling results, however the implicit surface was rendered off-line using ray-tracing.

An early implicit modeling system which did support interactive visualization was reported by Desbrun [42], where interactivity was achieved by creating a separate polygonization for each primitive in the model. This approach is essentially a local-update scheme, closely related to [5] and sharing the same limitation; namely, that if the update region is complex then interactivity fails. To reduce overall computation, level-of-detail (LOD) implicit surfaces were explored by Barbier et al [15]. This technique, first proposed by Bloomenthal and Wyvill [29], uses lower-resolution versions of primitives to speed visualization time. Per-primitive polygonization and level-of-detail were combined to rapidly visualize implicit surfaces based on skeletal subdivision curves [58]. However, this method was limited to surfaces which can be described by a skeleton composed of branching curves. In a similar vein, Akleman [7] employed ray-quadric surfaces which supported interactive polygonization but were limited to star-shaped volumes.

Witkin and Heckbert [117] proposed a general technique for interactive implicit surface visualization based on "floater" particles, and applied it in an accompanying 3D modeling system. This approach was extended by Hart [56], however that system was limited to constructing surfaces which could be defined by a single multi-dimensional polynomial. In both cases, only relatively simple models could be constructed.

Variational implicit surfaces (Section 3.5.1) were initially proposed as a method for interactive

implicit surface modeling [113, 114]. Recent sketch-based modeling systems have adapted this method [65, 13]. However, the computational cost of solving for the variational solution limits this approach to relatively simple models. A similar issue limits the sketching system described by Alexe et al [10], which is based on blending large numbers of implicit point primitives.

Distance fields were combined with mesh subdivision techniques in the Skin system [68]. While very smooth results were produced, and the technique is quite interactive [59], it does not address the problems encountered with real-time visualization of general implicit surfaces.

Much more promising results have been produced by systems which employ implicit surfaces based on discrete volume datasets. The sculpting system described by Ferley et al [45] is highly interactive and supports adaptive volume resolution [46]. Adaptive distance fields have also been employed in an interactive volume sculpting system [83]. Owada et al [78] describe a sketch-based modeling system for volumetric data. Volume datasets can be interactively deformed and manipulated using level set techniques [73], and interactively functionally composed [18]. However, as noted by [18], hierarchical techniques cannot easily be applied to volume datasets due to the massive amount of sample data which must be stored and manipulated.

Several conclusions can be drawn from the review above. First, no existing system supports interactive manipulation and editing of complex hierarchical implicit models. Many of the systems described above support some aspects of hierarchical functional implicit modeling, but the general framework is not exploited. Second, systems based on implicits generated from volume data have generally supported higher levels of interactivity and more complex models. Attempts to create a synthesis of these two methods lead directly to the hierachical spatial caching technique described in Chapter 4. In the following sections a novel interactive hierarchical implicit modeling system is described which, rather then exploiting specific properties of the various primitives and operators, relies on this new caching technique to produce interactive visualization.

## 6.2 Overview

The ShapeShop hierarchical implicit modeling system began development as a testbed for evaluating hierarchical spatial caching (Chapter 4). The initial system consisted of a few basic BlobTree primitives and operators [120] which could be created and interactively manipulated in a traditional CAD-style interface. The BlobTree model was visualized using a basic polygonization algorithm [26], the surface was fully re-polygonized when changes were made to the model.

The CAD-style interface, while functional, was cumbersome and tedious to use. This was not entirely due to the actual modeling interface, however. Constructive modeling with the basic BlobTree primitives (sphere, cylinder, etc [120]) requires many primitives and operators to create complex shapes. Creating and placing all these elements is a time-consuming process. To reduce construction time, the free-form sweep primitives described in Chapter 5 were added to the system, allowing complex shapes to be created using far fewer primitives. However, instead of creating sweep primitives by interactively manipulating parametric curves, a novel sketch-based construction method was implemented [99]. This modification had a significant impact on how models were constructed. In particular, "blobby sweep" primitives (Section 5.4.1) have proven

to be an incredibly powerful modeling tool.

ShapeShop has evolved into primarily a sketch-based BlobTree modeling tool, with elements of the CAD-style interface still used where sketch-based equivalents have not been developed or implemented. The full CAD-style interface is still available, however the current system is designed to be used for sketch-based model construction. In the following sections, the non-standard elements of the ShapeShop modeling system will be described, followed by details on the internal architecture of the BlobTree implementation.

## 6.3 ShapeShop Interface

A screenshot of the ShapeShop user interface is shown in Figure 6.1. The system is implemented on Microsoft Windows using the MFC C++ application framework. The main interface window is the single-pane model view. Two additional windows are used to manipulate the scene - a tree view for interacting with the current BlobTree hierarchy, and a list view for changing parameters of a single tree node. Standard menus and toolbars are also used to control functionality which has not yet been implemented in the sketch-based interface.

Various sketch-based interface components are embedded in a Heads-Up Display (HUD) rendered over top of the model view. The Expectation List dynamically responds to sketches drawn by the user, offering possible model interactions. The Parameter Toolbar offers interactive manipulation of the most commonly-used parameters of the selected node. Similarly, the Options Toolbar provides access to frequently changed scene parameters, and the View Toolbar provides interactive camera control.

## 6.4 Sketch-Based Modeling Operations

In contrast to traditional 3D modeling interfaces, which are generally based on manipulation of control points and abstract parameters, *Sketch-Based Modeling* systems attempt to explicitly infer 3D shape from hand-drawn strokes. A variety of approaches have been taken. Systems such as SKETCH [126] and GiDES++ [63] generate polyhedral objects based on simple gestures. In the Teddy [60, 59] system, arbitrary strokes were used to directly specify free-form modeling operations. A variety of recent works [65, 13, 78, 112, 36, 10] have explored this style of interface. Most of these systems have been based on volumetric implicit surface shape representations, however none of these systems take advantage of hierarchical implicit modeling.

ShapeShop largely follows the free-form sketch-based modeling interface style introduced by [60]. The primary shape-creation operation in these interfaces is *inflation*, where a closed 2D contour is inflated into a "blobby" 3D shape. This operation can be easily accomplished using implicit sweeps with the "blobby" endcap style, as described in Section 5.4.1. The 2D sketch (Figure 6.2a) is projected onto a plane through the origin parallel to the current view plane, and then inflated in both directions (Figure 6.2b). After creation, the width of the primitive can be manipulated interactively with a slider (Figure 6.2c). The inflation width is functionally defined and could be manipulated to provide a larger difference between thick and thin sections. One advantage of an implicit representation is that holes and disjoint pieces can be handled

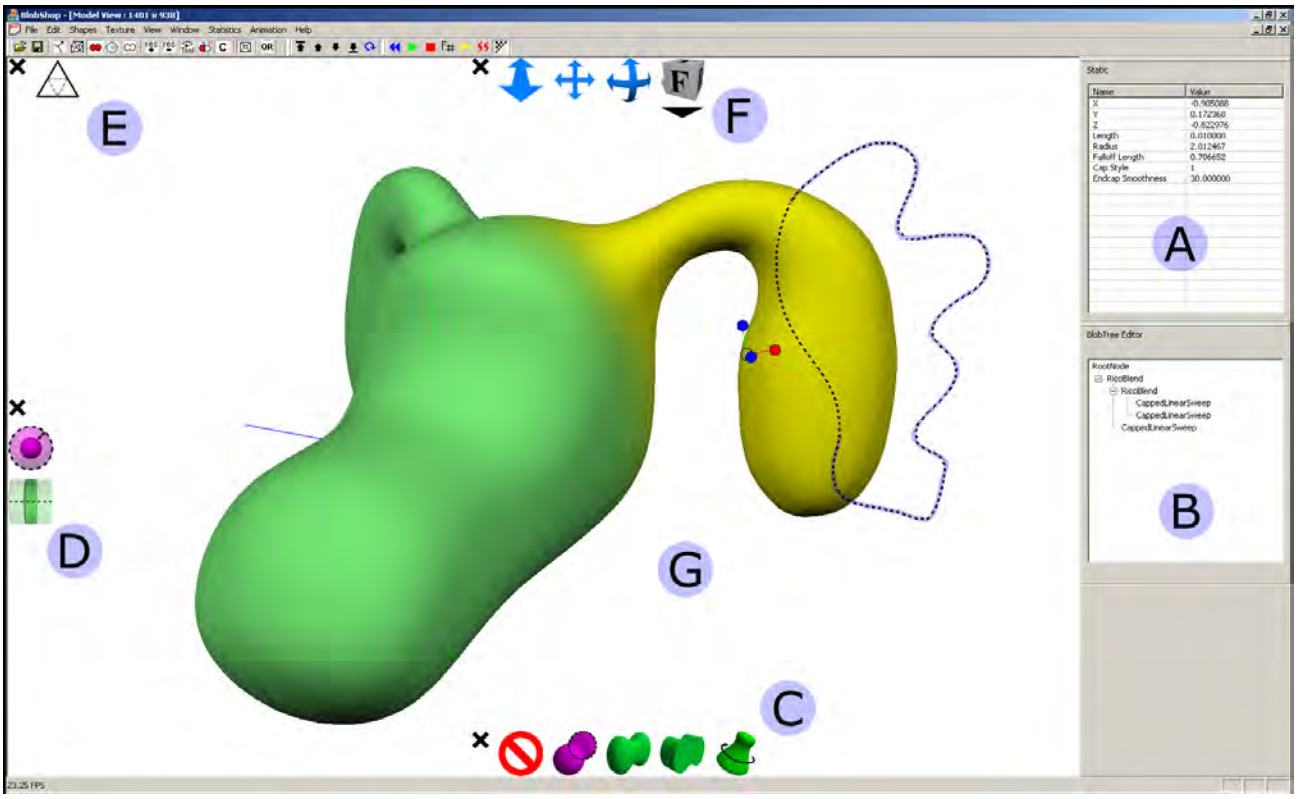



Figure 6.1: *ShapeShop user interface screenshot. The various major interface elements include the (A) Parameter Editor, (B) BlobTree Editor, (C) Expectation List, (D) Parameter Toolbar, (E) Options Toolbar, (F) View Toolbar, and (G) Model View.*

transparently.

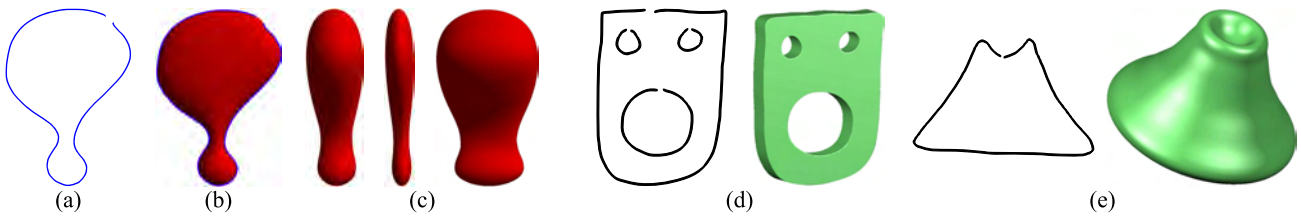



Figure 6.2: *Blobby inflation converts the 2D sketch shown in (a) into the 3D surface (b) such that the 2D sketch lies on the 3D silhouette. The width of the inflated surface can be manipulated interactively, shown in (c). Sketched 2D curves can also be used to create (d) linear sweeps and (e) surfaces of revolution.*

The sweep surface representation underlying this blobby inflation scheme also supports linear sweeps (Figure 6.2d) and surfaces of revolution (Figure 6.2e). Linear sweeps are created in the same way as blobby shapes, with the sweep axis perpendicular to the view-parallel plane. The initial length of the sweep is proportional to the screen area covered by the bounding box of the 2D curve, but can be interactively manipulated with a slider. Surfaces of revolution are created by revolving the sketch around an axis lying in the view-parallel plane. Revolutions with both spherical and toroidal topology can be created.

Existing sketch-based systems have generally not included these additional types of shapes[1] While blobby inflation is highly useful for many modeling tasks, linear sweeps and revolutions are invaluable in situations such as mechanical modeling. In particular, surfaces of revolution are a class of shape that cannot be approximated with blobby inflation.

Since the underlying shape representation is a true volume model, cutting operations can be easily implemented using CSG operators. Users can either cut a hole through the object or remove volume by cutting across the object silhouette. Once a hole is created the user may transform the hole interactively. An interactive control is also available to modify the depth of cutting operations. Cut regions are represented internally as linear sweeps, no additional implementation is necessary to support cutting in the BlobTree. An example is shown in Figure 6.3. This CSG-based cutting operation is both more precise and less restrictive than in existing systems.

To create more complex free-form shapes, the user can blend new blobby primitives to the current volume via oversketching. To position the new blobby primitive, rays through the sketch vertices are intersected with the current implicit volume. The new primitive is centered at the average z-depth of the intersection points. The width of the new blobby primitive can be manipulated with a slider, as can the amount of blending. Blended volumes can be transformed interactively, an example is shown in Figure 6.4. Karpenko et al [65] supported sketching of the blend profile but also noted that this technique was not robust and is very slow to compute. The level of control over the blend surface provided in ShapeShop has not been available in previous sketching systems.

Any BlobTree primitive can be used to add surface detail based on sketches. As an initial experiment, ShapeShop supports "surface-drawing". Rays through the 2D sketch are intersected

[1] A notable exception is the more advanced revolution technique of Cherlin et al [36]

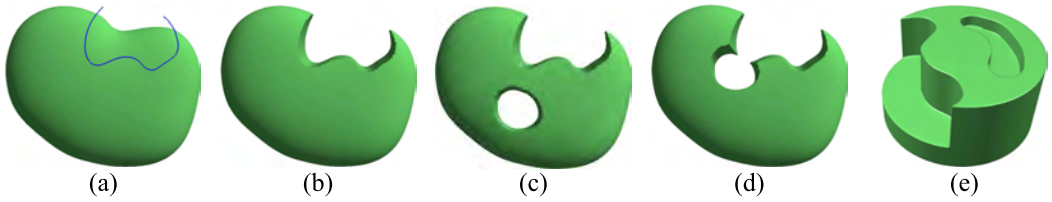



Figure 6.3: *Cutting can be performed (b) across the object silhoutte or (c) through the object interior. Holes can be interactively translated and rotated. Intersection with other holes is automatically handled, as shown in (d). Hole depth can also be interactively modified to create cut-out regions (e).*

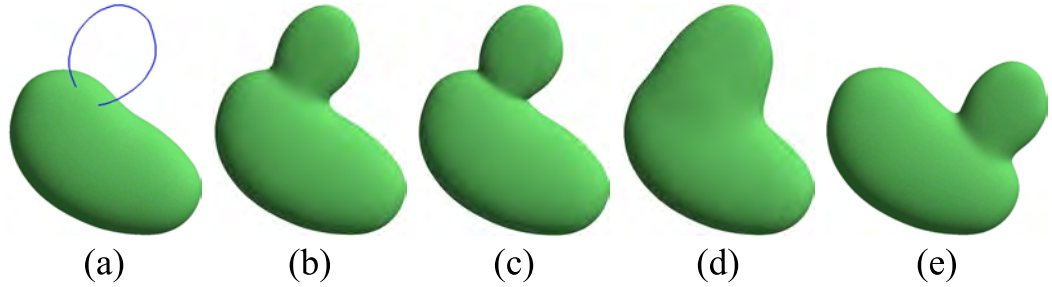



Figure 6.4: *The sketch-based blending operation (a) creates a new blobby inflation primitive (b) and blends it to the current volume. The blending strength is parameterized and can be interactively manipulated, the extreme settings are shown in (c) and (d). The blend region is recomputed automatically when the blended primitives move, as shown in (e).*

with the current implicit volume, and a 3D implicit polyline primitive is generated. Interactive controls are provided to manipulate both the surface radius and linear scaling (tapering) along the polyline. Results are shown in Figure 6.5.

Surface drawing with implicit volumes is a very flexible technique. Any pair of implicit primitive and composition operator can be used as a type of "brush" to add detail to the current surface. For example, creases could be created by subtracting swept cone primitives using CSG operations. Implementing these alternative tools within the BlobTree framework is very straightforward. In addition, since each surface-drawing stroke is represented independently in the model hierarchy, individual surface details can be modifed or removed using the existing modeling interface.

By surface-drawing on temporary construction surfaces, it is possible to create non-planar 3D geometry that would otherwise be impossible to sketch (Figure 6.5d,e). These temporary construction surfaces are a novel property of ShapeShop which was not designed into the system, but simply emerged out of the non-linear hierarchical editing capabilities of the BlobTree.

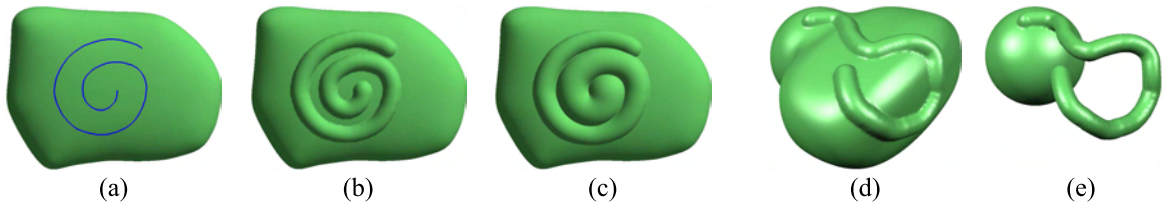



Figure 6.5: *Surface-drawing is specified by a 2D sketch, as shown in (a). Blended skeletal implicit point primitives are placed along the line at intersection points with the model, shown in (b). In (c) the radius of the points is increased and then tapered along the length of the 2D curve. Temporary construction surfaces (d) can be used to create more complex 3D curves (e).*

## 6.5 Node Selection and BlobTree Traversal

Procedurally-defined BlobTree volumes inherently support non-linear editing of internal tree nodes. However, before a primitive can be manipulated it must be selected. One option is to cast a ray into the set of primitives and select the first-hit primitive. This technique is problematic when dealing with blending surfaces, since the user may click on the visible surface but no underlying primitive is hit.

Picking in ShapeShop is implemented by intersecting a ray with the current volume, and then selecting the primitive which contributes most to the total field value at the intersection point. This algorithm selects the largest contributor in blending situations, and selects the subtracted primitive when the user clicks on the inside of a hole surface created by CSG difference operations. However, selecting the largest field-value contributor can result in un-intuitive behavior in cases where a small primitive is blended with a larger one, as the larger primitive may contribute more to the field at all points on the surface. In this case, the user must use the BlobTree Editor tree view (Figure 6.1) to select the desired node.

This selection system only allows for selection of primitives. To move up the tree to parent nodes, a *parent* gesture is implemented which selects the parent of the current node. The *parent* gesture is entered as a straight line towards the top of the screen. No similar child gesture has been implemented because it is unclear how to disambiguate which child is desired in cases where a node has multiple children. More complex tree traversal and manipulation, such as re-arranging nodes, currently requires the use of the BlobTree Editor tree view.

A selected primitive or composition node can be removed using a simple *erase* gesture, entered as a small "scribble" over top of the selected primitive. Removing a compostion node is equivalent to cutting a branch from the model tree - all children are also removed.

## 6.6 BlobTree Visualization

To interactively visualize BlobTree models in ShapeShop, real-time polygonization of the surface is performed. The system supports a variety of algorithms, the most basic being an optimized version of Bloomenthal's polygonizer [26]. Two modified versions of this polygonizer are also available. The first simply adds the crease-finding techniques described in the Extended Marching

Cubes work [66]. The second provides support for local updates, where re-meshing is limited to the regions in which the current model has changed.

Note that supporting partial re-meshing introduces significant complexity - the polygonizer must maintain knowledge about which triangles and vertices are associated with each polygonization cube, and the mesh must support efficient partial decimation. The overhead from these operations is significant, in particular when re-meshing the entire surface the local update polygonizer takes approximately twice as long as the standard polygonizer. However, since smaller local updates are so much more efficient, the benefits largely outweigh this particular cost.

All the polyognization algorithms used in ShapeShop are descendants of the method introduced by Wyvill et al [124] which is commonly referred to as *marching cubes* [67]. These algorithms are known to produce meshes with many near-degenerate triangles. To improve mesh quality, Ohtake's mesh refinement algorithm [76] has been implemented. The user can choose to always apply this algorithm as a meshing post-process, however it is expensive and can greatly reduce interactivity. As an alternative, a "Quick Refine" button is available which runs 5 iterations of mesh relaxation. This significantly improves triangle quality and only takes a few seconds for moderately complex models.

When a primitive or sub-tree of the model hiearchy is selected, it is often non-obvious what the shape of the selected internal volume is (particularly in blending situations). To assist in comprehension, several rendering modes have been implemented to display the selected volume (Figure 6.6). Another option is to visualize the influence region of a volume by coloring the mesh based on the field value of the selection.

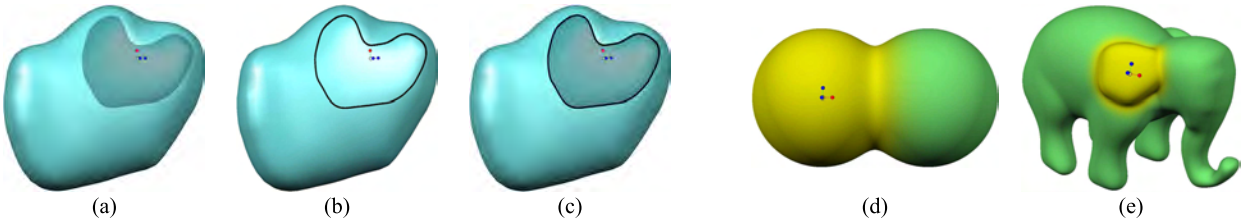



Figure 6.6: *Internal volumes can be directly rendered using (a) transparency, (b) silhouette lines, or (c) transparency and silhouttes. Mesh highlighting can also be used to convey the influence region of a selection, as in (d) and (e).*

## 6.7 BlobTree Implementation Details

The BlobTree [120] implementation used in ShapeShop is relatively non-traditional, compared to recent implementations such as that described by [51]. First, only portions of the BlobTree relating specifically to shape modeling have been implemented. Additional functionality, such as color, texture, and animation have been avoided to simplify the implementation and reduce computational costs. Some of these exclusions have been driven by the use of Hierarchical Spatial Caching (Chapter 4) - for example, adding floating-point RGBA color caches would quadruple the memory requirements (and update time) for each cache.

Another significant difference is in the division of BlobTree functionality. ShapeShop uses a 3-tiered object-oriented BlobTree architecture. The lowest tier is a library of basic implicit surface objects, including various primitives and composition functions. All objects implement a generic *ScalarField* interface which defines basic functions for computing the field value, gradient, bounding box, and seed points. None of the objects at this level have any notion of a hierarchical organization.

The second tier represents the standard BlobTree hierarchical modeling system. A variety of primitive and operator nodes are implemented, some which simply wrap objects from the implicit surface library, and others which are an aggregation of multiple implicit surface objects. All composition nodes at this level have an associated spatial cache, following the *tightly-coupled* caching implementation style described in Section 4.5. Similarly, all nodes support a basic event-passing mechanism for notifying parent nodes about invalidation events.

The third tier can be considered the user-interface view of a BlobTree. The nodes at this level largely wrap nodes at the BlobTree level, adding functionality for supporting the various user-interface tools and parameter editors. However, nodes at this level can also represent an aggregation of multiple BlobTree nodes, providing a conceptual overview of an otherwise much more complicated underlying structure.

While this three-tiered architecture may seem over-engineered, the benefits have already been hinted at above. By separating implicit surface functionality from the BlobTree nodes, more complex BlobTree nodes can easily be assembled. Since all implicit surface objects support the same basic interface, alternate implementations can be quickly swapped in and out without significant changes to the BlobTree code. Similarly, changes to the user-interface do not require modification of the BlobTree or implicit surface packages. The ability to present a "conceptual" BlobTree to the user which does not actually map one-to-one to the actual BlobTree nodes is also desirable from a user-interface perspective. For example, the model tree view can be simplified by not explicitly representing transformation nodes, reducing apparent tree complexity and depth.

The notion of a "conceptual" BlobTree also provides an avenue for transparently supporting dynamic BlobTree optimization. A variety of techniques for dynamically re-organizing a BlobTree to provide more efficient evaluation have been developed [15]). However, it is not appropriate to present this optimized tree directly to the user. In most cases, the model tree will have been carefully arranged to preserve certain abstract hierarchical divisions. While perhaps less efficient, the user-defined organization must be preserved. The user-interface tree can easily provide this view, while the underlying BlobTree can be freely optimized. While these techniques have yet to be implemented in ShapeShop, they will not require any significant architectural changes to support.

## 6.8 Limitations and Future Work

ShapeShop is, at this time, only a proof-of-concept system. Interactive hierarchical implicit modeling is a relatively unexplored area, largely because real-time interaction was very limited prior to the development of hierarchical spatial caching. Experience with the ShapeShop interface has raised a variety of issues, some well-known, and others less obvious.

A key limitation exists in ShapeShop regarding visualization of sharp creases and other high-frequency features on complex models. Adaptive polygonization schemes [53, 5, 14] may significantly improve visualization of small features. However, as shown in Section 4.8.4, hierarchical spatial caching interferes with existing techniques for improving crease reconstruction. Adaptive polygonization will not improve this situation, new techniques must be developed.

In the user interface, one of the biggest challenges is interacting with the BlobTree hierarchy. The *parent* gesture described in Section 6.5 only allows the user to traverse the tree in an upward direction. Download traversal, or selection of non-leaf nodes, must occur through the BlobTree Editor tree view 6.1. However, for deep or large trees, this tree view quickly becomes extremely difficult and time-consuming to use. A better solution for tree traversal is needed, preferably one which can be applied directly in the model view, without the aid of a separate tree view. A further problem is how to support tree manipulation directly in the model view.

Closely related to tree traversal is the problem of visualizing internal nodes and sub-trees of the BlobTree model. The techniques for mesh highlighting and internal visualization described in Section 6.6 are ad-hoc methods that were simply easy to implement. User studies are necessary to determine which techniques are most effective at conveying both node shapes and the BlobTree structure.

A final problem, which concerns any functional implicit modeling system, is that of reproducibility. Many of the primitives and operators used in ShapeShop are highly complex and cannot be described with a simple functional definition. For example, the variational-curve-based sweep primitives of Chapter 5 rely not only on many parameters, but on a variety of other algorithms. Unless these algorithms are duplicated exactly, the resulting scalar fields will be different. Hence, transferring models from ShapeShop to future implicit modeling systems (or even future versions of ShapeShop) may not produce the same results. There is no easy solution to this problem. One approach may be to re-write the implicit surface library using some virtual machine language, and package the implicit library with each saved file. A similar method is used by the HyperFun [4] system, although the use of interpreted code there is unlikely to provide interactive performance.

## 6.9 Chapter Summary

Interactive tools for hierarchical implicit modeling systems such as the BlobTree have traditionally been very limited. Systems for real-time manipulation of skeletal primitives have been developed, but interactive visual feedback on the changing implicit surface did not scale beyond trivial models. The ShapeShop system presented in this chapter is essentially the first system to provide scalable interactive visualization of complex BlobTree models as they are modified. ShapeShop has a novel sketch-based interface which supports fast and intuitive construction of complex models. A variety of other technical issues relating to interactive BlobTree editing have also been discussed, such as direct selection of primitives and dynamic highlighting of selected regions. Various issues relating to the implementation of BlobTree data structures used in ShapeShop were covered, and directions for future work were suggested.

# Chapter 7

# Results and Conclusion

*A task-based analysis of the ShapeShop interactive BlobTree modeling system is performed by considering a series of possible applications. Directions for future work are considered, followed by a summary of the thesis.*

*Some images in this chapter are taken from the publications "Interactive Implicit Modeling with Hierarchical Spatial Caching" by Schmidt, Wyvill, and Galin[97], "Generalized Sweep Templates for Implicit Modeling" by Schmidt and Wyvill [96], "Sketch-Based Modeling with the BlobTree" by Schmidt, Wyvill, and Sousa[98], "ShapeShop: Sketch-Based Solid Modeling with BlobTrees" by Schmidt, Wyvill, Sousa, and Jorge [99], "Sketch Based Construction and Rendering of Implicit Models" by Wyvill, Foster, Jepp, Schmidt, Sousa, and Jorge [119], "Interactive Decal Compositing with Discrete Exponential Maps" by Schmidt, Grimm, and Wyvill [94], and "Interactive Pen-and-Ink Rendering for Implicit Surfaces" by Schmidt, Isenberg, and Wyvill [95]. The images appearing here are due to Schmidt.*

## 7.1 Results

The goal of this thesis, as stated in section 1.4, is to provide a framework in which interfaces for interactive hierarchical implicit modeling can be explored. In some sense, determining whether or not this goal has been met is trivial. An implicit modeling system has been created. ShapeShop enables interactive BlobTree modeling in a way that was not previously possible. The technical merits of Hierachical Spatial Caching and implicit sweep surfaces were demonstrated in their respective Chapters (4 and 5).

However, these technical evaluations have only moderate relevance to the actual interactive experience. The important question is whether or not implicit modeling in ShapeShop is useful. Of course, as research software, ShapeShop cannot be reasonably compared to well-established commercial products. Standard interface evaluation methods are also difficult to apply. ShapeShop's underlying implicit shape representation is radically different from traditional modeling interfaces, as is the sketch-based interface. Under these conditions, isolating relevant factors which can be subjected to HCI evaluation techniques provides little information about the system as a whole. Furthermore, many aspects of the user interface have been given only limited attention, so test subjects are likely to encounter problems which are completely unrelated to implicit modeling.

How, then, can ShapeShop be evaluated? The approach taken in this chapter is to demonstrate the effectiveness of ShapeShop at supporting a variety of modeling tasks. Unfortunately the format of this thesis has limited capability for demonstrating interactive tasks. As a consolation, several 3D models have been produced for each task. In some cases, the models are quick examples, while others are the result of many hours of effort and test the limits of ShapeShop.

The intent is to convey some sense of what ShapeShop is good at, and the potential of interactive hierarchical implicit modeling.

### 7.1.1 Interactive Assembly

One very basic task in shape modeling is assembling a complex model from a set of simpler components. Compared to implicit surfaces, B-Rep modeling systems have limited support for this task. Part assembly is common in CAD systems, however generally the parts are simply positioned, rather than combined. Mesh and spline modeling tools rarely provide any facility for composing complex shapes. However, this task is generally not interactive for implicit models - times are often reported in minutes for complex models [11].

As discussed in Section 4.8, ShapeShop supports interactive assembly of relatively complex implicit models. The timings there showed that blended components of the Medusa model, made from 9490 skeletal point primitives segmented into 7 parts, could be manipulated at several frames per second. Various parts of the Medusa model have also been combined with other complex implicit models, as shown in Figure 7.1. Note that the mesh resolutions shown in these figures are significantly higher than those used during interaction. However, interactive mesh resolutions were sufficient to carry out position fine-tuning without extensive checking at higher resolutions.


Figure 7.1: *Interactive assembly of complex implicit models. An initial design study of combining the Medusa model of Chapter 4 with an implicit cup (left), and the final result rendered with a volume texture (center). Letters were added to the tail using surface drawing (Section 6.4). The Medusa head was also combined with an implicit cup (right).*

Another model created from interactive assembly is shown in Figure 7.2. In this case, the individual components (arm, leg, body, head, and wing) were created in ShapeShop, using the sketch-based interface. These parts were then imported and assembled into a final model containing 65 linear sweep primitives and 35 composition operators.

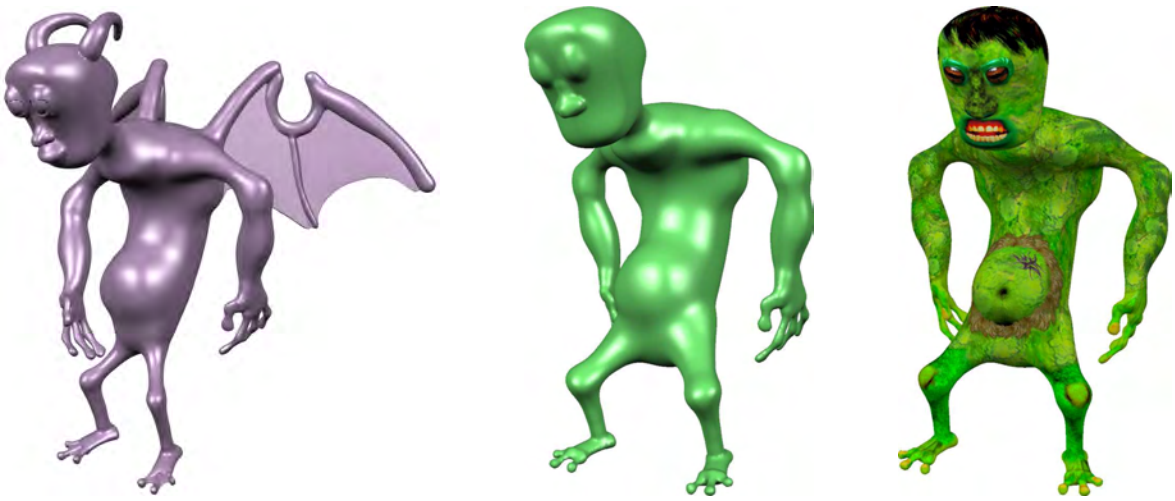


Figure 7.2: *A complex implicit Gremlin model (left). A simplified version (middle) has been textured using a decaling technique [94] (right).*

### 7.1.2 Character Modeling

While this research has focused on the shape modeling properties of hierarchical implicit surfaces, another key benefit is that the model hierarchy can be trivially animated [120]. Combined with smooth blending, BlobTree models are a powerful tool for representing both the shape and motion of animated characters.

Some character models created with an early version of ShapeShop are shown in Figure 7.3. These models are notable because although the interactive tools available at the time were very limited, each of these models was generated in a single session lasting under an hour (a necessity, since the software did not then support saving or loading). The models shown are also the initial attempts at each character. ShapeShop easily supports this style of free-form, experimental modeling, and generally produces meaningful results.

Likewise, the skeleton model in Figure 7.4 was sketched in a single session, this time on the second try (the system crashed half-way through the first trial). The model is fully articulated and could easily be animated. A notable feature of this model is the skull, which was initially created as a solid volume. Later in the session, the interior was removed using CSG subtraction operations, and exposed by cutting out the eyes and nose. These changes took only a few minutes. Performing similar modifications on most B-Rep models would require significantly more user effort.

A common workflow for character modeling is to first create a clay sculpture of the character as a design study, and use it as a basis for the 3D computer model. An attempt was made to mimic this process using the clay statue in Figure 7.5. Note that this model would be difficult to laser scan, due to the complex topology. In terms of exact duplication, the test was clearly not a success. However, very little time was spent refining the model. The goal was to create a quick approximation that could be used as a basis for texture mapping. ShapeShop met this objective; the modeling time was under half an hour. One of the reasons clay modeling is still

Figure 7.3: *Early examples of character models sketched using ShapeShop. The rat (left) and scorpion (right) were created entirely using sketched blobby primitives. The elephant (center) is a combination of standard implicit primitives, sketched primitives, and a bezier sweep for the nose.*

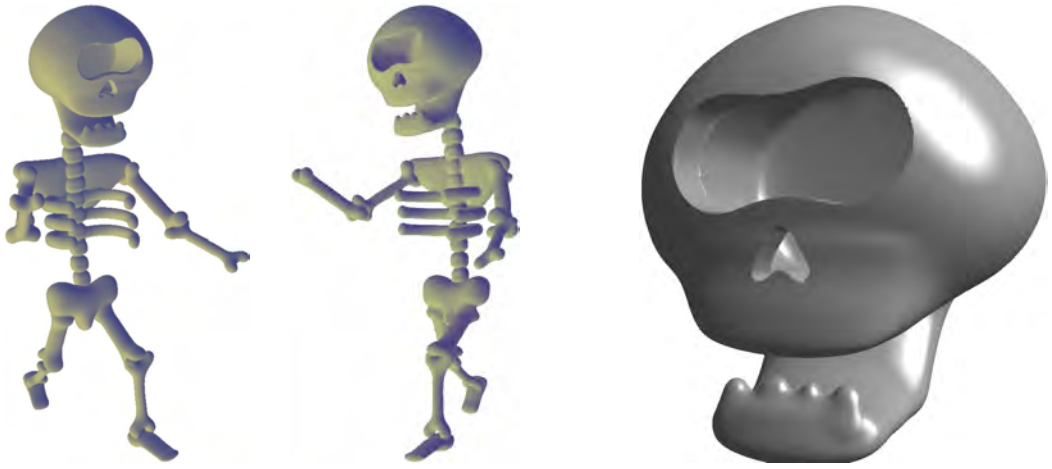


Figure 7.4: *A skeleton model sketched in ShapeShop. The head of the skull has been hollowed out using CSG operations, and the bones are articulated such that they can be animated directly.*

performed is that clay is more malleable than the computer representation and easily supports global modification. Implicit modeling, which easily supports large-scale manipulation of blended surfaces, may have some impact on this practice.

Figure 7.5: *An elephant model (middle) sketched based on an existing physical clay elephant statue (left). Sketching this model took under half an hour. Additional surface detail was added using texture mapping (right).*

Finally, a re-posing task similar to that which would be carried out in character animation is shown in Figure 7.6. ShapeShop's interface does not fully support hierarchical re-posing, so the modifications shown are rather slight. However, these two poses could easily be interpolated to create a short animation.

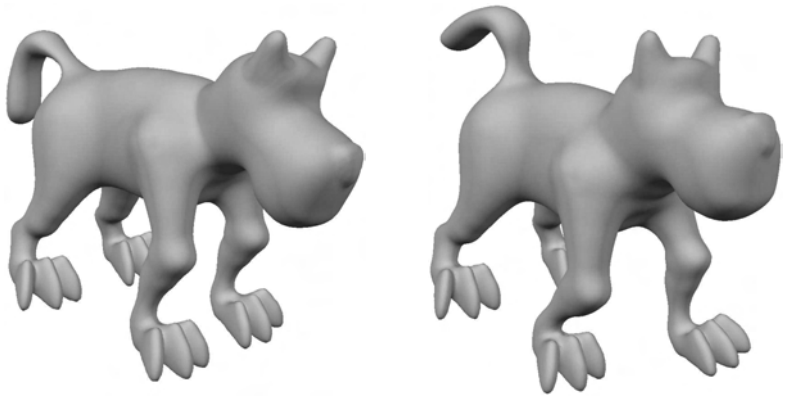


Figure 7.6: *A dog model sketched using ShapeShop. The hierarchical organization of the model corresponds to the anatomical organization, allowing the figure to be easily re-posed.*

### 7.1.3 Mechanical Parts

Computer-Aided Design (CAD) is perhaps the most common application of 3D modeling. The use of computer modeling in product engineering has become almost ubiquitous. Solid models are used in design, manufacturing, and engineering analysis. The guaranteed validity and analytic properties of implicit surfaces have clear benefits here. A variety of basic models of physical

objects are shown in Figure 7.7. The piston model in that figure demonstrates a one of the novel capablities of ShapeShop. This model has multiple interior holes but was sketched in under 10 minutes (and done with a mouse, instead of a tablet). The goal here was not to create a part ready for production, but rather to demonstrate that ShapeShop can be used as an effective tool to convey an idea. To work at this speed, product designers are generally limited to 2D sketching, which is very flexible but has limits when used for representing 3D shapes.

Figure 7.7: *Mechanical objects. The piston on the left was sketched in under 10 minutes, despite it's complex topology.*

Another engineering model is shown in Figure 7.8. This car model was initially constructed in much the same way as the physical clay prototypes used by the automotive industry - as a solid block. The interior was then hollowed out, and seats and windows were added. The ease with which this process was carried out suggests an alternate workflow for car modeling. Clay body prototypes are already routinely laser scanned. The results could easily be converted to implicit form using any of the techniques mentioned in Section 3.5, and the interior interactively modeled using ShapeShop. It has already been noted that composing implicit surfaces is trivial, so part libraries could be easily integrated.

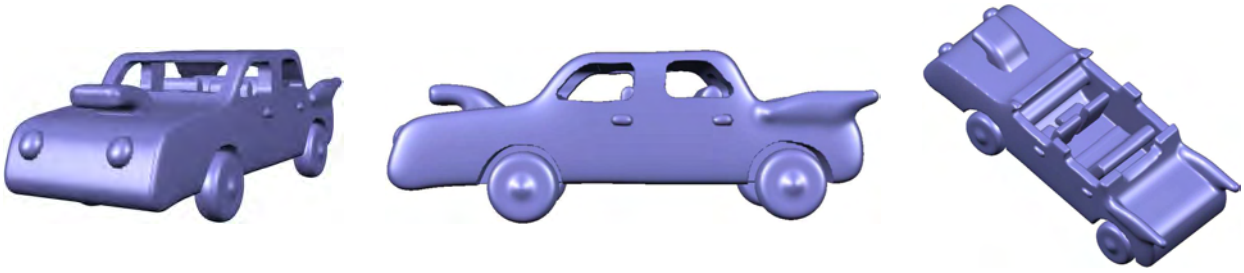


Figure 7.8: *Car model sketched using ShapeShop. The interior structure was added after the initial car body was designed. A cut-away view is shown on the right.*

A third model is shown in Figure 7.9. Taking a more traditional CAD approach, this model is created entirely out of sweep primitives. However, implicit blending is used to create water-tight smooth joints between surfaces. Since the model is actually an implicit volume, it can be used directly to generate surface and volume meshes suitable for use in physical simulations.

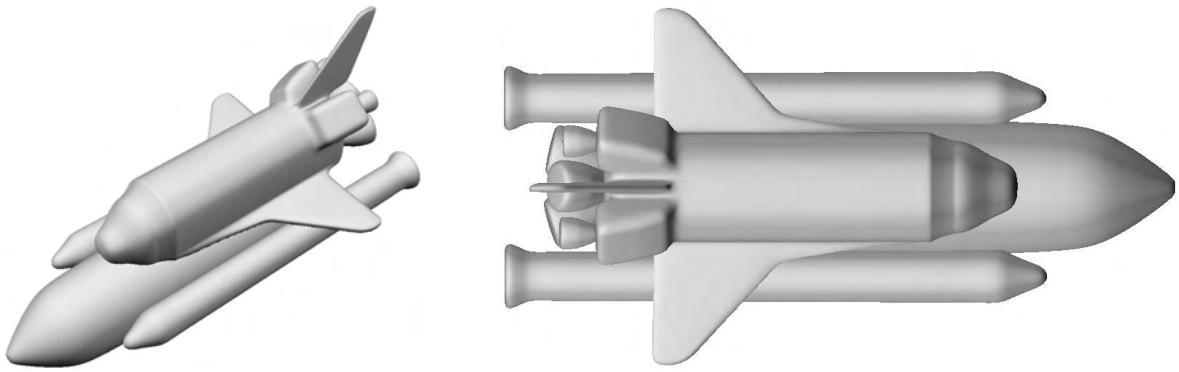


Figure 7.9: *Shuttle model created using implicit sweep primitives. Sweep primitives can be blended with arbitrary levels of smoothness.*

### 7.1.4 Biological Modeling

One of the widely-recognized benefits of implicit surfaces is their ability to represent the smooth, amorphous surfaces often found in nature. Recent interest in surgical simulators and large-scale simulation of human biology have increased the need for physiological models. The possibility for interactive manipulation of shapes reconstructed from CT data was already examined in Section 5.8. Another option is shown in Figure 7.10, where a 3D model of a human heart has been sketched. Modeling time was approximately 30 minutes. The heart is not biologically accurate, but clearly demonstrates the ability of ShapeShop to represent the smooth, complex branching structures found in nature. Smaller features such as veins, which may be difficult to reconstruct from scan data, can be easily added. The use of such quickly-sketched biological models for use in medical training has been previously examined [14].

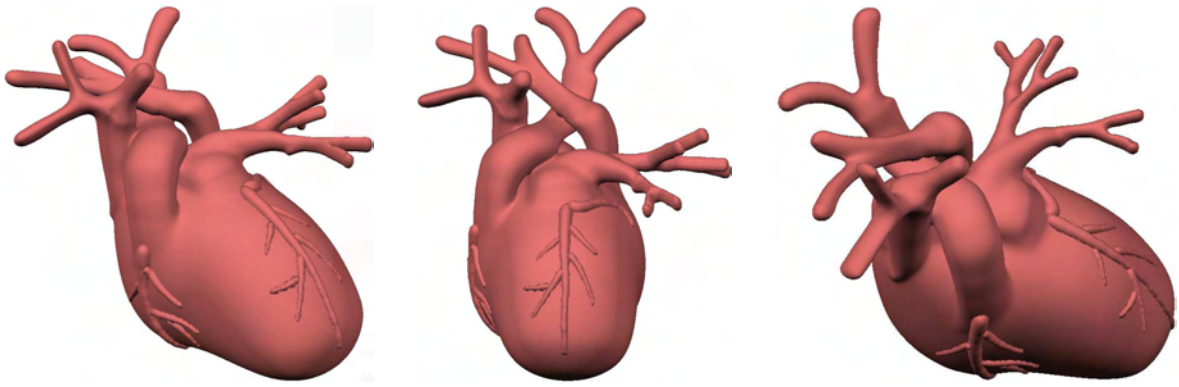


Figure 7.10: *Heart model sketched using ShapeShop in approximately 30 minutes. Complex interlocking branching structures can easily be created by concatenating smaller segments.*

### 7.1.5 Design Iteration

A significant portion of most design cycles is iteration. Computer modeling easis this process, as computer models are generally cheaper than their physical counterparts. However, with B-Rep systems it is still often necessary to "start from scratch" if significant changes are necessary. The non-linear editing capabilities of the BlobTree can minimize the need to re-start, or at least allow satisfactory portions of the previous iteration to be easily cannibalized. Several design iterations were performed on the basic dog model shown in the left of Figure 7.11. A notable modification is the addition of the pig nose in the rightmost model. The pig nose was simply sketched over top of the existing dog nose, and blended with the model. In a B-Rep system, this would likely have produced an invalid model. Blending composition in the BlobTree simply combines the underlying functions, resulting in a well-defined volume.

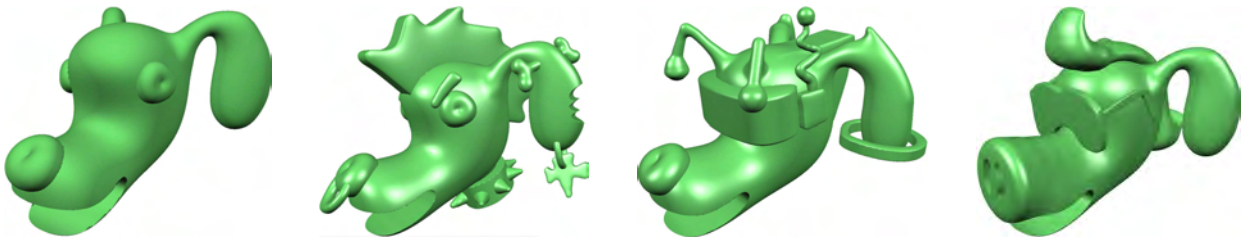


Figure 7.11: *Design iterations on a basic dog model (left). The implicit model can be easily modified by deleting components and sketching new ones. The pig nose (right) was simply blended onto the existing model.*

### 7.1.6 Observations

The models shown in the previous sections clearly demonstrate both the benefits and limitations of ShapeShop. In it's current form, ShapeShop is usable primarily as a rapid prototyping tool for 3D models. This is in some part decided a priori by the sketch-based interface, which emphasizes speed over precision. However, various limitations in both the interface and the current implementation prevent ShapeShop from scaling to very complex models. To give some sense of the limits of ShapeShop, the full Gremlin model in Figure 7.2 can only be manipulated interactively if the mesh resolution is very low, too low to be really considered usable. However, initial experiments with adaptive polygonization indicate that this is only a temporary limitation.

Whether for use as an exploratory design environment, teaching aid, or virtual back-of-a-napkin, ShapeShop provides a compelling 3D modeling experience. As the 3D doodle below demonstrates, ShapeShop enables a style of 3D modeling that is non-existent in current B-Rep-based tools.

## 7.2 Future Work

The ideas presented in this thesis suggest various directions for future work. In terms of the theory of implicit surfaces, the classification produced in Chapter 3 should be analyzed further,

Figure 7.12: *A 3D Doodle*

and likely refined. There is a bias towards solid modeling in the properties selected to organize and compare the different techniques. An application-independent approach may be beneficial. Further analysis of normalization properties would be particularly useful, as normalization error has a significant impact on many algorithms.

Hierarchical Spatial Caching has proved to be quite successful at increasing visualization speed in ShapeShop. However, a variety of limitations exist in the current implementation. The loss of sharp features is a significant drawback which should be addressed. Separable operators (Section 4.5) may produce significant speed-ups, and should be explored, as should automated cache management algorithms based on more detailed scene analysis. Another significant improvement would be the development of adaptive-sampling methods that preserve higher-order field continuity.

The implicit sweep surfaces developed in Chapter 5 can be improved in a variety of ways. Boundary constraints in the sweep template are currently placed in somewhat of an ad-hoc method. Level-set interfaces propagating at curvature-dependent speed could be used to generate more controllable boundary constraints. The 2D distance field approximation technique could also be extended to 3D, although the computational cost of solving the dense variational system will be challenging to overcome. In terms of the 3D sweeps, one limitation is that numerical root-finding is usually necessary in the case of curved trajectories. Root finding is expensive and requires an analytic expression for the curve tangent to function robustly. A key avenue for future work is a more general implicit sweep formulation that does not rely on root-finding. A constructive approach based on approximation by union of linear segments can be employed, however this technique breaks down in regions with high curvature. Approximation by union of simpler curved segments could provide superior results.

Finally, ShapeShop enables a wide range of possible future work on interactive BlobTree modeling. Comparable systems simply do not exist, so research on human factors in interactive implicit modeling is essentially an open field. The implementation simplicity of implicit solid

modeling makes development of new interaction techniques very straightforward, however care must be taken to extract what is useful from what is possible. Of particular use would be free-form local and global deformation tools. A variety of other current problems were discussed in Section 6.8. One particularly challenging issue which bears repeating is the reproducibility problem, which must be addressed if implicit modeling is to enter the mainstream.

## 7.3 Conclusion

In Chapter 1, two key limitations that essentially prevent interactive BlobTree modeling were identified - interactive visualization and direct surface manipulation. The stated goal of this thesis was to address these issues, and produce a framework which could be used to explore interactive tools for hierarchical implicit modeling. Several contributions have been made in pursuit of this goal.

First, chapters 2 and 3 provided a summary and comparison of current techniques for shape modeling with implicit surfaces. The novel classification developed there was used to justify the choice of the BlobTree modeling framework as a suitable basis for an interactive system.

The interactive visualization problem was addressed in Chapter 4, with the introduction of Hierarchical Spatial Caching. This novel technique was shown to provide an order-of-magnitude improvement in visualization times for relatively complex hierarchical implicit models. This speed-up is sufficient to enable real-time visual feedback of a changing implicit surface as the underlying hierarchical model is manipulated. Finally, an extensive evaluation of Adaptive Distance Fields was carried out, resulting in the identification of several issues which prevent their use for spatial approximation of smooth scalar fields.

Implicit sweep surfaces which support direct manipulation of the sweep profile were introduced in Chapter 5. To generate these sweep volumes, a novel approach to $C^2$-smooth 2D distance field approximation was developed. A technique for preserving creases in the sweep template was presented, and three different sweep endcap variations were described. These sweep surfaces are compatible with the BlobTree, providing a powerful and intuitive new tool for quickly modeling more complex shapes.

A proof-of-concept interactive BlobTree modeling environment was described in Chapter 6. This system, known as ShapeShop, includes both a traditional CAD-style interface and a novel sketch-based modeling input system. A variety of details on the implementation were discussed, and an informal evaluation of the system was related in Section 7.1.

Interactive visualization and direct surface control are long-standing problems in hierachical implicit modeling. Hierarchical spatial caching makes a considerable advance over existing techniques. The new sweep surfaces presented in this work give significantly more control over the implicit surface than is available in previous BlobTree modeling systems. Clearly, neither of these problems can be considered to be solved. However, no interactive BlobTree modeling system comparable to ShapeShop has been published to date, and ShapeShop could not have been developed without either of these two new techniques. Hence, the goals of this thesis have been met. Further, it is hoped that ShapeShop demonstrates the viability of hierarchical implicit modeling in interactive design, and will enable future research in this direction.

# Appendix A

# Linear and Quadratic Reconstruction Filters

A variety of reconstruction filters have been proposed for creating a continuous 3D signal based on a set of uniformly-spaced 3D sample poitns. A variety of techniques are surveyed by Marschner and Lobb [69]. While the tri-cubic and windowed sinc filters were identified as producing the smoothest reconstructions, these filters require a large number of samples and are expensive to evaluate. For example, the tri-cubic filter requires 64 field samples and evaluation of 21 cubic polynomials. These filters are impractical for interactive applications. The tri-linear and tri-cubic filters described in this appendix provide moderate smoothness at an acceptable cost.

## A.1 Tri-Linear Reconstruction Filter

The tri-linear reconstruction filter [69] is a separable, $C^0$ continuous filter. Evaluation of the filter at a point $\mathbf{p}$ requires 8 neighbouring samples. The filter is an interpolating filter, passing directly through the sample points. The three-dimensional filter is defined in terms of 7 applications of the one-dimensional linear reconstruction filter $\mathcal{R}_l$, which is simply a linear interpolation between the sample points $s_i$ and $s_{i+1}$

$$\mathcal{R}_l\left(s_i, s_{i+1}, t\right) = (1-t)s_i + t\ s_{i+1}$$

$\mathcal{R}_l$ reconstructs a signal over the interval $\mathcal{I} = [i, i+1]$. The parameter $t \in [0,1]$ is defined over $\mathcal{I}$.

The tri-linear filter $\mathcal{R}_{l^3}$ is defined in terms of $\mathcal{R}_l$. Evaluating $\mathcal{R}_{l^3}$ at a 3D sample point $\mathbf{p}$ requires 8 neighbouring voxels, with the nearest voxel being referred to as $v_{ijk}$. The one-dimensional parameter $t$ is computed along each grid axis, these are denoted $t_i$, $t_j$, and $t_k$. $\mathcal{R}_{l^3}(\mathbf{p})$ is calculated by repeated applications of $\mathcal{R}_l$:

$$\begin{aligned}
\mathcal{R}_i(\phi, \rho) &= (1-t_i)v_{i\phi\rho} + t_i\ v_{(i+1)\phi\rho} \\
\mathcal{R}_j(\rho) &= (1-t_j)\mathcal{R}_i(j, \rho) + t_j\ \mathcal{R}_i(j+1, \rho) \\
\mathcal{R}_{l^3}(\mathbf{p}) &= (1-t_k)\mathcal{R}_j(k) + t_k\ \mathcal{R}_j(k+1)
\end{aligned}$$

Here $\phi$ and $\rho$ are placeholders. $\mathcal{R}_i(\phi, \rho)$ is an evaluation of $\mathcal{R}_l$ in the $i$ direction, and $\mathcal{R}_j(\rho)$ is an evaluation of $\mathcal{R}_l$ in the $j$ direction.

## A.2 Tri-Quadratic Reconstruction Filter

The tri-quadratic reconstruction filter [18] is a separable, $C^1$ continuous filter. Evaluation of the filter at a point $\mathbf{p}$ requires 27 neighbouring samples. The filter is an approximating filter and hence does not necessarily pass through the sample points. The three-dimensional filter is

defined in terms of 13 applications of the one-dimensional quadratic reconstruction filter, which will be described first.

Evaluation of the 1D quadratic B-spline filter $\mathcal{R}_q$ requires 3 sample points $s_{i-1}$, $s_i$, and $s_{i+1}$ to reconstruct a signal over the interval $\mathcal{I} = [i - 0.5, i + 0.5]$. $\mathcal{R}_q$ is a function of one parameter $t \in [0, 1]$ defined over $\mathcal{I}$.

The 1D quadratic B-spline filter $\mathcal{R}_q$ is defined as

$$\mathcal{R}_q\left(s_{i-1}, s_i, s_{i+1}, t\right) = \left(\frac{s_{i-1} + s_{i+1}}{2} - s_i\right) t^2 + \left(s_i - s_{i-1}\right) t + \frac{s_{i-1} + s_i}{2}$$

The tri-quadratic filter $\mathcal{R}_{q^3}$ is defined in terms of $\mathcal{R}_q$. Evaluation of the filter at a 3D sample point requires 27 neighbouring voxels. For a 3D sample point $\mathbf{p}$, find the nearest voxel $v_{ijk}$. The one-dimensional parameter $t$ is computed along each grid axis, these are denoted $t_i$, $t_j$, and $t_k$. $\mathcal{R}_{q^3}(\mathbf{p})$ is calculated by repeated applications of $\mathcal{R}_q$, defined in the the following set of equations.

$$\begin{aligned}
\mathcal{R}_i(\phi, \rho) &= \mathcal{R}_q\left(v_{(i-1)\phi\rho}, v_{i\phi\rho}, v_{(i+1)\phi\rho}, t_i\right) \\
\mathcal{R}_j(\rho) &= \mathcal{R}_q\left(\mathcal{R}_i(j-1, \rho), \mathcal{R}_i(j, \rho), \mathcal{R}_i(j+1, \rho), t_j\right) \\
\mathcal{R}_{q^3}(\mathbf{p}) &= \mathcal{R}_q\left(\mathcal{R}_j(k-1), \mathcal{R}_j(k), \mathcal{R}_j(k+1), t_k\right)
\end{aligned}$$

Here $\phi$ and $\rho$ are placeholders. $\mathcal{R}_i(\phi, \rho)$ is an evaluation of $\mathcal{R}_q$ in the $i$ direction, and $\mathcal{R}_j(\rho)$ is an evaluation of $\mathcal{R}_q$ in the $j$ direction.

# Appendix B

# Variational Implicit Curves

A common approach to generating a 2D scalar field is to interpolate a set of 2D field value samples $(\mathbf{m}_i, v_i)$, where $v_i$ is the desired field value at point $\mathbf{m}_i$. One well-known technique is variational interpolation based on thin-plate splines, which produces a globally $C^2$ field. This scattered data interpolation technique has been used define implicit curves and surfaces [113].

The thin-plate spline $f(\mathbf{u})$ is defined in terms of points $(\mathbf{m}_i, v_i)$ weighted by coefficients $w_i$, and a polynomial $\mathcal{P}(\mathbf{u}) = c_1\mathbf{u}_x + c_2\mathbf{u}_y + c_3$:

$$f(\mathbf{u}) = \sum w_i \|\mathbf{u} - \mathbf{m}_i\|^2 \ln(\|\mathbf{u} - \mathbf{m}_i\|) + \mathcal{P}(\mathbf{u}) \tag{B.1}$$

The weights $w_i$ and coefficients $c_1$, $c_2$, and $c_3$ are found by solving a linear system defined by evaluating Equation B.1 at each known solution $f(\mathbf{m}_i) = v_i$. The resulting linear system is a dense matrix with dimension $(N + 4)^2$ . The coefficients determined by solving this system describe a variational solution which is guaranteed to interpolate all sample points $(\mathbf{v}_i, v_i)$ with second-derivative continuity while minimizing a global curvature metric [44].

# Appendix C

# Normalized Implicit Polygons

A 2D closed contour $\mathcal{C}$ can be represented by an implicit polygon defined using the following approach (reproduced from [22]). First, $\mathcal{C}$ is approximated with a set of line segments. Then for a segment $((x_1, y_1), (x_2, y_2))$, with $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$, a scalar field can be defined:

$$f(x, y) = \frac{1}{d}\left((x - x_1)(y_2 - y_1) - (y - y_1)(x_2 - x_1)\right) \tag{C.1}$$

A circular scalar field is also defined for the line segment:

$$t(x, y) = \frac{1}{d}\left[\left(\frac{d}{2}\right)^2 - (x - \frac{x_1 + x_2}{2})^2 - (y - \frac{y_1 + y_2}{2})^2\right] \tag{C.2}$$

This circular field is used to normalize the line segment field, defining a new scalar field $h$:

$$h = \sqrt{f^2 + 0.25 * (|t| - t)^2} \tag{C.3}$$

These normalized line segments are then combined using the $R$-conjunction operator:

$$F(h_1, h_2) = h_1 + h_2 - \sqrt{h_1^2 + h_2^2} \tag{C.4}$$

The resulting field is unsigned, but since $\mathcal{C}$ is closed a polygon point-containment test can be used to determine whether a distance value should be positive or negative.

# Bibliography


[1] Abdel-Malek, K., Blackmore, D., and Joy, K. Swept volumes: Foundations, perspectives and applications. *submitted to International Journal of Shape Modeling* (2000).

[2] Abdel-Malek, K., Yang, J., and Blackmore, D. Closed-form swept volume of implicit surfaces. In *Proceedings of ASME Design Engineering Technical Conferences* (2000).

[3] Abdel-Malek, K., and Yeh, H.-J. Geometric representation of the swept volume using jacobian rank-deficiency conditions. *Computer-Aided Design 29*, 6 (1997), 457–468.

[4] Adzhiev, V., Cartwright, R., Fausett, E., Ossipov, A., Pasko, A., and Savchenko, V. Hyperfun project: a framework for collaborative multidimensional f-rep modeling. In *Eurographics / ACM Workshop on Implicit Surfaces* (1999).

[5] Akkouche, S., and Galin, E. Adaptive implicit surface polygonization using marching triangles. *Computer Graphics Forum 20*, 2 (2001), 67–80.

[6] Akkouche, S., and Galin, E. Implicit surface reconstruction from contours. *The Visual Computer 20*, 6 (2004), 380–391.

[7] Akleman, E. Interactive construction of ray-quadrics. In *Proceedings of Implicit Surfaces 98* (1998), pp. 105–114.

[8] Akleman, E., and Chen, J. Constant time updateable operations for implicit shape modeling. In *Proceedings of Implicit Surfaces 99* (1999), pp. 73–80.

[9] Akleman, E., and Chen, J. Generalized distance functions. In *Proceedings of the International Conference on Shape Modeling and Applications* (1999), pp. 72–79.

[10] Alexe, A., Gaildrat, V., and Barthe, L. Interactive modelling from sketches using spherical implicit functions. In *Proceedings of AFRIGRAPH 2004* (2004), pp. 25–34.

[11] Allgre, R., Galin, E., Chaine, R., and Akkouche, S. The hybridtree: Mixing skeletal implicit surfaces, triangle meshes, and point sets in a free-form modeling system. *Graphical Models 68*, 1 (2006), 42–64.

[12] Angelidis, A., and Cani, M.-P. Adaptive implicit modeling using subdivision curves and surfaces as skeletons. In *ACM Symposium on Solid Modeling and Applications* (2002), pp. 45–52.

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

[14] Araújo, B., and Jorge, J. Curvature dependent polygonization of implicit surfaces. In *Proceedings of the XVII Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI 2004)* (2004).

[15] Barbier, A., Galin, E., and Akkouche, S. Complex skeletal implicit surfaces with levels of detail. In *Proceedings of WSCG* (2004), vol. 12, pp. 35–42.

[16] Barbier, A., Galin, E., and Akkouche, S. A framework for modeling, animating and morphing textured implicit models. *Graphical Models 67*, 3 (2004), 166–188.

[17] Barthe, L., Dodgson, N. A., Sabin, M. A., Wyvill, B., and Gaildrat, V. Two-dimensional potential fields for advanced implicit modeling operators. *Computer Graphics Forum 22*, 1 (2003), 23–33.

[18] Barthe, L., Mora, B., Dodgson, N., and Sabin, M. Interactive implicit modelling based on $c^1$ reconstruction of regular grids. *International Journal of Shape Modeling 8*, 2 (2002), 99–117.

[19] 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.

[20] Bischoff, S., and Kobbelt, L. Structure preserving cad model repair. *Computer Graphics Forum 24*, 3 (2005), 527 – 536.

[21] Bishop, R. L. There is more than one way to frame a curve. *American Mathematical Monthly 82*, 3 (1975), 246–251.

[22] Biswas, A., and Shapiro, V. Approximate distance fields with non-vanishing gradients. *Graphical Models 66*, 3 (2004), 133–159.

[23] Blackmore, D., Leu, M., and Wang, L. The sweep-envelope differential equation algorithm and its application to nc machining verification. *Computer-Aided Design 29*, 9 (1997), 629–637.

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

[25] Bloomenthal, J. *Graphics Gems*. Morgan Kaufmann, 1990, ch. Calculation of references frames along a space curve, pp. 567–571.

[26] Bloomenthal, J. *Graphics Gems IV*. Academic Press Professional Inc., 1994, ch. An implicit surface polygonizer, pp. 324–349.

[27] Bloomenthal, J., Ed. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc., 1997.

[28] Bloomenthal, J., and Shoemake, K. Convolution surfaces. In *Computer Graphics (Proceedings of SIGGRAPH 91)* (1991), vol. 25, pp. 251–257.

[29] Bloomenthal, J., and Wyvill, B. Interactive techniques for implicit modeling. In *Proceedings of the symposium on Interactive 3D graphics* (1990), pp. 109–116.

[30] Blum, H. A transformation for extracting new descriptors of shape. In *Models for the Perception of Speech and Visual Form*, W. Wathen-Dunn, Ed. MIT Press, 1967, pp. 362–380.

[31] Botsch, M., Pauly, M., Gross, M., and Kobbelt, L. Primo: Coupled prisms for intuitive surface modeling. In *Eurographics Symposium on Geometry Processing* (2006), pp. 11–20.

[32] Cani, M. P. An implicit formulation for precise contact modeling between flexible solids. In *Computer Graphics (ACM SIGGRAPH)* (1993), pp. 313–320.

[33] Cani, M. P. Implicit representations in computer animation : a compared study. In *Implicit Surfaces 99* (1999).

[34] Carr, J. C., Beatson, R. K., Cherrie, J. B., Mitchell, T. J., Fright, W. R., McCallum, B. C., and Evans, T. R. Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of ACM SIGGRAPH 2001* (2001), pp. 67–76.

[35] Chen, M., Kaufman, A. E., and Yagel, R., Eds. *Volume Graphics*. Springer, London, 2000.

[36] Cherlin, J. J., Samavati, F., Sousa, M. C., and Jorge, J. A. Sketch-based modeling with few strokes. In *Proceedings of the Spring Conference on Computer Graphics* (2005).

[37] Crespin, B., Blanc, C., and Schlick, C. Implicit sweep objects. *Computer Graphics Forum 15*, 3 (Aug. 1996), 165–174.

[38] Cuisenaire, O., and Macq, B. Fast and exact signed euclidean distance transformation with linear complexity. In *Proceedings of ICASSP 99* (1999), vol. 6, pp. 3293–3296.

[39] de Groot, E., and Wyvill, B. Rayskip: faster ray tracing of implicit surface animations. In *GRAPHITE 2005* (2005), pp. 31–36.

[40] Desbrun, M., and Cani, M.-P. Animating soft substances with implicit surfaces. In *Computer Graphics (ACM SIGGRAPH)* (1995), vol. 29, pp. 287–290.

[41] Desbrun, M., and Cani, M.-P. Active implicit surface for animation. In *Graphics Interface* (1998).

[42] Desbrun, M., Tsingos, N., and Gascuel, M.-P. Adaptive sampling of implicit surfaces for interactive modelling and animation. *Computer Graphics Forum 15*, 5 (1996), 319–325.

[43] Dinh, H., Slabaugh, G., and Turk, G. Reconstructing surfaces using anisotropic basis functions. In *International Conference on Computer Vision* (2001), pp. 606–613.

[44] Duchon, J. *Constructive Theory of Functions of Several Variables*. Springer-Verlag, 1977, ch. Splines minimizing rotation-invariant semi-norms in Sobolev spaces, pp. 85–100.

[45] Ferley, E., Cani, M.-P., and Gascuel, J.-D. Practical volumetric sculpting. *The Visual Computer 16*, 8 (2000), 469–480.

[46] Ferley, E., Cani, M.-P., and Gascuel, J.-D. Resolution adaptive volume sculpting. *Graphical Models 63*, 6 (2001), 459–478.

[47] Foley, J. D., van Dam, A., Feiner, S. K., Hughes, J. F., and Phillips, R. *Introduction to Computer Graphics*. Addison-Wesley, 1993.

[48] Fox, M., Galbraith, C., and Wyvill, B. Efficient implementation of the blobtree for rendering purposes. In *Proceedings of Shape Modeling International* (2001), pp. 306–314.

[49] Frisken, S., Perry, R., Rockwood, A., and Jones, T. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of SIGGRAPH 2000* (2000), pp. 249–254.

[50] Frisken, S. F., and Perry, R. N. Simple and efficient traversal methods for quadtrees and octrees. *Journal of Graphics Tools 7*, 3 (2002), 1–11.

[51] Galbraith, C. *Modeling Natural Phenomena with Implicit Surfaces*. PhD thesis, Department of Computer Science, University of Calgary, 2005.

[52] Galbraith, C., Muendermann, L., and Wyvill, B. Blobtree trees. *Computer Graphics Forum 23*, 3 (2004), 351–360.

[53] Galin, E., and Akkouche, S. Incremental polygonization of implicit surfaces. *Graphical Models 62*, 1 (1999), 19–39.

[54] Galin, E., Leclercq, A., and Akkouche, S. Morphing the blobtree. *Computer Graphic Forum 19*, 4 (2000), 257–270.

[55] Grimm, C. Implicit generalized cylinders using profile curves. In *Implicit Surfaces 99* (1999), pp. 33–41.

[56] Hart, J. C., Bachta, E., Jarosz, W., and Fleury, T. Using particles to sample and control more complex implicit surfaces. In *Proceedings of the Shape Modeling International 2002 (SMI 02)* (2002), pp. 129–136.

[57] Ho, C.-C., Wu, F.-C., Chen, B.-Y., Chuang, Y.-Y., and Ouhyoung, M. Cubical marching squares: Adaptive feature preserving surface extraction from volume data. *Computer Graphics Forum 24*, 3 (2005), 537–546.

[58] Hornus, S., Angelidis, A., and Cani, M.-P. Implicit modelling using subdivision-curves. 94–104.

[59] Igarashi, T., and Hughes, J. F. Smooth meshes for sketch-based freeform modeling. In *Proceedings of the 2003 symposium on Interactive 3D graphics* (2003), pp. 139–142.

[60] Igarashi, T., Matsuoka, S., and Tanaka, H. Teddy: A sketching interface for 3d freeform design. In *Proceedings of ACM SIGGRAPH 99* (1999), pp. 409–416.

[61] Jain, A. *Fundamentals of Digital Image Processing*. Prentice-Hall, 1989, ch. 2.

[62] Jevans, D., Wyvill, B., and Wyvill, G. Speeding Up 3D Animation For Simulation. *Proc. SCS Conference* (1988), 94–100. Proc. MAPCON IV (Multi and Array Processors).

[63] Jorge, J. A., Silva, N. F., and Cardoso, T. D. Gides++. In *Proceedings of 12th Encontro Português de Computação Gráfica* (2003).

[64] Ju, T., Losasso, F., Schaefer, S., and Warren, J. Dual contouring of hermite data. *ACM Transactions on Graphics 21*, 3 (2002), 339–346.

[65] Karpenko, O., Hughes, J., and Raskar, R. Free-form sketching with variational implicit surfaces. *Computer Graphics Forum 21*, 3 (2002), 585 – 594.

[66] Kobbelt, L. P., Botsch, M., Schwanecke, U., and Seidel, H.-P. Feature-sensitive surface extraction from volume data. In *Proceedings of ACM SIGGRAPH 2001* (2001), pp. 57–66.

[67] Lorensen, W. E., and Cline, H. E. Marching cubes: A high resolution 3d surface construction algorithm. In *Computer Graphics (Proceedings of SIGGRAPH 87)* (1987), vol. 21, pp. 163–169.

[68] Markosian, L., Cohen, J. M., Crulli, T., and Hughes, J. F. Skin: A constructive approach to modeling free-form shapes. *Proceedings of SIGGRAPH 99* (1999), 393–400.

[69] Marschner, S., and Lobb, R. An evaluation of reconstruction filters for volume rendering. In *Proceedings of Visualization 1994* (1994), pp. 100–107.

[70] Meyer, M. D., Georgel, P., and Whitaker, R. T. Robust particle systems for curvature dependent sampling of implicit surfaces. In *Proceedings of the International Conference on Shape Modeling and Applications 2005 (SMI 05)* (2005), pp. 124–133.

[71] Mora, B., Jessel, J.-P., and Caubet, R. Visualization of isosurfaces with parametric cubes. *Computer Graphics Forum 20*, 3 (2001), 377–384.

[72] Morse, B., Yoo, T., Rheingans, P., Chen, D., and Subramanian, K. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *Proceedings of Shape Modeling International* (2001), pp. 89–98.

[73] Museth, K., Breen, D. E., Whitaker, R. T., and Barr, A. H. Level set surface editing operators. *ACM Transactions on Graphics 21*, 3 (2002), 330–338.

[74] Nishimura, H., Hirai, A., Kawai, T., Kawata, T., Shirikawa, I., and Omura, K. Object modeling by distribution function and a method of image generation. *Journal of papers given at the Electronics Communications Conference 1985 J69-D*, 4 (1985). (In Japanese).

[75] Ohtake, Y., Belyaev, A., Alexa, M., Turk, G., and Seidel, H.-P. Multi-level partition of unity implicits. *ACM Transactions on Graphics 22*, 3 (2003), 463–470.

[76] Ohtake, Y., Belyaev, A., and Pasko, A. Dynamic mesh optimization for polygonized implicit surfaces with sharp features. *The Visual Computer 19*, 2 (2003), 115–126.

[77] Osher, S. J., and Fedkiw, R. P. *Level Set Methods and Dynamic Implicit Surfaces*. Springer, 2002.

[78] 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.

[79] Pasko, A., Adzhiev, V., Sourin, A., and Savchenko, V. Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer 11*, 8 (1995), 419–428.

[80] Pasko, A., Savchenko, A., and Savchenko, V. Implicit curved polygons. Tech. Rep. University of Aizu, Japan, Technical Report 96-1-004, 1996.

[81] Pasko, A., Savchenko, A., and Savchenko, V. Polygon-to-function conversion for sweeping. In *Proceedings of Implicit Surfaces 96* (1996), pp. 163–171.

[82] Pasko, G., Pasko, A., Ikeda, M., and Kunii, T. Bounded blending operations. In *International Conference on Shape Modeling and Applications* (2002), pp. 95–103.

[83] Perry, R. N., and Frisken, S. F. Kizamu: A system for sculpting digital characters. In *Proceedings of ACM SIGGRAPH 2001* (2001), Computer Graphics Proceedings, Annual Conference Series, pp. 47–56.

[84] Press, W. H., Flannery, B. P., and Teukolsky, S. A. *Numerical Recipes In C*. Cambridge University Press, October 1992.

[85] Requicha, A. A. G. Representations for rigid solids: theory, methods and systems. *Computing Surveys 12*, 4 (1980), 437–464.

[86] Requicha, A. A. G., and Voelcker, H. B. Solid modeling: a historical summary and contemporary assessment. *IEEE Computer Graphics and Applications 2*, 2 (1982), 9–24.

[87] Reuter, P. *Reconstruction and Rendering of Implicit Surfaces from Large Unorganized Point Sets*. PhD thesis, LABRI - Universite Bordeaux, 2003.

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

[89] Rowland, T. Manifold. From MathWorld–A Wolfram Web Resource, created by Eric W. Weisstein. http://mathworld.wolfram.com/Manifold.html.

[90] Rvachev, V. L. On the analytic description of some geometric objects. *Reports of the Ukrainian Acadamey of Sciences 153* (1963), 765–767.

[91] Rvachev, V. L., Sheiko, T. I., Shapiro, V., and Tsukanov, I. Transfinite interpolation over implicitly defined sets. *Computer Aided Geometric Design 18*, 4 (2001), 195–220.

[92] Savchenko, V., Pasko, A., Okunev, O., and Kunii, T. Function representation of solids reconstructed from scattered surface poitns and contours. *Computer Graphics Forum 14*, 4 (1995), 181–188.

[93] Schaefer, S., and Warren, J. Dual marching cubes: Primal contouring of dual grids. *Computer Graphics Forum 24*, 2 (2005), 195–203.

[94] Schmidt, R., Grimm, C., and Wyvill, B. Interactive decal compositing with discrete exponential maps. *Transactions on Graphics* (2006). To Appear.

[95] Schmidt, R., Isenberg, T., and Wyvill, B. Interactive pen-and-ink rendering for implicit surfaces. In *SIGGRAPH 2006* (2006). Extended Abstract (Sketches Program).

[96] Schmidt, R., and Wyvill, B. Generalized sweep templates for implicit modeling. In *3rd International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia (GRAPHITE 2005)* (2005), pp. 187–196.

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

[98] Schmidt, R., Wyvill, B., and Sousa, M. C. Sketch-based modeling with the blobtree. In *SIGGRAPH 2005* (2005). Extended Abstract (Sketches Program).

[99] Schmidt, R., Wyvill, B., Sousa, M. C., and Jorge, J. A. Shapeshop: Sketch-based solid modeling with blobtrees. In *Proceedings of the 2nd Eurographics Workshop on Sketch-Based Interfaces and Modeling* (2005). to appear.

[100] Schneider, P. J. *Graphics Gems*. Morgan Kaufmann, 1990, ch. A Bezier curve-based root finder, pp. 409–415.

[101] Sealy, G., and Wyvill, G. Smoothing of three dimensional models by convolution. In *Computer Graphics International 1996* (1996).

[102] Sealy, G., and Wyvill, G. Representing and rendering sweep objects using volume models. In *Computer Graphics International 1997* (June 1997), pp. 22–27.

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

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

[105] Shen, C., O'Brien, J. F., and Shewchuk, J. R. Interpolating and approximating implicit surfaces from polygon soup. In *Proceedings of ACM SIGGRAPH 2004* (2004), pp. 896–904.

[106] Sherstyuk, A. Interactive shape design with convolution surfaces. In *SMI '99: Proceedings of the International Conference on Shape Modeling and Applications* (1999), pp. 56–65.

[107] Sherstyuk, A. Kernel functions in convolution surfaces: a comparative analysis. *The Visual Computer 15*, 4 (1999), 171–182.

[108] Sourin, A., and Pasko, A. Function representation for sweeping by a moving solid. *IEEE Transactions on Visualization and Computer Graphics 2*, 1 (1996), 11–18.

[109] Stander, B. T., and Hart, J. C. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In *Proceedings of SIGGRAPH 97* (1997), Computer Graphics Proceedings, Annual Conference Series, pp. 279–286.

[110] Su, W. Y., and Hart, J. C. A programmable particle system framework for shape modeling. In *Proceedings of the International Conference on Shape Modeling and Applications 2005 (SMI 05)* (2005), pp. 114–123.

[111] Sutherland, I. E. Sketchpad: A man-machine graphical communication system. In *SJCC* (1963), Spartan Books.

[112] 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.

[113] Turk, G., and O'Brien, J. Shape transformation using variational implicit functions. In *Proceedings of SIGGRAPH 99* (1999), pp. 335–342.

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

[115] Weisstein, E. W. Boundary, July 2006. From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/Boundary.html.

[116] Winter, A. S., and Chen, M. Image-swept volumes. *Computer Graphics Forum 21*, 3 (2002), 441–450.

[117] Witkin, A. P., and Heckbert, P. S. Using particles to sample and control implicit surfaces. In *Proceedings of SIGGRAPH 94* (1994), Computer Graphics Proceedings, Annual Conference Series, pp. 269–278.

[118] Wyvill, B. *Graphics Gems I.* Morgan Kaufmann Publishers, 1993, ch. 3D grid hashing function, p. 343.

[119] Wyvill, B., Foster, K., Jepp, P., Schmidt, R., Sousa, M. C., and Jorge, J. A. Sketch based construction and rendering of implicit models. In *Eurographics Workshop on Computational Aesthetics in Graphics, Visualization and Imaging* (2005).

[120] 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.

[121] Wyvill, B., and van Overveld, K. Polygonization of Implicit Surfaces with Constructive Solid Geometry. *Journal of Shape Modelling 2*, 4 (1996), 257–274.

[122] Wyvill, B., and Wyvill, G. Better blending of implicit objects at different scales. In *ACM SIGGRAPH 2000* (2000). Extended Abstract (Sketches Program).

[123] Wyvill, G. Wyvill function. Personal Communication, 2005.

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

[125] Yngve, G., and Turk, G. Robust creation of implicit surfaces from polygonal meshes. *IEEE Transactions on Visualization and Computer Graphics 8*, 4 (2002), 346–359.

[126] Zeleznik, R. C., Herndon, K. P., and Hughes, J. F. Sketch: an interface for sketching 3d scenes. In *Proceedings of ACM SIGGRAPH 96* (1996), pp. 163–170.