Implicit Surfaces that Interpolate

Greg Turk and Huong Quynh Dinh
GVU Center, College of Computing
Georgia Institute of Technology

James F. O'Brien
EECS, Computer Science Division
University of California, Berkeley

Gary Yngve
Dept. of Computer Science and Engineering
University of Washington

Abstract

Implicit surfaces are often created by summing a collection of radial basis functions. Recently, researchers have begun to create implicit surfaces that exactly interpolate a given set of points by solving a simple linear system to assign weights to each basis function. Due to their ability to interpolate, these implicit surfaces are more easily controllable than traditional "blobby" implicits. There are several additional forms of control over these surfaces that make them attractive for a variety of applications. Surface normals may be directly specified at any location over the surface, and this allows the modeller to pivot the normal while still having the surface pass through the constraints. The degree of smoothness of the surface can be controlled by changing the shape of the basis functions, allowing the surface to be pinched or smooth. On a point-by-point basis the modeller may decide whether a constraint point should be exactly interpolated or approximated. Applications of these implicits include shape transformation, creating surfaces from computer vision data, creation of an implicit surface from a polygonal model, and medical surface reconstruction.

1. Introduction

In the field of computer graphics, the first complex shapes that were created using implicit functions appeared nearly 20 years ago. As often happens, two groups who were unaware of each others work independently discovered this new method of representing surfaces. Both James Blinn and Hitoshi Nishimura and his co-workers realized that a surface can be modeled using an implicit function that is the sum of several Gaussian radial basis functions [3, 16]. Such models (sometimes called *blobbies* or *metaballs*), have been used in a number of animations, including The Great Train Rubbery, the blood floating in zero gravity in Star Trek VI, and the creatures in Flubber. Despite these successes, blobby implicit surfaces are seldom the method of choice in 3D modelling and animation. We believe that the main reason for this is the lack of control that the modeller has when using blobby implicits. In particular, it is difficult to specify 3D points that the surface will pass through.

This paper describes a relatively new method of creating implicit surfaces that overcomes this chief limitation of implicit surfaces. We will use the phrase *interpolating implicits* to refer to this form of surface because they are created primarily by specifying points through which the surface will pass. The implicit functions that define these surfaces are similar to blobby implicits in that they are both defined as the sums of radial basis functions. Interpolating implicits differ from blobby implicits in a couple of ways, however. First, the weights of the basis functions are not specified by the modeller but are instead determined by solving a matrix equation. Second, the basis functions of interpolating implicits typically have much wider support than Gaussian implicits, and the reason for this is to create a smoother surface.

The remainder of this paper will describe interpolating implicits in greater detail. First, we will review the formulation of blobby implicits. We will then show how interpolating implicits differ from these by going over how various constraints are defined by the modeller when creating a new surface. We then survey several applications of interpolating implicits, including conversion of polygons to implicits, shape transformation, and surface reconstruction for computer vision and medicine. We end by describing several open problems having to do with interpolating implicits.

2. Blobby Implicit Surfaces

Before describing the math behind interpolating implicits, we will first review the notion of an implicit surface and go over the more traditional blobby implicit formulation.

Implicit functions can describe shapes in any dimension, although we will restrict ourselves to 2D and 3D shapes in this paper. An *implicit function* is a function that, given a point $\mathbf{p}$, returns a scalar value $f(\mathbf{p})$ that specifies whether the point is inside, on, or outside the shape that is being described. In this paper, we will use positive function values, $f(\mathbf{p}) > 0$, to mean that the point $\mathbf{p}$ is inside a shape, and $f(\mathbf{p}) < 0$ will mean that the point is outside. Those positions where the function evaluates to zero are exactly on the shape. In 2D, the set $\{\mathbf{p} \mid f(\mathbf{p}) = 0\}$ (locations where the function is zero) is an *implicit curve* and in 3D this set is an *implicit surface*. The set $\{\mathbf{p} \mid f(\mathbf{p}) = 0\}$ is sometimes referred to as a *level set* of the function f for the value zero. Unfortunately the term *level set* has also recently come to be associated with a method that uses partial differential equations to create an evolving implicit surface over time.

There are many kinds of functions that have been used to describe implicit surfaces. Quadratic functions can be used to describe simple surfaces such as spheres, ellipsoids and cylinders [13]. Higher degree polynomial functions can be used to create more complex shapes, although controlling these shapes can be quite difficult [19, 12]. Algebraic patches can be used to gain more control over surface creation, although there is a price to pay in terms of the machinery needed to maintain smoothness between patches [2]. These and many other kinds of implicit surfaces are described in the excellent book that is edited by Bloomethal [6]. We will now look at one formulation of implicit surfaces in more detail, namely blobby implicits.

A *radial basis function* is a function that can be described in terms of a center point $\mathbf{c}$ and a function $h(x)$ over the non-negative real numbers. A radial basis function $b(\mathbf{p})$ can be written as $b(\mathbf{p}) = h(|\mathbf{c} - \mathbf{p}|)$, which means that the function b gets its value by evaluating h of the distance between $\mathbf{p}$ and the center $\mathbf{c}$. The term *radial* comes from the fact that $b(\mathbf{p})$ evaluates to exactly the same value for all of the points that are at a fixed radius from the $\mathbf{c}$. There is no restriction on the behavior of the function $h(x)$ that defines a radial basis function $b(\mathbf{p})$, although h is nearly always monotonically decreasing when used for 3D modelling.

A blobby implicit function is a sum of Gaussian radial basis functions. Typically these functions have the following form:

$$f(\mathbf{p}) = O + \sum_{i=1}^{n} h_i(|\mathbf{c}_i - \mathbf{p}|)) \tag{1}$$

In the above equation there is a distinct center point $\mathbf{c}_i$ and an associated function $h_i(x)$ that together are used to create each individual radial basis function. The scalar quantity O (which may be negative) is an offset term. Each Gaussian radial basis function may be written in the following form:

$$h_i(x) = a_i e^{x^2/b_i^2} \tag{2}$$

In this equation, the values a_i and b_i specify the maximum value and the standard deviation of each basis function. The basic controls that a modeller has over a blobby implicit function are the values for $\mathbf{c}_i$, a_i and b_i. Notice that none of these parameters allow the modeller to specify exactly where the surface is located. The implicit surface may or may not pass near a given center position $\mathbf{c}_i$, depending on where other centers are located. This means that the modeller has only indirect control over the shape of a a blobby implicit surface.

Some research has been done on how to make modelling with blobby implicits more direct. In particular, Witkin and Heckbert created a way of allowing the user to directly move control points that a blobby surface is to pass through [21]. Unfortunately, the user must also specify exactly which parameters of which basis functions are free and which are fixed while dragging around the surface. Such a method is reasonable for modelling simple shapes, but quickly becomes unwieldy for modelling complex shapes.

3. Interpolating Implicit Surfaces

We advocate a different method of creating implicit functions, one that allows the modeller to directly specifies points through which the surface will pass. As was the case for blobby implicits, these interpolating implicit functions were independently discovered by more than one research group. The earliest paper of which we are aware that uses this approach is by Savchenko et al. [18]. They make use of interpolating implicit functions to construct a 3D surface from scattered points. Turk and O'Brien also used this same basic approach to create implicit functions that they use to produce shape transformations [20]. Both of these approaches use the same matrix formulation to weight a collection of radial basis functions. The methods differ only in how they specify the location of the interior of a shape, and in this paper we will follow the approach in [20].

3.1 Constraint Specification

An interpolating implicit function is defined in terms of a collection of constraint locations and the values at each of these locations. We will use the set of points $\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_k\}$ to represent the constraint locations, and we will use $\{v_1, v_2, \ldots, v_k\}$ to specify the values at these locations. For now, let us consider the case where we wish to specify a curve in 2D, so that the constraint positions lie in the plane. The modeller may wish to think of these constraints as falling into several categories, although they are all treated identically when solving for the unknowns

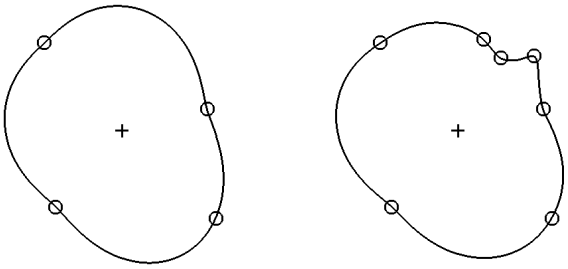

Figure 1. Interior constraints.

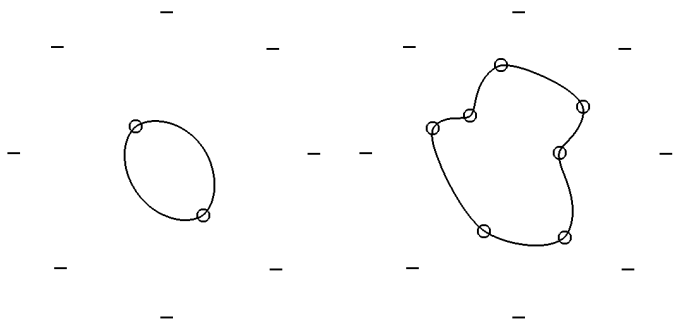

Figure 2. Exterior constraints.

that will define the implicit function. One category of constraint is a *boundary constraint*, which are those locations where the modeller wishes the surface to pass through. At a boundary constraint position $\mathbf{p}_i$, the value is equal to zero: $v_i = 0$. Another category of constraint is the *interior constraint*, and at these locations the value of the constraint is positive. In most cases it suffices for all such locations to have $v_i = 1$. These are the locations that the modeller knows should be interior to the shape. Figure 1 (left) shows four boundary constraints, shown as open circles, and one interior constraint, shown as a plus sign. The shape that is defined by these constraints is also drawn, and it is the curve that passes through the boundary constraints. On the right of this figure three additional constraints have been added to refine the shape.

Another category of constraint is the *exterior constraint*. These are the locations that the modeller knows should be exterior to the shape. At an exterior constraint, the constraint value is negative, typically $v_i = -1$. Figure 2 (left) shows a shape that is defined in terms of just two boundary constraints that are surrounded by eight exterior constraints. The ellipse-like curve that is created is quite a reasonable closed shape that passes through the two constraints. A more complex shape is shown in the right portion of this figure. Notice that in this figure as well, the curves pass exactly through the boundary constraints. This is due to the interpolating property of this form of implicit function. Boundary, interior and exterior constraints may be freely mixed when specifying the constraints for a shape. At least one interior or exterior constraint is necessary in order for the resulting implicit function to be non-degenerate. For many shapes just a few interior or exterior constraints are necessary.

There is one final category of constraint that is closely related to interior and exterior constraints, and this is the *normal constraint*. A normal constraint is simply an interior or exterior constraint that has been placed near a boundary constraint. (For the purposes of this discussion we will limit ourselves to normal constraints that are interior constraints, but everything we say also applies to exterior constraints.) Due to the nature of interpolating implicit functions, pairing up an interior constraint with a boundary constraint actually has the effect of specifying the surface normal at the corresponding boundary constraint. Figure 3 (left) illustrates six boundary constraints (open circles) that each have a normal constraint (plus signs) near to them. If the upper left normal constraint is moved, we get the shape that is shown in the right portion of this figure. If we drew a line through a boundary constraint and its paired normal constraint, we would find that this line was almost exactly perpendicular to the curve, that is, this line is nearly parallel to the normal of the curve. The closer the normal constraint is to its boundary constraint, the more exact this match is. This means that a modeller not only has control over the location of a curve or a surface, but also control over the normal to the curve or surface. This is another property that blobby implicits do not possess.

3.2 Creating the Implicit Function

We have just described how a modeller creates a shape by specifying a collection of constraints. Now we will discuss how these constraints define an implicit function. Interpolating implicit functions are created using a method that is well-known for performing scattered data interpolation. We wish to create a function $f(\mathbf{p})$ so that this function will take on the constraint value v_i at each of the constraint po-

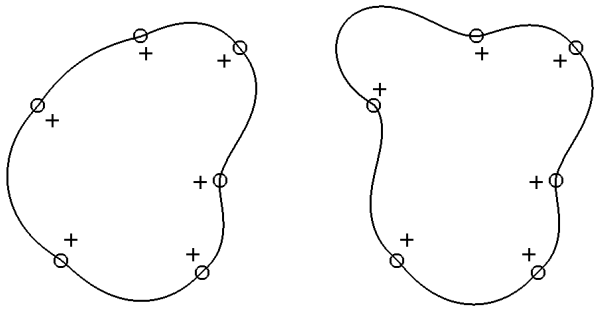

Figure 3. Normal constraints.

sitions $\mathbf{p}_i$:

$$f(\mathbf{p}_i) = v_i \tag{3}$$

Our goal is to find weights for a number of radial basis functions, one basis function that is centered on each of our constraints. We will do this by creating a set of linear equations, one equation for each constraint. The implicit function will have the following form:

$$f(\mathbf{p}) = \sum_{j=1}^{n} w_j h(|\mathbf{p} - \mathbf{p}_j|) \tag{4}$$

In the above equation, the values w_j are the unknown weights that we must find. We will discuss suitable choices for the scalar function $h(x)$ later. Note that we use the same function $h(x)$ for all of the constraints, and that only the weights change at the different constraint locations. Often there is an additional low-order polynomial term that is added to this equation, but we have found this unnecessary when there are more than about a dozen constraints.

Now that we have seen the general form of the implicit function, we are ready to create one linear equation for each constraint. If we examine just a single constraint with position $\mathbf{p}_i$ and value v_i and combine equations 3 and 4, we get:

$$v_i = \sum_{j=1}^{n} w_j h(|\mathbf{p}_i - \mathbf{p}_j|) \tag{5}$$

The only unknowns in the above equation are the weights w_j. If we have k constraints, then we can create k equations similar to equation 5, giving us k equations and k unknowns $\{w_1, w_2, \ldots, w_k\}$. If we let $h_{ij} = h(|\mathbf{p}_i - \mathbf{p}_j|)$, then the entire set of equations can be written as a single matrix equation:

$$\begin{bmatrix} h_{11} & h_{12} & \ldots & h_{1k} \\ h_{21} & h_{22} & \ldots & h_{2k} \\ \vdots & \vdots & & \vdots \\ h_{k1} & h_{k2} & \ldots & h_{kk} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_k \end{bmatrix} \tag{6}$$

There are many ways this matrix equation can be solved to find the unknowns $\{w_1, w_2, \ldots, w_k\}$. We have often solved it using LU decomposition, but at times we have also found SVD (singular value decomposition) or the conjugate gradient method to be useful. Once the values for all w_j have been determined, then equation 4 gives us our implicit function. By construction, this function evaluates to zero at all boundary constraints, thus the shape it defines exactly passes through all boundary constraints.

The only issue that we have left until now is the nature of the function $h(x)$. For reasons first discovered by Duchon [9] and discussed in [20], $h(x) = x^2 \log x$ is a good choice when creating 2D curves, and $h(x) = x^3$ is a good choice for creating 3D surfaces. These basis functions create very smooth implicit functions, which is often a goal when creating free-form models. We will discuss additional choices of basis functions later in this paper.

Now that we have seen the basic formulation of interpolating implicit functions, we can start to look at applications that make use of this shape representation.

4. Free-Form Modelling

The first application of interpolating implicit surfaces that we will examine is free-form modelling. As described in the previous section, the main task in constructing these surfaces is in specifying the constraint points. One approach is to allow a user to directly specify 3D points that constrain the surface. We have written a 3D modelling program that lets a user do exactly this. Our program initializes the surface as a collection of four boundary constraints placed at the corners of a regular tetrahedron, and these are surrounded by a group of exterior constraints that are at the corners of a cube. This 3D arrangement is similar to the way the curves in Figure 2 were specified in 2D. Figure 4 (left) shows the shape that this initial 3D configuration defines. A user can perform several operations on the current shape:

- Move a boundary constraint.
- Create a new boundary constraint.
- Create and move a normal constraint.

The most basic of these operations is to move an already existing boundary constraint. When the user drags a boundary constraint with a pointer, this changes the position of the boundary constraint. We re-solve the matrix equation 6, and this gives a new set of weights for the basis functions. Due to the interpolating nature of these functions, the new surface that these weights define will still pass through the moved constraint. For up to a few dozen constraint points, the matrix solution can be performed at interactive rates.

When the user wishes to add more constraints to the surface, he or she clicks the pointer over the existing surface. A ray is cast from this position into the image, and the intersection point with the surface becomes the position of a new boundary constraint. This new boundary constraint is now included in the collection of already existing constraints. Notice that because all boundary constraints have constraint values of zero, this new constraint will not affect the surface in any way. Now, however, the user can push or

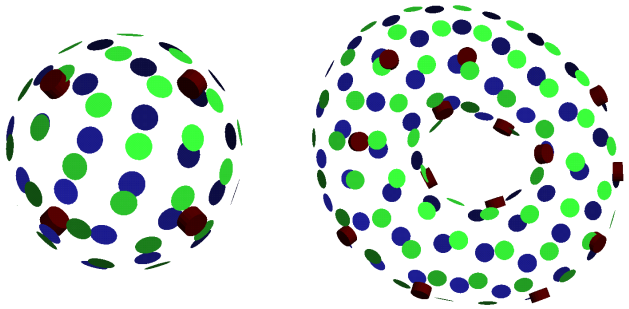

Figure 4. Sculpted implicit surfaces.

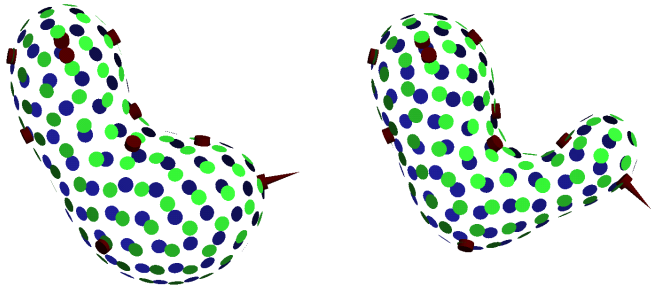

Figure 5. Changing a normal constraint.

pull on the surface at this location by dragging the new constraint. Adding a new boundary constraint exactly on the existing surface is much like adding a new knot in a spline curve – the curve remains the same shape but the user now has an additional degree of freedom that may be changed. Figure 4 (right) shows a torus that was made by creating a number of boundary constraints that forced the surface to re-join itself.

Another operation that the user may perform is to create a normal constraint. The user selects a boundary constraint and indicates that he or she wishes to change the surface normal at this location. Suppose that the boundary constraint is at position $\mathbf{p}$ and that the surface normal at this location is $\mathbf{n}$. The system adds a new external constraint with a position $\mathbf{p} + \mathbf{n}$ and a value of $f(\mathbf{p} + \mathbf{n})$. The user may then grab the normal constraint (represented as a long cone) and pivot its position around its corresponding boundary point. Thus the position of the normal constraint is moved, but the value remains fixed. This has the effect of altering the surface normal at the boundary constraint while still creating a surface that passes through it. Figure 5 (and Color Plate 1) shows a surface with a normal constraint. The left and right portions of the figure contain exactly the same boundary constraints, but the normal constraint has been pivoted to two different positions and this creates two different shapes.

Our modelling system borrows a very successful display technique from Witkin and Heckbert, namely their "floater" particles. A floater particle is a point that remains on the surface over time, even when the surface is changing shape. The floaters repel one another in order to evenly populate the surface, and floaters may be divide or removed in order to maintain an even coverage of the surface if its surface area changes. These floater particles are passive particles, and do not participate in the definition of the surface. This is in contrast to Witkin and Heckbert's control particles, which they use as a way to manipulate the surface. We do not use control particles. Our boundary constraints give the user fine control, but do not require the user to specify the free and fixed parameters of the surface, as is necessary when using Witkin and Heckbert's control particles.

5. From Polygons to Implicits

Despite the ability to model free-form shapes using interpolating implicit surfaces, many applications require the use of models that have already been constructed. Thus it is important to be able to convert an already existing model into an implicit form. Collections of polygons are by far the most common representations of 3D models. There are actually several ways that an existing polygonal model may be converted to an interpolating implicit surface.

The first method was originally described in [20], and this method used the vertices and surface normals of a polygonal model to create boundary and normal constraints for an implicit surface. Creating boundary constraints is easy – each vertex position is used for a boundary constraint. If a vertex $\mathbf{v}$ has a surface normal $\mathbf{n}$, then a normal constraint may be placed at the location $\mathbf{v}+\mathbf{n}$, and any negative value (e.g. -1) may be used as the value of this normal constraint. Once the boundary and normal constraints are defined, the implicit surface is created using the technique that is described in Section 3. This method is quite successful when the model is described by polygons that are fairly uniform in size over the entire surface. Figure 7 shows a polygonal bunny with 800 vertices (left) and the interpolating implicit version of this model (right) that was created in the manner just described.

Sometimes a model will contain polygons that vary a great deal in size over different portions of the model. In such an instance, the simple method of creating boundary and normal constraints from the vertices is not sufficient. Yngve describes an iterative approach to converting a polygonal model to an implicit model that does not have this polygon size restriction. We will give a brief overview of this method, and refer the reader to [22] for the details.

The first step in the iterative conversion process is to create a sampled signed distance function for the input model. This is a 3D grid in which each voxel contains the distance to the nearest point on the input model. Using this, the

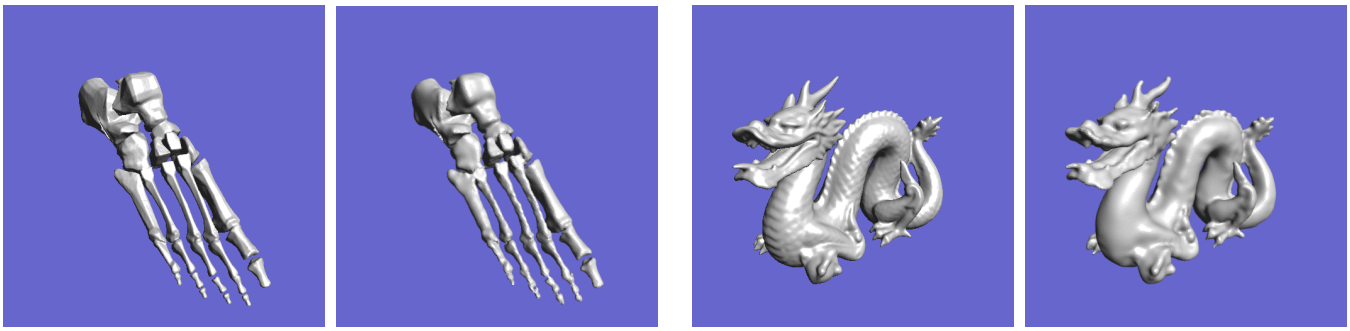

Figure 6. Conversion from polygons to interpolating implicits, performed using the iterative approach.

method places an initial set of boundary, interior and exterior constraints on and around the original surface. Then an implicit function is created for these constraints, and the volume is queried to find out how good a match this implicit function is to the original surface. In places where the match is poor, more constraints are added and then a new implicit function is created using the original constraints and the new ones. This process of refinement continues until the implicit function is a close match to the input model.

Figure 6 shows the result of the iterative method of converting polygonal models to interpolating implicit surfaces. The first input model (far left) is a collection of human foot bones, and the implicit version of this model is shown at in the second image. The third image in this figure is a dragon model with over 800,000 faces, and the image at the far right shows the implicit form.

6. Surface Reconstruction

Polygonal models are just one source of 3D data points that can be used to make constraints for an implicit surface. Another form of data that may be used to create a surface is computer vision data. There are a wide variety of methods that use multiple photographs to determine the location of 3D points on the surfaces of objects. Some of these methods include shape from shading, voxel coloring, various forms of triangulation, structure from motion, and so on. Many of these techniques may produce noisy or sparse range images, and reconstructing a surface from such data can be quite difficult. Interpolating implicit functions may be used to reconstruct surfaces from noisy and sparse data, and we describe work on this problem by Dinh et al. [8]. The original paper by Savchenko et al. also addresses the surface reconstruction problem [18].

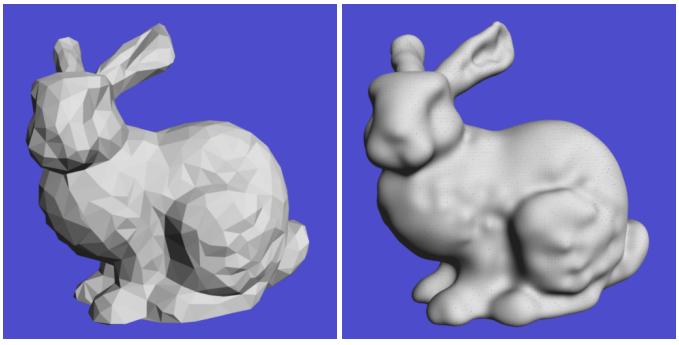

Figure 7. A polygonal model (left) and an interpolating implicit version of the model (right).

Given computer vision data in the form of a range image, boundary constraints for an implicit function can be created simply by using the positions of the range points. A more difficult problem is determining where to place interior or exterior constraints. If only 3D data points are given, there is no obvious answer to this problem. If, however, we also know the position of the camera use to determine the surface points, then we can put this information to use. The region of space swept out by a line from the camera to a range point on the surface must be entirely unblocked, thus exterior points can be placed anywhere along such a line. Using data taken from several range images, it is also possible to determine with high confidence the 3D locations of points that can be used as interior constraints. Using boundary, interior and exterior constraints from computer vision data, objects such as the toy dinosaur in Figure 8 (and Color Plate 2) may be reconstructed. At the left of this figure is a surface reconstruction using a polygonal approach called Crust [1], in the middle is the interpolating implicit reconstruction, and at the right is the implicit surface colored from the photographs of the object. Notice how much smoother the implicit reconstruction is than the polygonal reconstruction.

Often there is a certain amount of noise associated with the positions of points in a range image. When this is the case, it is preferable to approximate rather than to exactly interpolate the data. The matrix equation 6 can be modified

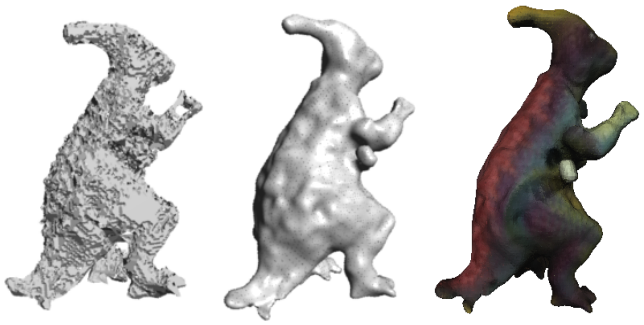

Figure 8. Dinosaur model reconstructed using Crust (left), interpolating implicits (middle), and as an implicit surface with color (right).

to allow the constraint points to be approximated or interpolated on a point-by-point basis. Here is the modified matrix equation:

$$\begin{bmatrix} \lambda_1 + h_{11} & h_{12} & \dots & h_{1k} \\ h_{21} & \lambda_2 + h_{22} & \dots & h_{2k} \\ \vdots & \vdots & & \vdots \\ h_{k1} & h_{k2} & \dots & \lambda_k + h_{kk} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_k \end{bmatrix}$$

When constraint number i is to be exactly matched, then the value λ_i should be set to zero. If this constraint is to be approximated instead, then λ_i should be set to be non-zero. The larger the value of λ_i, the further the implicit surface is allowed to deviate from the constraint point. We refer the reader to [8] for more details.

In addition to describing how to reconstruct surfaces from range data, Dinh also demonstrated the uses of a different family of radial basis functions. These basis functions, first described by Chen and Suter [7], allow the user to select the degree of smoothness of the implicit surface. Figure 9 shows the effect of varying one of the smoothness parameters (τ) of this radial basis family. When τ is close to zero, the surface can capture sharp features of a model (left). When τ is larger the surface is more smooth, but some of the more fine features are lost. Again, see [8] for the details.

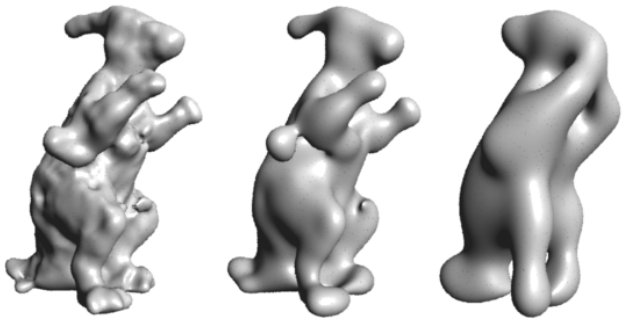

Figure 9. Effect of varying τ on the smoothness of the surface. implicits (middle), and as an implicit surface with color (right).

7. Shape Transformation

There are several geometric operations that are best performed using models that are in an implicit form. We believe that shape transformation is such an operation. The goal of shape transformation is that, given two shapes A and B, produce a sequence of shapes that smoothly change from A to B. Interpolating implicit functions can be used to perform shape interpolation by casting the problem as a shape creation problem in one higher dimension [20]. To transform a 3D surface A into another surface B, we place the constraints that describe each shape into a 4D space described by coordinates of the form (x, y, z, w). For each constraint (x, y, z) of shape A, we create a 4D constraint at the position $(x, y, z, 0)$. Similarly, the constraints of shape B are placed in the $w = 1$ sub-space, so each of these constraints is of the form $(x, y, z, 1)$. The 4D constraints from both shapes are collected together into one large set of constraints, and these constraints define an implicit function in four dimensions. A three dimensional slice through this function at the value $w = 0$ yields shape A, and a slice at $w = 1$ gives us shape B. Slices between these two locations, $0 < w < 1$, result in shapes that are intermediate between the two original shapes.

Figure 10 illustrates a shape transformation sequence between a knot and a human fist. Note that this method of shape interpolation gracefully handles changes of topology.

8. Rendering

In this section we examine two traditional approaches for rendering iso-surfaces that both perform well for interpolating implicit surfaces.

8.1 Iso-Surface Extraction

One way to render an implicit surface is to create a set of polygons that approximate the surface. The most common method of creating polygons for an implicit surface is to divide up a region of 3-space into regular cells such as cubes or tetrahedra, and to create polygons that approximate the surface within each cell. Perhaps the best known approach of this type is the Marching Cubes algorithm [14].

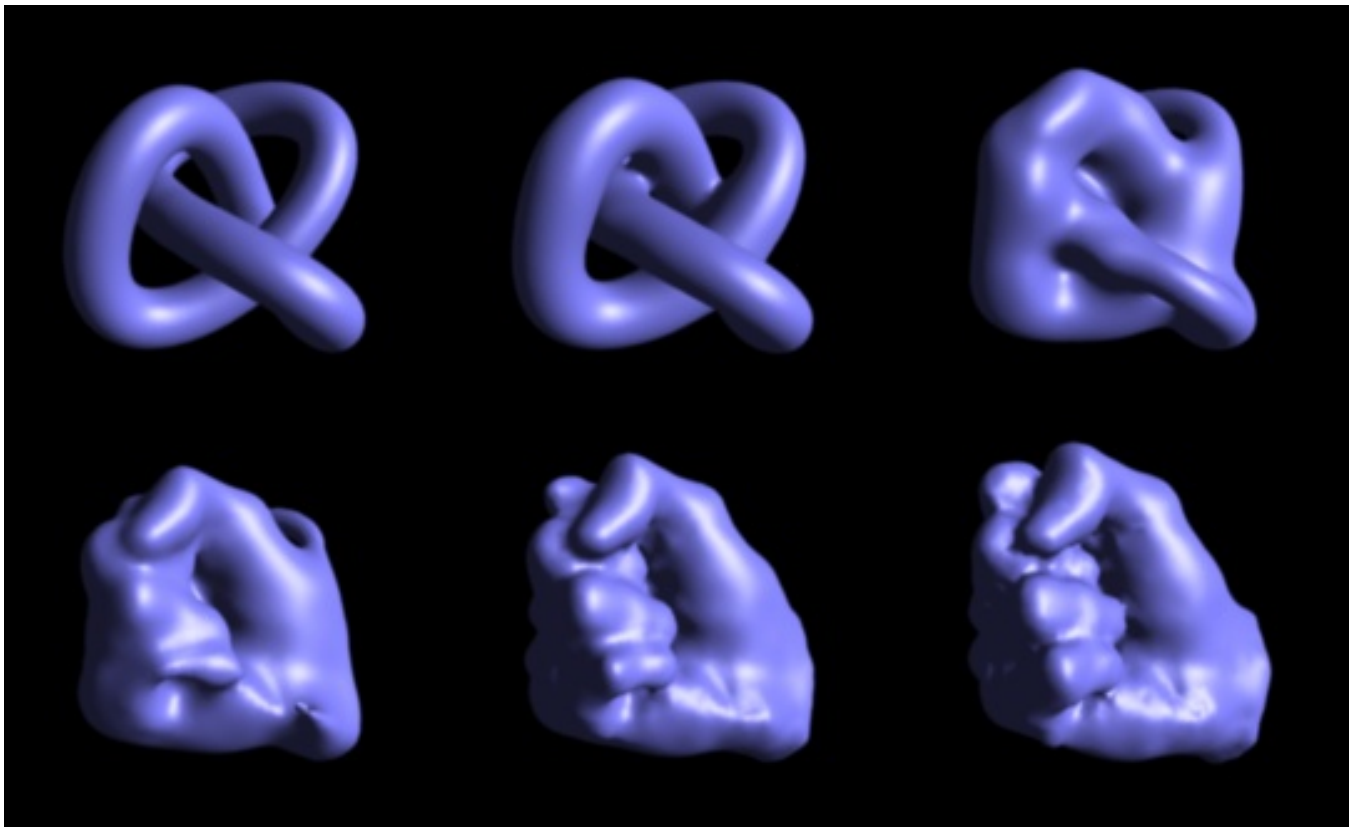

Figure 10. Shape transformation between a knot and a fist.

When an implicit function is to be extracted from a measured dataset such as from medical CT or MRI, an isosurface extraction algorithm typically examines each voxel of the given volume. For an analytically-defined implicit function such as the kind described in this paper, there is no pre-defined set of voxels to traverse. We may perform isosurface extraction using an algorithm known as a *continuation* approach. The model in Figure 7 (right) was rendered from an iso-surface that was created using the continuation method. This method first locates any position that is on the surface to be tiled. The continuation method then examines the values of the implicit function at neighboring points on the cubic lattice and creates polygons within each cube that the surface must pass through. The neighboring vertices of these cubes are examined in turn, and the process eventually crawls over the entire surface defined by the implicit function. We use the implementation of this method from [5] that is described in detail by Bloomenthal in [4].

8.2 Ray Tracing and CSG

There are a number of techniques that may be used to ray trace implicit surfaces, and a review of these techniques can be found in [10]. We have produced ray traced images of interpolating implicit surfaces using a particular technique introduced by Hart that is known as sphere tracing [11]. The basis for this method is that some implicit functions (including those that we are interested in) have what is called the *Lipschitz property*. A function f is said to have the Lipschitz property if and only if there exists some positive constant k such that:

$$|f(\mathbf{x}) - f(\mathbf{y})| \leq k\|\mathbf{x} - \mathbf{y}\| \tag{7}$$

The smallest k that satisfies the above equation is called the Lipschitz constant.

Sphere tracing is based on the idea that we can find the intersection of a ray with a surface by traveling along the ray in steps that are small enough to avoid passing through the surface. We declare that we have intersected the surface when the value of $f(\mathbf{p})$ falls below some tolerance ϵ. At any point $\mathbf{p}$ on a ray, we decide on step size by determining the radius of a sphere that is guaranteed not to intersect the surface. Assuming that k is the Lipschitz constant for the function f, the radius of such a "safe" sphere at $\mathbf{p}$ is $f(\mathbf{p})/k$. For interpolating implicit functions, we have used a simple heuristic to determine an appropriate value for k. We sample the space in and around our implicit surface at 2000 positions, and we use for k the maximum gradient magnitude over all of these locations. For extremely pathological surfaces this heuristic may fail, although it has worked well

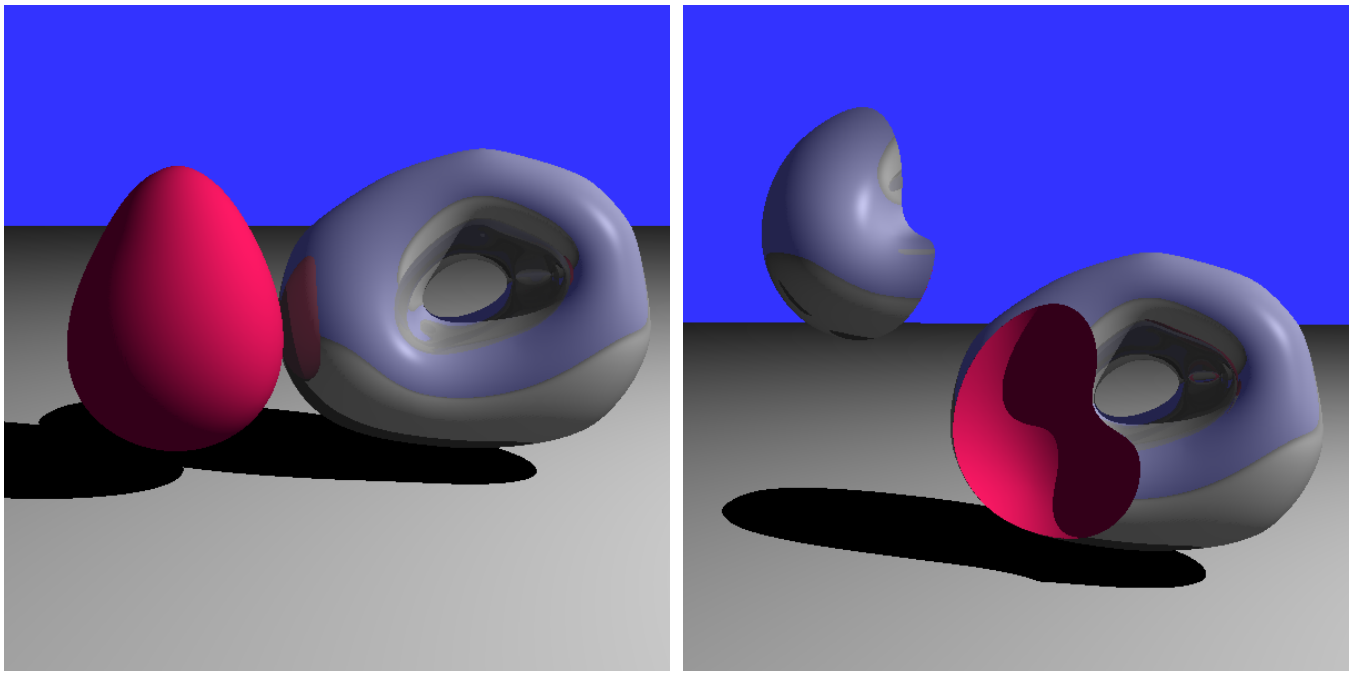

Figure 11. Ray tracing of interpolating implicit surfaces. The left image shows reflection and shadows of two implicit surfaces, and the right image illustrates constructive solid geometry.

for all of our images. Finding guaranteed bounds for the Lipschitz constant of an interpolating implicit function is a good area for future work.

Figures 11 (left) is an image of two interpolating implicit surfaces that were ray traced using sphere tracing. Note that this figure includes shadows and reflections. Figure 11 (right) illustrates constructive solid geometry with interpolating implicit surfaces. The figure shows (from left to right) intersection and subtraction of two implicit surfaces. This figure was created using standard ray tracing CSG techniques as described in [17].

The rendering techniques of this section highlight a key point – interpolating implicit surfaces may be used in almost all of the contexts in which other implicit formulations have been used.

9. Open Problems

Despite the variety of algorithms and applications that have been developed using interpolating implicit surfaces, there are still a number of open research problems related to them. It is our hope that other researchers will become interested in this new method of creating implicit surfaces and will resolve some of these issues.

Perhaps the most important issue related to interpolating implicits is that there is a practical limit on the number of constraints that may be used to define an implicit surface. Beyond around 4,000 constraints, solving matrix equation 6 becomes a major computational bottleneck. What would be ideal is to modify the implicit function formulation or solution method so that even millions of constraints could be used to construct a surface. One possible avenue towards this is to explore other radial basis functions whose properties allow the matrix to be solved faster and would allow the implicit function to be evaluated more rapidly. Morse et al. have recently made progress on this problem [15].

Another direction for future research is to find higher-level interactive modelling techniques for creating these implicit surfaces. Perhaps several new constraints could be created simultaneously, maybe arranged in a line or in a circle for greater surface control. It might also make sense to be able to move the positions of more than one constraint at a time. Another modelling issue is the creation of surfaces with boundaries. Perhaps a second implicit function could specify the presence or absence of a surface. Another issue related to interactivity is the possibility of displaying the surface with polygons rather than with floaters. We think it is likely that with sufficient processor power, creating and displaying a polygonal isosurface of the implicit function can be done at interactive rates.

Although we have described how to create normal constraints, the method we use only approximately specifies the normal to the surface. For some applications this approximation may suffice, but in other cases it may be desirable to specify the surface normal exactly. How might the process of creating an implicit function be modified to incorporate exact surface normals?

Finally, what other applications may make use of this form of implicit surface? Because of its ability to handle sparse and noisy data, we speculate that some medical applications may benefit from this surface representation. In fact, some researchers have already begun to use interpolating implicits for reconstructing surfaces from irregularly spaced data slices [23]. Another potential application is to time-varying data. The shape transformation method of Section 7 shows that we can easily create implicit functions in four dimensions. Time sequences of 3D shapes can be thought of as 4D data, and interpolating implicits may be a good way to represent and manipulate such data.

10. Conclusion

Despite their long history, implicit surfaces have never really become a widely-used representation for surface in computer graphics. One possible reason for this is the difficulty of controlling such surfaces. Interpolating implicit surfaces offer an alternative to the traditional sum-of-Gaussian implicit surfaces. This method of creating implicit surfaces has several benefits over other implicit methods, and we hope that more researchers will start to explore the possibilities of this approach to surface creation.

11. Acknowledgements

We thank the many friends who have encouraged us in our research in this area. This work was funded by ONR grant N00014-97-1-0223.

References

[1] N. Amenta, M. Bern, and M. Kamvysselis. A new voronoi-based surface reconstruction algorithm. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 98)*, pages 415–420, Aug. 1998.

[2] C. L. Bajaj, F. Bernardini, and G. Xu. Automatic reconstruction of surfaces and scalar fields from 3d scans. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 95)*, pages 109–118, Aug. 1995.

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

[4] J. Bloomenthal. Polygonization of implicit surfaces. *Computer-Aided Geometric Design*, 5(4):341–355, 1988.

[5] J. Bloomenthal. An implicit surface polygonizer. In P. S. Heckbert, editor, *Graphics Gems IV*, pages 324–349. Academic Press, 1994.

[6] J. Bloomenthal, editor. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers, Inc., San Francisco, California, 1997.

[7] F. Chen and D. Suter. Multiple order laplacian spline - including splines with tension. Technical report, Monash University, July 1986. MECSE 1996-5, Dept. of Electrical and Computer Systems Engineering Technical Report.

[8] H. Q. Dinh, G. Turk, and G. Slabaugh. Reconstructing surfaces by volumetric regularization. Technical report, College of Computing, Georgia Institute of Technology, Nov. 2000. Tech Report GIT-GVU-00-26.

[9] J. Duchon. Spline minimizing rotation-invariant semi-norms in sobolev spaces. In W. Schempp and K. Zeller, editors, *Constructive Theory of Functions on Several Variables, Lecture Notes in Mathematics 571*, Berlin, 1977. Springer-Verlag.

[10] J. Hart. Ray tracing implicit surfaces. *Siggraph 93 Course Notes: Design, Visualization and Animation of Implicit Surfaces*, pages 1–16, 1993.

[11] J. Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(10):527–545, 1997.

[12] D. Keren and C. Gotsman. Tight fitting of convex polyhedral shapes. *International Journal of Shape Modeling*, pages 111–126, 1998.

[13] J. Levin. A parametric algorithm for drawing pictures of solid objects composed of quadric surfaces. *Communications of the ACM*, 19:555–563, 1976.

[14] W. Lorensen and H. E. Cline. Marching cubes: A high resolution 3-d surface construction algorithm. *Computer Graphics (SIGGRAPH 87)*, 21(4):163–169, July 1987.

[15] B. Morse, T. S. Yoo, P. Rheingans, D. T. Chen, and K. Subramanian. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. *Shape Modelling International*, 2001.

[16] H. Nishimura, M. Hirai, T. Kawai, T. Kawata, I. Shirkawa, and K. Omura. Object modeling by distribution function and a method of image generation. *Transactions of the Institute of Electronics and Communication Engineers of Japan*, J68-D(4):718–725, 1985.

[17] S. Roth. Ray casting as a method for solid modeling. *Computer Graphics and Image Processing*, 18(2):109–144, 1982.

[18] V. V. Savchenko, A. A. Pasko, O. G. Okunev, and T. L. Kunni. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum*, 14(4):181–188, Oct. 1995.

[19] G. Taubin. An improved algorithm for algebraic curve and surface fitting. In *Fourth International Conference on Computer Vision (ICCV '93)*, pages 658–665, Berlin, Germany, May 1993.

[20] G. Turk and J. O'Brien. Shape transformation using variational implicit functions. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1999)*, pages 335–342, Aug. 1999.

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

[22] G. Yngve and G. Turk. Creating smooth implicit surfaces from polygonal meshes. Technical report, College of Computing, Georgia Institute of Technology, Sept. 1999. Tech Report GIT-GVU-99-42.

[23] T. S. Yoo, B. Morse, K. Subramanian, P. Rheingans, and M. J. Ackerman. Anatomic modeling from unstructured samples using variational implicit surfaces. *Medicine Meets Virtual Reality*, 2001.