Implicit Surfaces Seminar, Spring 2012

University of Waterloo Technical Report CS-2013-08

Khodakhast Bibak, Chun Liu, Hamideh Vosoughpour, Grace Yao,
Zainab AlMeraj, Alex Pytel,
William Cowan and Stephen Mann

September 25, 2013

Abstract

Implicit surfaces are one technique for surface modeling in computer graphics. In this report, we survey some implicit surface papers, present some projects using implicit surfaces, and give our evaluation of this work.

1 Introduction

In Spring 2012, a seminar course in the David R. Cheriton School of Computer Science at the University of Waterloo studied implicit surfaces. While the intent was to study recent modeling techniques in implicit surfaces, we ended up reviewing the basics of implicit surfaces, various ways to render implicit surfaces, as well as some modeling papers for implicit surfaces. In this report, we summarize our findings.

Generally speaking, there are four styles of modeling with implicit surfaces: algebraic surfaces, piecewise algebraic patches, CSG-style, and interpolation/approximation methods. Modeling directly with algebraic surfaces is difficult and will only be discussed in the background section.

There are a several topics that we did not investigate in this study, or only touched upon them when looking at other work. In particular, there are a large number of papers on fast rendering of implicit surfaces; we only touched on a few of these papers, and did not look at work on interval arithmetic as it relates to rendering implicit surfaces. We also did not look at any papers on texture mapping or parametrication of implicit surfaces.

This report begins with a review of some background material on rendering implicit surfaces. Section 3, Section 4, and Section 5 discuss the papers that were covered in lecture (with the sections being a rough attempt at grouping the papers). The remaining sections are the reports on individual student projects, as well as some tests that one of the faculty members made on one implicit scheme.

2 Background

Abel Gomes, Irina Voiculescu, Joaquim Jorge, Bryan Wyvill, and Callum Galbraith. *Implicit Curves and Surfaces: Mathematics, Data Structures and Algorithms*. Springer Verlag, 2009

As background for this course, we worked from the book of Gomes et al. In particular, we were interested in root isolation methods and root finding techniques (to be used for ray tracing and other rendering techniques) and in methods for tessellating an implicit surface, such as Marching Cubes. Chapter 3 of this book discusses root isolation methods; Chapter 5 discusses root finding techniques; and Chapter 6 discusses methods for tesselating an implicit surface.

2.1 Root Finding/Ray Tracing

One way to render an implicit surface is to ray trace it. If the implicit surface is the zero set of a scalar valued function, $S(P) = 0$, and a ray is given by $r(t) = P_0 + t\vec{v}$, then solving $S(r(t)) = 0$ for t gives the intersections of the ray with the implicit surface. Thus, one approach to ray tracing an implicit surface is to do root finding.

For *algebraic* surfaces (i.e., S is a polynomial), we can exploit the properties of polynomials to isolate roots. While Descartes rule of signs gives a bound on the number of real roots of a polynomial, Cauchy's Theorem gives a finite interval in which all real roots must occur, and Sturm sequences give the exact number of roots within an interval. Sturm sequences in

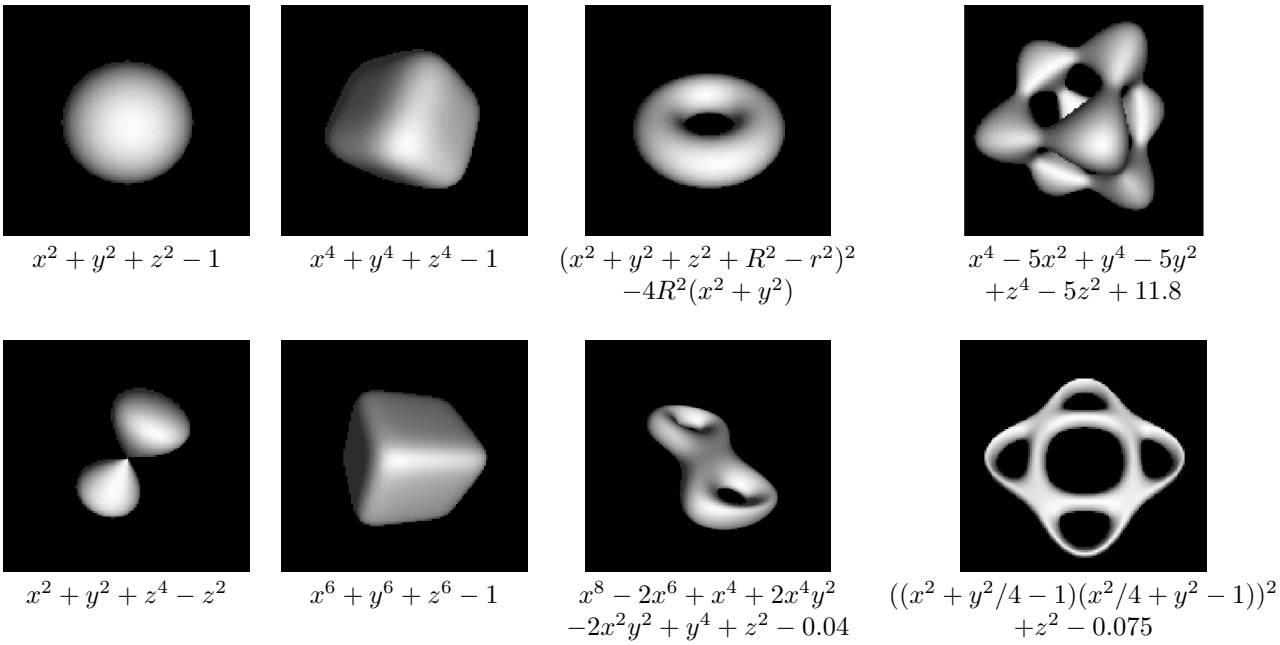


Figure 1: Algebraic surfaces ray traced with a simple ray tracer.

particular gives us a way to recursively narrow down the interval in which roots occur, until we have subdivided the real line into intervals each of which contain either 0 or 1 root.

The root finding methods discussed by Gomes et al. [20] are the 1-point and 2-point methods. The 1-point methods include Newton-Ralphson, Newton's method, and the Secant method, which use derivatives or approximations to the derivatives to find the 0's of a function. These methods have the potential problems of diverging or converging to a local minimum.

With a 2-point method, we start knowing two x values where $f(x_0) < 0 < f(x_1)$ where f is the function whose roots we are trying to find. A simple bisection method will find the root (assuming there is one root within the interval), although for faster convergence the Regula Falsi method may be preferred.

All these methods work reasonably well in the univariate case, but additional complications occur in the multivariate case. Interval arithmetic is another method used for root finding, although we did not investigate it.

We implemented a simple ray tracer in Octave, using Sturm sequences to isolate the roots of an algebraic surface and then using the bisection method to find the root.[1] The ray tracer used a single, fixed point light source, up and to the right of the viewer, and diffuse, gray scale shading was used. The most difficult part of the ray tracing code was representing and manipulating multivariate polynomials. Figure 1 shows some surfaces ray traced in this manner.

2.2 Marching Cubes

As an alternative to ray tracing the surfaces, Marching Cubes [31] is one method of triangulating implicit surfaces.[2] The idea of Marching Cubes is to sample 3D space on a regular grid of points and evaluate the implicit surface at these points (Figure 2 shows a 2D version of Marching Cubes). Then on adjacent grid points, compare the the signs of the values of the implicit function; if they are of opposite sign, then the surface intersects the line segment between the two grid points. We can then use linear interpolation to approximate the zero along this line segment, or use the bisection method if more precision is desired. A piecewise linear approximation to the implicit surface can be made by connecting the zeros on the line segments.

Marching Cubes has a variety of issues. In particular, the grid density needs to be high enough so that you do not miss small features of the implicit surfaces; the step to create the piecewise linear approximation is difficult; and for general implicit surfaces, finding the region where the surface lies is non-trivial.

Much of the complexity of Marching Cubes comes from handling ambiguous cases where there are multiple possibilities for how a surface could slice a cell (Figure 3 shows a 2D example); this topic is further considered in one of the projects (Section 7). We can vastly simplify Marching Cubes if we only want line drawings and if we ignore the ambiguous cases. With this simplified approach, we just construct line drawings through the 2D faces of the cubes. The advantage of this approach is

[1] This root finder was compared to the `roots` routine built-in to Octave and found to give the same results.

[2] Although Marching Cubes is the most well-known of these algorithms, similar ideas appeared earlier in the literature [61].

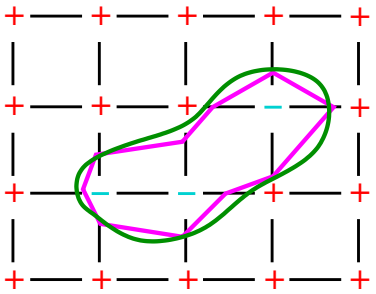


Figure 2: Simple Marching Squares example. Implicit function drawn in green; piecewise linear approximation in magenta.

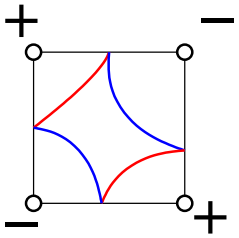


Figure 3: Ambiguities in Marching Cubes. Both the red and blue curves give the same signs at the corners.

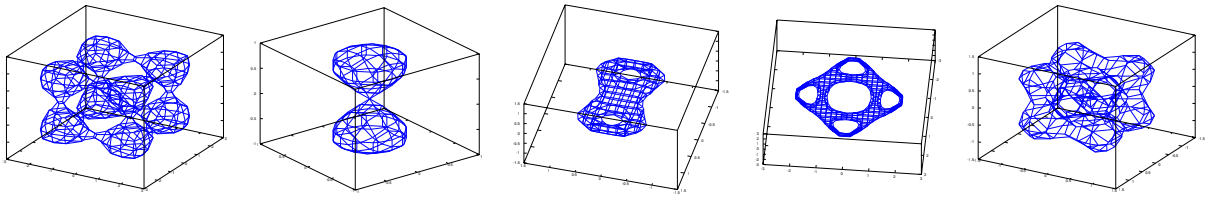


Figure 4: Algebraic surface rendered with a simplified Marching Cubes. The surface on the far right is the BanchoffChmutov B_4; all others appears in Figure 1

that you can get an impression of your surface and interactively manipulate it without much coding effort, although there is a limit to the complexity of the surface that can be visualized this way because of the lack of hidden surface removal.

In addition to a 2D Marching "Squares" algorithm (used to generated Figures 5, 6), a simplified 3D Marching Cubes was implemented. This version of Marching Cubes merely calls the 2D version of Marching Cubes on each face of each cube, giving a line drawing rather than a surface. Figure 4 shows some examples of surfaces drawn this way.

Bloomenthal's version of Marching Cubes [24] was also implemented for one of the projects (Section 10).

Another difficulty with Marching Cubes is finding the region where the surface lies. Given a seed point P, a search can be made in an ever growing region around P until a cell is found that contains a zero, and a grid of cells can be grown around this point (this is roughly the approach taken by Bloomenthal [8]). The search for the initial point on the surface is potentially expensive, so starting with a seed poing on or near the surface greatly speeds the process. However, among other issues, this "growing method" will likely miss disconnected components.

One approach to modeling with implicit surfaces that avoids the problem of finding the region where the surface lies is to start with simple implicit surfaces whose bounding box you know, and then create a complex model by using CSG and other blending techniques; by construction, you then know a reasonable bounding box that contains the surface. For example, the $\max$ and $\min$ functions act as intersection and union, respectively, as illustrated in Figure 5. If smoother boundaries are desired, other blending functions can be used; Figure 6 shows an example of a "union" of two circles that are smoothly blended. Note that these figures were created by using a 2D version of Marching Cubes; the somewhat asymmetrical result of the $\max$ in Figure 5 is a result of using a cell size that is too large and linear interpolation is used to approximate the location of the zero on the Marching Squares edges.

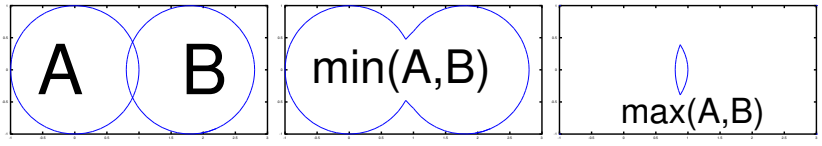


Figure 5: Max and min act as CSG operations on implicit circles.

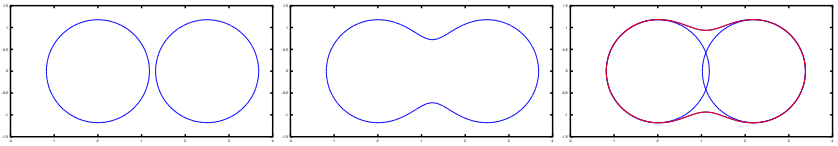


Figure 6: CSG-like blending operations on circles give smoother joins.

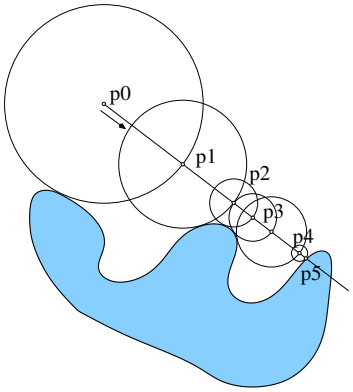


Figure 7: Sphere tracing from point $p0$.

2.3 Sphere Tracing

Another approach to rendering implicit surfaces is *sphere tracing* [23]. The idea in sphere tracing is to trace a ray $P_0 + t\vec{v}$. From P_0, we find the point X_0 on the implicit surface S such that X_0 is the point on S closest to P_0.[3] This guarantees that S does not lie within the sphere of radius $|P_0 - X_0|$ centered at P_0. We then repeat this process from $P_1 = P_0 + |P_0 - X_0|\vec{v}$ until S is within a pre-specified distance ϵ of P_i, at which point we use P_i as the intersection of our ray with S. See Figure 7 for an example.

The key idea in sphere tracing is that the surface S has to have a Lipschitz bound. Such a bound allows us to compute a signed distance function for S. GPU versions of the algorithm exist [16].

3 Papers—CSG Style

Older papers

Discussion

James F. Blinn. A generalization of algebraic surface drawing. *ACM Trans. Graph.*, 1(3):235–256, July 1982

Brian Wyvill, Craig McPheeters, and Geoff Wyvill. Animating *soft* objects. *The Visual Computer*, 2(4):235–242, 1986

The 1982 paper of Blinn [7] was one of the first papers in computer graphics to use implicit surfaces for modeling. The particular modeling problem was the display of molecular models. Rather than a ball-and-stick model, Blinn decided to draw the molecules as blended spheres. The modeling aspects comprise only a few pages of the paper, with the bulk of the paper containing now-standarded rendering details, as well as speed and memory optimizations, with the latter being needed to overcome the limited memory of the 1980's PDP-11 computer used to render the models.

Blinn modeled each atom as an exponential drop-off function from the center of the atom (with the square of the distance from the center being used to avoid a square root computation). For a molecule model, he used a sum of exponential functions for each atom.

Figure 8 shows a 2D reproduction of Figure 3 of Blinn's paper of 3D spheres. The B and R parameters are those appearing in Blinn's paper, with B representing "blobbiness" and R being the radius of the sphere(s).

The 1986 paper of Wyvill et al. [60] builds on the work of Blinn to build and animate *soft* objects. The main idea is that a soft object is represented by a set of spheres, blended in a method similar to Blinn. Animation is achieved by implementing

[3] We don't actually need the closest point on S; a bound on the distance to S is sufficient, although the tighter the bound, the faster the process will converge.

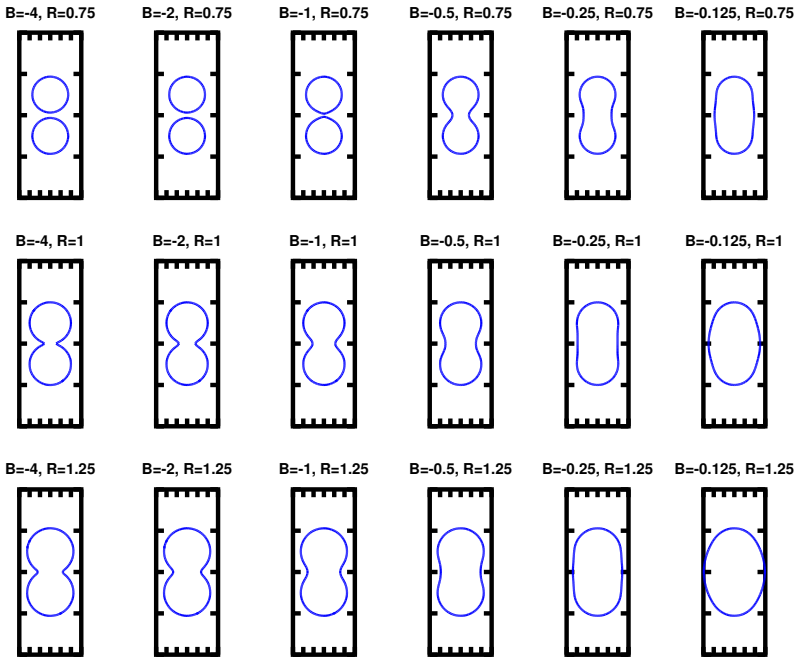


Figure 8: Blending two circles with Blinn's model; centers are at (0,1) and (0,-1); axes are [-1,1] in x and $[-4, 4]$ in y in all cases.

a keyframing system on the spheres, with the blending filling in the gaps between the spheres and having reasonable shape as long as the spheres remain "close enough to each other".

3.1 Skeletal Implicit Modelling — Grace Yao

Andrew Guy and Brian Wyvill. Controlled blending for implicit surfaces. In *Implicit Surfaces '95*, April 1995

Brian Wyvill, Eric Galin, and Andrew Guy. Extending the CSG tree. Warping, blending and boolean operations in an implicit surface modeling system. *Computer Graphics Forum*, 18(2):149–158, June 1999

Loïc Barthe, Brian Wyvill, and Erwin de Groot. Controllable binary CSG operators for "soft objects". *International Journal of Shape Modeling*, 10(2):135–154, 2004

Implicit surfaces have proven to be well suited and efficient for modeling animating and morphing shapes of arbitrary topologies. To allow complex objects to be modelled easily and intuitively, a series of algorithms have proposed for enhancing the range of models that can be defined with a skeletal implicit surface system [22, 59, 4].

Guy and Wyvill [22] introduced a method using a graph to specify blending among objects. The main idea is that objects connected in the graph are blended with each other and objects not connected are deformed against each other with the resulting function of implicit surfaces constrained to be continuous. Their method partially achieved the goal of making blending occurs only when it is desirable. However, this paper failed to consider field values that are not C^1, which results in discontinuities in the normals.

Barthe, Wyvill, and de Groot [4] presented a theoretical background that laid the foundation for the construction of binary Boolean composition operators for bounded implicit primitives. They derived a set of Boolean operators providing union, intersection and difference with or without smooth transition. The new operators integrate accurate point-by-point control of smooth transitions and they generate G^1-continuous potential fields even when sharp transition operators are used. However, the operators are expensive to evaluate and limited to only binary compositions.

Wyvill and Guy [59] introduced the Blob-Tree model to assemble complicated objects from a small number of skeleton based primitives using arbitrary combinations of blending, warping and Boolean operations. A systematic approach to the construction of the Blob-Tree is given that notes what each node stands for and how to define the operators on the nodes to maintain continuity. The Blob-Tree is applied to render the models in two ways: polygonalization and ray tracing by traversing the tree. For polygonalization, Wyvill and Guy illustrated a post-processing method to reproduce the discontinuities that play

an important role in the resulting model. This method iteratively looks for more accurate approximations of the intersection point of the Boolean operations. The field value and normal from each of the contributing nodes at a number of points in space are needed to guide the moving direction from the original guess. These values can be computed by traversing the tree, which is one of the pros of Blob-Tree model. The ray tracing method is based on Lipschitz techniques to determine the roots. The efficiency of their ray tracer is further improved by finding accurate Lipschitz bounds using spatial subdivision and disable non-contributing elements along the ray.

There are several problems with the method proposed by Wyvill and Guy [59]. First, their blending operation can cause unwanted bulging and unwanted shape changes due to large size differences. Guy and Wyvill [22] suggested using the graph as a method of solving the unwanted blending problem by applying a deformation function to each skeletal primitive to manipulate the field function. Wyvill and Guy [59] combined the graph idea with the Blob-Tree model by labelling graph edges so that the type of join between primitives can be specified. In addition, de Groot, Wyvill and van de Wetering [14] introduced a method for solving this problem using locally restricted blending. They added a parameter to each pair of nodes to control the influence of the nodes far way or that need not be considered. However, this method is limited due to the requirement of processing the blending in sequence. (Turk and O'Brien [53] also considered how to make bulge-free blends with implicit surfaces, although their method is in a different setting than that of the Blob-Tree.)

Second, in an articulated model, there is a need to make the movement of joints smooth. Such smooth motion would be needed to generate an animation of an articulated object modeled with the Blob-Tree. However, Wyvill and Guy do not discuss the effects of motion on the joints of a Blob-Tree.

The last concern is that the pre-processing step to find the boundary curve between two nodes of a Boolean operation can be computationally expensive; this boundary is needed to improve the quality of a Marching Cubes solution, for example. One approach to improve the efficiency would be to attach side information to the Blob-Tree nodes to help rendering. Instead of finding the joint during a pre-processing step, the joint points can found during the blending step in the Blob-Tree and marked for later use.

3.2 Complex Skeletal Implicit Surfaces with Levels of Detail — Hamideh Vosoughpour

Aurlien Barbier, Eric Galin, and Samir Akkouche. Complex skeletal implicit surfaces with levels of detail. In *Proceedings of WSCG*, page 2004, 2004

The paper proposed a framework for automatic handling of different levels of detail which is essential in interactive design of implicit surfaces, because the designer needs to see the result in real-time. Also, it would be useful in terms of saving process and time to automatically switch between levels of detail, for example, when a complex surface is going to be shown in small size and most of the details are not actually needed to be computed and rendered. To achieve this goal, the paper introduced a modified version of BlobTree in which the skeletal primitives are categorized into four groups: curves, surfaces, voluminal skeletons, and surfaces of revolution. Modifying the primitives of the BlobTree makes it more powerful in modeling more complex surfaces with fewer primitives. To model the objects in different levels of detail, the paper introduces high level primitives that are modeled with a different number of basic primitives in different levels of details. The proposed high level primitives are: curve based primitives, surface based primitives, and volumes of revolution each of which are composed of several primitives based on the required level of detail.

The paper talks about several ideas and techniques from other studies to propose a framework that handles different levels of detail automatically. Firstly, it introduces the modified BlobTree, which it claims is more powerful. Afterward, the paper discusses levels of details and suggests a technique for automatic transition between them. Finally, it talks about the optimization techniques.

Smooth transition between different levels of detail, which is supposed to be the main contribution of the paper, is discussed only roughly in the paper. The proposed method is only developed and discussed on curve based high level primitives and it cannot be easily generalized into two other categories of high level primitives. In curve based high level primitives, as explained in the paper, if we decrease the contribution of some sample points while switching between levels and finally remove those sample points, then a good transition will be achieved, but if we do the same procedure for surface based high level primitives, some holes will appear in the surface. That is because in curve based primitives the connection between the previous and the next points to the removed point compensate for the lack of the contribution of the removed point and having a point with small weight of contribution is similar to when that point is removed. Hence, the transition is smooth. But this cannot be generalized into the surface situation.

3.3 WarpCurves — Chun Liu

Masamichi Sugihara, Brian Wyvill, and Ryan Schmidt. WarpCurves: A tool for explicit manipulation of implicit surfaces. *Computers & Graphics*, 34(3):282–291, 2010. Shape Modeling International (SMI) 2010

The main contribution of this paper is that it develops a curve-based interface supporting explicit manipulation of implicit surfaces. The WarpCurves technique is achieved by applying curve-based spatial deformation to a model. It deforms the underlying scalar field of an implicit point primitive and renders the iso-surface of the deformed scalar field. Mathematically, it takes a scalar field of the original implicit model f_M as input and generates a deformed scalar field f'_M as output. This task is realized by displacing the sample 3D points p of the original model. The deformed scalar field f'_M is defined as follows: $f'_M(p) = f_M(p + f_{bounding}(p) * D(p))$ where $f_{bounding} : R^3 \rightarrow R$ is bounding field, and $D : R^3 \rightarrow R^3$ is deformation field. The deformation field $D(p)$ returns the displacement of a point p using the variational warp techniques with off-curve constraints. And the bounding field $f_{bounding}(p)$ localizes the deformation effects using convolution with the Cauchy kernel function and Wyvill function adjustments. With this equation, they achieved curve-based spatial deformation.

The main advantage is that they integrate WarpCurves in ShapeShop, which demonstrated an interface for novice users to create 3D models with implicit surfaces. ShapeShop uses BlobTree, a hierarchical data structure that allows arbitrary compositions such as blending warping and Boolean operations.

4 Papers—Piecewise Algebraic

4.1 A-patch rendering of algebraics — Stephen Mann

Chandrajit L. Bajaj, Jindon Chen, and Guoliang Xu. Modeling with cubic A-patches. *ACM Trans. Graph.*, 14(2):103–133, April 1995

Lionel Alberti, Georges Comte, and Bernard Mourrain. Meshing implicit algebraic surfaces: the smooth case. In Schumaker, editor, *Mathematical Methods for CAGD: Tromso 2004*, pages 11–26. Nashboro, 2005

Stephen Mann. Using A-patches to tessellate algebraic curves and surfaces. Technical Report CS-2009-21, University of Waterloo, 2009

A-patches are a form of algebraic patch defined over the tri-variate Bernstein polynomials. The tri-variate Bernstein polynomials are

$$B^n_{\vec{i}}(P) = \binom{n}{\vec{i}} u_0^{i_0} u_1^{i_1} u_2^{i_3} u_3^{i_4},$$

where (u_0, u_1, u_2, u_3) are the Barycentric coordinates of P relative to a tetrahedron $\triangle T_0T_1T_2T_3$ in 3-space, $\vec{i} = (i_0, i_1, i_2, i_3)$ with $i_j \geq 0$ and $i_0 + i_1 + i_2 + i_3 = n$ and

$$\binom{n}{\vec{i}} = \frac{n!}{i_0!i_1!i_2!i_3!}.$$

An algebraic surface in Bernstein form is given as

$$S(P) = \sum_{\vec{i},|\vec{i}|=n} c_{\vec{i}} B^n_{\vec{i}}(P),$$

where $c_{\vec{i}} \in R$. The coefficients can be thought of as being arranged in a tetrahedral array.

A-patches are a form of algebraic surface in Bernstein form with restrictions on their coefficients. Roughly speaking the algebraic surface is in A-patch format if when the coefficients are considered in layers relative to one of the directions, then there is one layer j such that all layers above j have positive sign and all layers below j have negative sign, where the coefficients on layer j may have either sign (Figure 9); see [2] for a more precise definition of the A-patch format.

One of the advantages of being in A-patch format is that you can show that the algebraic surface is single sheeted over the tetrahedron T and further that there is an easy way to tesselate it (see [35] for details). Bajaj et al. used A-patches to construct a piecewise algebraic surface that interpolated positions and normals of a triangular mesh, with the algebraic patches meeting with C^1 continuity [2].

Luk [32] later used A-patches as a means of tessellating a general algebraic surface. The idea is to place a tetrahedron around the region of interest, and convert the algebraic surface to the Bernstein form. If all the coefficients are of one sign, then the surface does not pass through the tetrahedron; if the coefficients are in A-patch format, then you can tessellate the algebraic surface in this region. Otherwise, the tetrahedron is divided into smaller tetrahedron and each subtetrahedron is recursive checked. (The actual algorithm of Luk worked on an Octree, with each bottom level Octree cell being subdivided into tetrahedrons and checked; if any of these tetrahedrons needs subdivision, then the Octree cell is subdivided, and the algorithm proceeds from there.)

At times, Luk's algorithm is slow to converge. Mann [35] found a more relaxed A-patch like condition. Using this relaxed condition, faster convergence was possible.

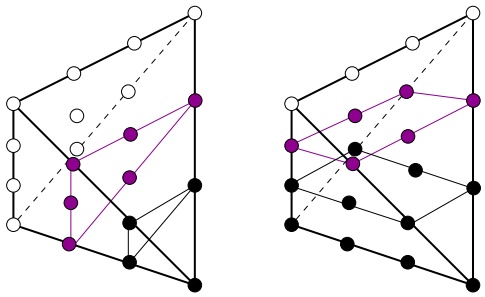


Figure 9: Examples of separating layers in A-Patches; the white points have opposite signs as the black points; the purple points (on the separating layer) may have either sign.

Alberti et al. had earlier developed a similar algorithm for algebraic surfaces represented in tensor-product format. The single-sheeted conditions on the tensor-product coefficients were (essentially) that in one of the three coordinate directions, the deriviatives must be of the same sign. (A later paper [15] claims to improve on this work, but we did not read it.)

A direct comparision of the Alberti algorithm to the Luk algorithm was not possible without implementing both of them. In particular, it is unclear whether one set of single-sheeted conditions on the coefficients (required to tessellate) would result in less subdivision, or if either type of condition could be used in the other technique.

5 Papers—Interpolation and Approximation

5.1 Implicit Manifolds, Triangulations, and Dynamics — Alex Pytel

L. Velho, J. de Miranda Gomes, and D. Terzopoulos. Implicit Manifolds, Triangulations, and Dynamics. *Journal of Neural, Parallel, and Scientific Computation*, 5(1–2):103–120, 1997. Special issue on CAGD

One class of methods for polygonizing an implicit surface involves the following three steps: decomposition of the ambient n-space into simplices, identification of a simplicial complex that intersects the surface, and triangulation of the surface inside each n-simplex that it intersects. This paper shows how the quality of triangulations resulting from such a procedure can be improved by constructing a mass–spring system out of the vertices and edges of the intersecting simplicial complex and letting a dynamics simulation reshape it.

The main dynamics equation that governs the position x_i of each particle of the mass-spring system is based on Newton's 2[nd] law:

$$m\frac{d^2x_i}{dt^2} + \gamma\frac{dx_i}{dt} + n_i = 0, \tag{1}$$

where γ is a dampening constant and n_i is the net force acting on each particle. The net force is a sum of three types of forces: spring forces exerted by the particle's neighbors, a force defined inside an ϵ–neighborhood of the implicit surface that repels particles away from the surface, and a force defined outside the ϵ–neighborhood that attracts particles to the surface. The simulation uses Euler integration to solve for x_i of an equilibrium state.

Simulations of particle dynamics have found many uses as versatile modeling and visualization tools. For example, Szeliski et al. [50] describe a system for modeling surfaces based on the motion of particles and the orientation of their local coordinate systems. Levet et al. [29] use particles to sample implicit surfaces. While particle simulations tend to have a lot in common, the formulation of particle motion used by Velho et al. is somewhat unusual for two reasons. First, the interactions between the particles are defined by the connectivity of the graph representing the simplicial complex that intersects the provided implicit surface. It is more typical for particle simulations to have particles exert forces on each other based on proximity instead. Second, the particles of Velho et al. are free to move in n-space while under the influence of various forces. Other systems impose additional constraints on the particles. In particular, Szeliski et al. [50] constrain the rotational dynamics of their particles, while the particles in the scheme of Levet et al. [29] move on the implicit surface that they are sampling.

One weakness of the paper by Velho et al. is that the authors do not discuss possible shortcomings of their system resulting from underconstrained particle motion. The examples given in the paper do not appear to exhibit any anomalous particle positions, but the examples are limited to spheres, cylinders, and tori. These surfaces do not possess areas of high curvature that could cause problems for underconstrained particles.

Another weakness of the paper concerns the concept of subordinate triangulation that the authors use to justify the improvements to the quality of triangles in the final triangulation produced with their algorithm. A subordinate triangulation τ is a

simplicial complex with the following special relationship to a given implicit surface M: for each simplex δ of τ that intersects M there is a point $p \in M \cap \delta$ close to the barycenter of δ such that the tangent space of M at p is close to the support hyperplane of one of the faces of δ.

This property seems to suggest that δ has one vertex on one side of M and three on the other. Additionally, assuming that the initial tetrahedralization of space is sufficiently fine, the property seems to imply that M inside δ is close to being parallel with one of its faces while also passing through the barycenter of δ. Intuitively, the polygonization of M inside δ in this situation should result in better triangles than if M intersected δ in an oblique way. However, Velho et al. make no provisions to ensure that these observations hold in the situation when a tetrahedron of the initial complex resulting from the regular decomposition of ambient space intersects M by having two vertices on each side of it. The mass-spring simulation is not guaranteed to pull one of the vertices to the other side through the repelling ϵ–neighborhood around M. Furthermore, the authors do not provide any proof of their mass–spring dynamics reshaping the starting simplicial complex into a subordinate triangulation.

5.2 Modeling with Implicit Surfaces That Interpolate — Khodakhast Bibak

Vladimir Savchenko, Er A. Pasko, Oleg G. Okunev, and Tosiyasu L. Kunii. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum*, 14:181–188, 1995

Greg Turk and James F. O'Brien. Shape transformation using variational implicit functions. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 335–342, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co

Greg Turk and James F. O'Brien. Modelling with implicit surfaces that interpolate. *ACM Trans. Graph.*, 21(4):855–873, 2002

Turk and O'Brien [53] gave an approach for modeling interpolating implicit surfaces. Their method is based on applying variational techniques and the radial basis function (RBF) $\phi(\mathbf{x}) = |\mathbf{x}|^3$. Their interpolation function can be written in the form

$$f(x) = n \sum_{j=1}^{k} w_j \phi(\mathbf{x} - \mathbf{c}_j) + P(\mathbf{x}), \tag{2}$$

where $\mathbf{c}_j$ are the points to interpolate and $P(\mathbf{x})$ is a linear polynomial (Turk and O'Brien only say that P contains linear terms missing from the radial basis function; Jin et al. [25] claim that it ensures affine invariance; regardless, P ensures that the linear system is solvable [57]). Given interpolation data $h_i = f(\mathbf{c}_i)$, (2) yields a linear system of equations that can be solved for the unknown w_i and the coefficients of P. Further, the system is positive definite and guaranteed to have a solution.

In their 2002 paper, Turk and O'Brien introduced interior, exterior, and normal constraints. By a *boundary constraint* we mean points located on the surface. Positive-valued constraints were used to specify *interior constraints*, which are to specify a region that is inside the surface. Likewise, negative-valued constraints were used to specify *exterior constraints*, which are to prevent the surface from overshooting into regions where the model should not exist. Also, *normal constraints* are placed very close to the boundary constraints. In fact, given a control point at (x, y, z), with the normal (n_x, n_y, n_z), a boundary constraint is created at (x, y, z) and a normal constraint is created at $(x - kn_x, y - kn_y, z - kn_z)$. Note that at least one interior, exterior, or normal constraint is required to avoid the trivial solution of $w_i = 0$.

Turk and O'Brien also discuss using the method of Witkin and Heckbert [58] to build an interactive modeling system where the user directly moves the points on the surface, although since Turk and O'Brien resolve the linear system each time a point is moved, it seems unlikely that their system would be interactive for large models. They also point out that their method can be used to blend multiple models by discarding those points of any model that lies within the implicit surface of any other model; the remaining points are interpolated using their technique. They also discuss using a Marching Cubes algorithm to render their surfaces, as well as using Sphere Tracing to ray trace their surfaces. For Sphere Tracing, they need a Lipschitz bound on their model over a reasonable region containing their model; they numerically calculate this constant by sampling the gradient of their surface over the relevant region, and using the maximum value of the gradient as the approximation to a Lipschitz constant. They note that in some cases their heuristic could fail; we further note that since their bound is unlikely to be tight for most of the space, then the sphere tracing is likely to be slow.

This 2002 paper of Turk and O'Brien is related to a paper of Savchenko et al. [46]. In Savnchenko et al.'s method, the implicit surface is reconstructed by introducing a *carrier solid*, which in the simplest case can be a sphere. First, the radii of the carrier functions are calculated at all given points. The reconstruction problem is now shifted to finding a volume spline function that interpolates all values of radii. This spline is found by solving a linear system of equations. While the system is symmetric, it is not positive definite. Savchenko et al.'s method has some drawbacks. For example, volume interpolation requires heavy

calculations (so the surface can neither be efficiently reconstructed nor efficiently evaluated). Also how to determine an adopted carrier solid is a challenging problem.

The 2002 Turk-O'Brien paper also builds on an earlier Turk-O'Brien [52] which used the variational technique in shape transformation. Interpolating implicit surfaces are constructed in a similar way, using (2) but with thin-plate splines radial basis functions instead of $|\mathbf{x}|^3$ (in this earlier paper, they also do not use interior/exterior/normal constraints). They then blend between two implicit surface as follows. Given two shapes A and B, 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, create a 4D constraint at the position $(x, y, z, 0)$. Similarly, the constraints of shape B are placed in the $w = 1$ subspace, so each of these constraints is of the form $(x, y, z, 1)$. Now, look at the implicit function formed by these 4D constraints (from both shapes). A three dimensional slice through this function at the value $w = 0$ yields shape A, and a slice at $w = 1$ gives shape B. Slices between these two locations, $0 < w < 1$, result in shapes that are intermediate between the two original shapes. (Tests of the Turk-O'Brien scheme appear in a companion technical report [36].)

6 Projects

The remaining sections are shortened forms of the project reports for the course. Each student wrote their own report, and the instructors made a pass through the reports, cleaning up the writing and deleting less relevant material. The projects could either be an implementation project, or a survey project, and possibly a combination of both.

7 Marching Cubes and Marching Squares: Face Ambiguity Chun Liu

There is an interest in approximating implicit surfaces with polygonal meshes [8] primarily to take advantage of graphics rendering systems that can render triangles faster than other primitives. The Marching Cubes algorithm is one technique that converts implicit surfaces into polygonal meshes. The purpose of this project is to investigate face ambiguity issues in the Marching Cube algorithms. I will illustrate the face ambiguity problems using the Marching Square algorithm in this project.

7.1 Marching Cubes

Marching Cube algorithm was originally proposed by Lorensen and Cline [31], although Wyvill et al. [61] proposed a similar algorithm earlier. Marching cubes is widely used in scientific visualization, including medical imaging [31, 51], biomechanical modeling [55], and constructing surface from contour data [27]. In Lorensen and Cline's paper, the algorithm was designed for the visualization of multiple 2D slices of the human body taken from computed tomography (CT), magnetic resonance (MR), and single-photon emission computed tomography (SPECT). Marching Cubes was used to generate the connectivity between two slices. A grid of cubes is formed in between slices. Each cube is processed individually. Marching cubes determines how the unknown data surface intersects a cube. There are twelve edges on a cube. The algorithm detects which edges the data surface intersects. Using the intersection points, we interpolate the data surface by connecting intersection points according to a topology assumption. Then inside a cube, interpolating polygonal meshes are formed. After processing a cube, the procedure moves (or marches) to the next cube in the grid and repeats the process until no more slices remain.

In summary, there are three steps to polygonize data surface in the Marching Cube Algorithm:

1. Partitioning in between inter-slices into a grid of cubes.
2. Determining how the data surface intersects each cube's edges, and
3. Connecting intersection points and forming triangular meshes according to topological configurations.

For determining the topological configuration in a cube, an 8-bit code can be used to encode the state of each corner vertex C. If $f(C) > 0$, the corresponding bit is set to 1. If $f(C) < 0$, the corresponding bit is set to 0. I will discuss case $f(C) = 0$ in Section 7.5.2. The 8-bit code is used as an index into a look-up table of all possible topological configurations of Marching Cubes.

Since there are eight vertices in each cube, there are $2^8 = 256$ possible ways in how a surface intersects a cube. Using symmetry and rotational operations, the 256 cases can be reduced into 14 patterns, some of which are shown in Figure 10.

7.2 Marching Squares

Marching Squares is a 2-Dimensional version of the Marching Cubes algorithm. In Marching Squares, the algorithm polygonizes a 2D curve instead of a 3D surface. Similar to Marching Cubes, there are three main steps to polygonize a 2D implicit curve with the Marching Square Algorithm:

1. Partitioning a region of interest into a grid of squares.

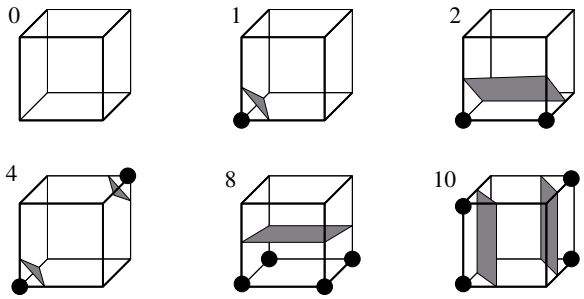


Figure 10: 6 of the 14 patterns on how a surface intersects a cube. Based on a figure of Lorensen and Cline [31]. Cases 4 and 10 in are ambiguous. Case 4 is the case that Chernyaev handled.

2. Determining how the curve intersects each edge of square, and
3. Connecting intersection points according to topological configurations.

For determining the topological configuration in a square, a 4-bit code can be used to encode the state of each corner vertex C. If $f(C) > 0$, the corresponding bit is set to 1. If $f(C) < 0$, the corresponding bit is set to 0. I discuss case $f(C) = 0$ further in Section 7.5.2. The 4-bit code is used as an index into a look-up table of all possible topological configurations of Marching Squares.

There are four vertices in each square. As a result, there are $2^4 = 16$ possible cases on how a curve can intersect a square. Using symmetry, these can be reduced from 16 to 8 cases. And using rotational symmetry, the cases can be reduced into four unique patterns (see Figure 11).

I re-implemented the Marching Square Algorithm of "Processing" [18] (an integrated development environment). Figure 12 shows an example of my re-implemented Marching Square Algorithm on the function $f(P) = x^2 + y^2 - 4.7$, where I avoided the case $f(C) = 0$ at the grid points. I implemented the Marching Squares algorithm as follows:

Step 1: Partition a region of interests into a grid of squares. In my implementation, the region of interests is scaled to fill the display area of the screen. As a result, the Euclidean width of square differs in the examples, even though they may appear the same size in the images.

Step 2: Determine how the curve intersects each square's edges. Evaluate the value $f(C)$ at the four corner vertices C of a square. If $f(C) > 0$, I display a '+' sign on that position. If $f(C) < 0$, I display a '$-$'. If $f(C) = 0$, I display 'o' to represent a vertex sitting on the implicit curve. See Figure 19 (b) for an example with all three cases. The states of four vertices are encoded in a 4-bit code. Using the 4-bit code and according to the topological configurations in Figure 11, I decide which edges are intersect with the curve. I find the curve-edge intersection point on an edge using the Secant Method.

Step 3: Connect (linearly interpolate) the intersection points according to the topological configurations in Figure 11.

Step 4: March to the next square and repeat step 2 and step 3 until there are no more squares to process.

An example of applying Marching Squares to a high degree implicit cure is shown in Figure 19 (a).

7.3 Face Ambiguity

Marching Squares does not guarantee that the topological configuration is correct. In Figure 11, I showed the 16 Marching Square topological configurations, and that these 16 cases can be reduced to four patterns. Among these four patterns, a face ambiguity appears in Pattern 3. The ambiguity occurs because we do not know how to connect pairs of intersection points on the edge of square. In Pattern 3, there are two ways to connect them, which are shown in Figure 11 in different colours: Either we connect the intersection points using the green lines or we connect them using the purple lines but not both. The chosen connected points forms the shape of a hyperbola. Notice that this ambiguity happens when positive and negative vertices are diagonally opposed. An example of the Marching Squares face ambiguity is shown in Figure 13 (a), and a high degree case in Figure 19 (c).

Likewise, a similar issue happens in the 3D case. Marching Cubes also does not have topological guarantees. Face ambiguities happen in configuration 3, 4, 6, 7, 10, 12, and 13 for Marching Cubes, with cases 4 and 10 shown in Figure 10. We simply do not know how to connect the intersection points on the edges of the cube. If points are incorrectly (or at least, inconsistently) connected, "holes" can appear on the polygonized surface mesh [39, 13, 20, 40, 37].

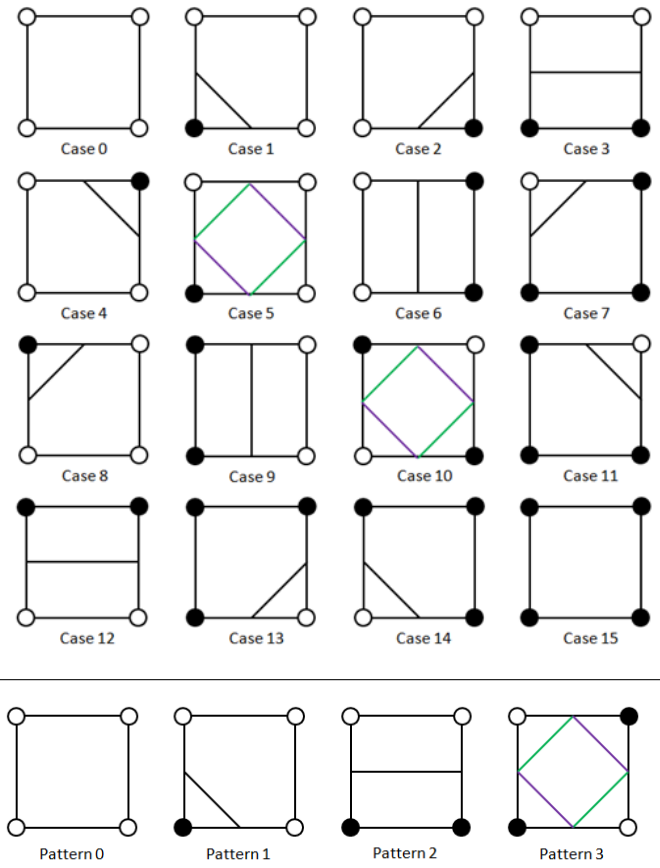


Figure 11: 16 cases and four summarized patterns on how a curve intersects a square

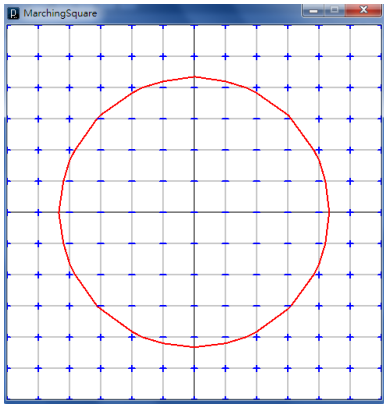


Figure 12: Applying Marching Squares to circle. The width of each square is 0.5 units. Circle: $f(P) = x^2 + y^2 - 4.7$

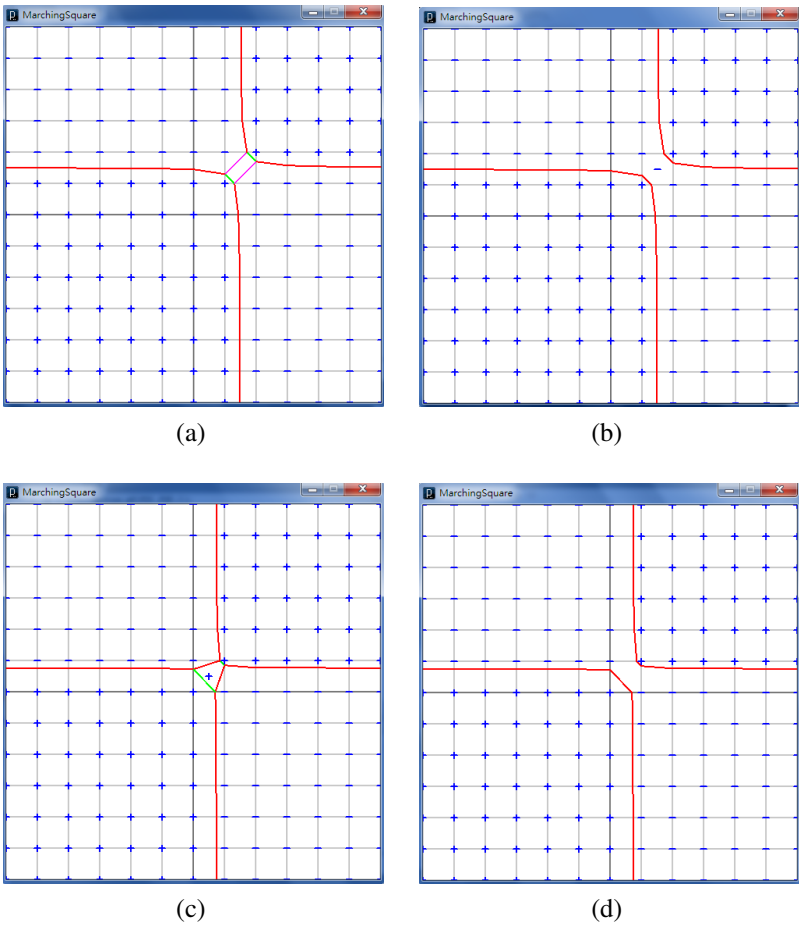


Figure 13: Rectangular Hyperbola: $f(P) = (x - 1.5)(y - 1.5) - 0.1$. (a) Face Ambiguity appears when we process Pattern 3 square. The width of each square is 1.0 unit. (b) Four triangles technique when we process Pattern 3 square. The width of each square is 1.0 unit. (c) Case when four triangles technique fails. The width of each square is 2.0 units. (d) Asymptotic Decider in Pattern 3 square. The width of each square is 2.0 units.

7.4 Face Disambiguation

There are a couple of techniques that try to solve the face ambiguity problem in Marching Cube. In this project, I will describe three face disambiguation techniques—Wyvill et al.'s four triangles [61], Nielson and Hamann's Asymptotic Decider [39], and Chernyaev's Marching Cube 33 [13]. I used my Marching Squares algorithm to briefly test these face disambiguation techniques.

7.4.1 Evaluate the Vertex in Square Center

Wyvill et al. [61] proposed a technique to avoid face ambiguity. The concept is simple. Basically, we evaluate an extra point in a Pattern 3 square. Wyvill et al. chose evaluating the vertex P at the center of the square because the value at the centre of a square is approximately the average function value of four corner vertices. If the function value $f(P)$ is greater than 0 (a threshold value, they called "magic"), they defined this point as a "hot" point. Otherwise, if $f(P)$ is less or equal than 0, they called this point "cold". In a Pattern 3 Square, there are two "hot" points and two "cold" points on the corner vertex. The center point can be either "hot" or "cold". We separate these five vertices as "hot" or "cold" using two lines. For example, if the center vertex is evaluated as "cold", then we group two "cold" corners with center vertex and we draw two lines to separate the "hot" points. One of my test results using their method is shown in Figure 13 (b). In this case, the configuration is correct.

Unfortunately, although this technique solves many cases with the face ambiguity problem, it does not solve all cases [40, 20, 37]. In other words, this technique does not really solve the face ambiguity problem. For as a counter-example, consider a situation in Figure 13 (c). Although the "hot" points were together, the interpolation is still wrong. The correct interpolation is shown in green lines. This technique fails because they assume that the value at the centre of a square is approximately the average function value of four corner vertices, but this assumption is not enough to guarantee correctness. This technique is a fast and simple way to solve many cases, but it is not recommended to guarantee correctness.

7.4.2 Asymptotic Decider

The second face disambiguation technique is called asymptotic decider. This technique was proposed by Nielson and Hamann [39]. Their original purpose was to try to solve more general ambiguities in the Marching Cube Algorithm, including face ambiguities and interior ambiguities. In this paper, I focused on testing face ambiguities in Marching Squares.

In Asymptotic Decider, they assume that the Pattern 3 topological configuration always forms the shape of a hyperbola, and the contour curves can be bilinearly interpolated. The function,

$$B(s,t) = (1-s, s) \begin{pmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{pmatrix} \begin{pmatrix} 1-t \\ t \end{pmatrix}$$

bilinear interpolates the contour curves of hyperbolas, where (s,t) is an arbitrary point in the square and $B_{00}, B_{01}, B_{10}, B_{11}$ are the four corner vertices of square. The interpolated value of the two asymptotes' intersection point $(s_\alpha\, t_\alpha)$ is

$$B(s_\alpha, t_\alpha) = \frac{B_{00}B_{11} + B_{10}B_{01}}{B_{00} + B_{11} - B_{10} - B_{01}}.$$

If $B(s_\alpha, t_\alpha) < 0$, then the two asymptotes' intersection point (s_α, t_α) is negative. Then we group the negative corners and the point (s_α, t_α) together and draw lines separating the two positive corners, which is similar to Wyvill et al.'s four triangle methods. One of the tests result is shown in Figure 13 (d).

Others have found that asymptotic decider gives a topological guarantee and works beautifully [13, 20, 37], but when I tried it on the case of two parallel lines, the asymptotic decider fails (see Figure 14). The red lines are how asymptotic decider interpolated, while the green lines are the actual implicit curves. This problem occurs because of the assumption that the Pattern 3 topological configuration always forms a hyperbolic shape. Ning [40] notes that "the decision is correct if bilinear interpolation is an accurate estimate of the actual function's behavior between samples." In Figure 14, the bilinear interpolation is an incorrect estimation of the actual function's behavior. However, while the reconstruction (of the lines) is incorrect, the result given by asymptotic decider is consistent (i.e., each connected component has no holes). Another problem case is shown in Figure 19 (d).

7.4.3 Marching Cube 33

Marching Cube 33 is a face disambiguation technique proposed by Chernyaev [13]. This technique uses Asymptotic Decider, but in additional generates 33 topological configuration patterns for the Marching Cubes algorithm. The difference with Nielson and Hamann's is that instead of evaluating $B(s_\alpha\, t_\alpha) < 0$, Chernyaev compares $B_{00}B_{11}$ and $B_{10}B_{01}$. If $B_{00}B_{11} > B_{10}B_{01}$, we connect B_{00} and B_{11} together and draw lines separating B_{10}, B_{01}; otherwise, we connect B_{10} and B_{01} together and separate $B_{00}B_{11}$.

There is another difference between Nielson's and Chernyaev's. Nielson-Hamann considered ambiguities that happen in configuration 3, 6, 7, 10, 12, and 13, while Chernyaey noticed that configuration 4 is also ambiguous. Configuration 4 forms an

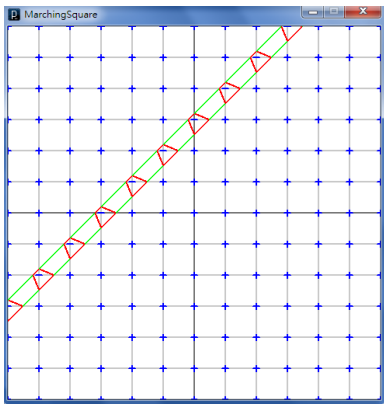


Figure 14: Asymptotic Decider fails in my implementation. The width of each square is 0.5 units. Two parallel lines: $f(P) = (y - (x - 1.2))(y - (x - 1.2) - 0.5)$

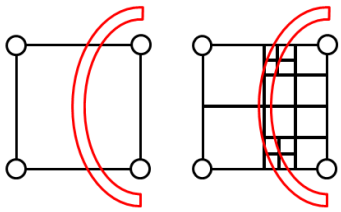


Figure 15: Multiple intersections on an edge of a square. Most of the time, repeatedly subdividing the square will result in a case that Marching Square can handle.

interior ambiguity instead of face ambiguity. In this project, I focused on face ambiguity. In particular, note that configuration 4 can not occur in Marching Squares.

7.5 Discussion

I encountered several concerns while implementing the Marching Squares algorithm and testing several cases. In particular, what happens if there are two curve-intersection points on the same edge? How do we handle the situation when the values of square's corners are zero? What problems occur when the implicit curve has self-intersections? I will discuss each concern in the rest of this section.

7.5.1 Two Intersections on the Same Edge

It is possible that an implicit curve intersects an edge of square twice. Such a case is shown in Figure 15, left. Marching Squares does not have a topological configuration for multiple intersections on the same edge. To solve this issue, most of the time, we keep subdividing the square until we reach a case that Marching Square Algorithm can handle [20, 8, 40]. In my implementation of Marching Squares, I did not handle the case of multiple intersections on an edge. As shown in figures 19 (b) and (c), the curve cannot be interpolated near the center.

7.5.2 Zeros at the Corner Vertices

A zero value on a square corner does not seem to affect how the Marching Squares algorithm behaves. Zero vertices do not change the topological patterns of Marching Square. In Figure 16, each zero case pattern can be summarized as a corresponding pattern in Marching Squares. Or we can say that a zero case is a sub-case of the four topological patterns. An example with a zero case is shown in Figure 19 (b).

Note that if two zero vertices are neighbors, we consider this case as two intersections on the same edge. As discussed before, we need to subdivide such an edge. An example is shown in Figure 17.

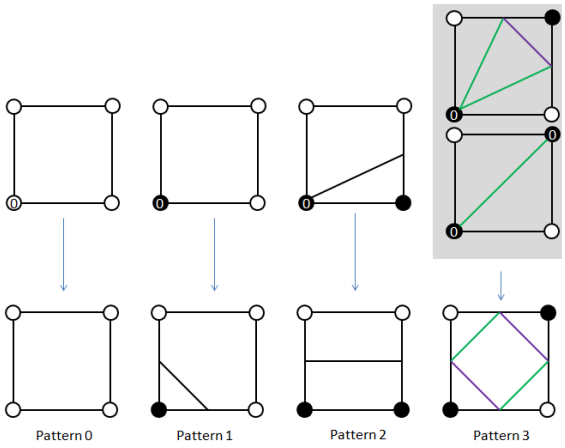


Figure 16: Each Zero case pattern can be summarized as a Marching Square pattern.

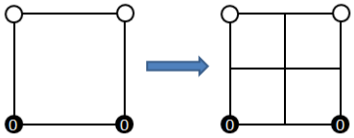


Figure 17: If two zero vertices are neighbors, we consider this case as two intersections on the same edge. As we discussed before, we need to subdivide.

7.5.3 Self-Intersections

Curve self-intersection probably is the worst case for the Marching Squares algorithm. This is illustrated in Figure 18. The correct interpolated curve cannot be generated by the above Marching Squares technique. The correct interpolation is neither green lines nor purple lines in Figure 18. The correct result is the yellow lines in Figure 18, which were are hand-coded for this example.

To fix this problem, there are some techniques exist that trying to handle self-intersections surfaces, but I did not investigate them further, since I was focusing on face ambiguity and face disambiguation.

7.6 Conclusion

In this project, I investigated face ambiguity issues in the Marching Cubes algorithm by analyzing the Marching Squares algorithm. I illustrated face ambiguity problems using my re-implementation of Marching Squares. And I discussed three techniques to solve face ambiguity in Marching Squares—Wyvill et al.'s four triangles method, Nielson and Hamann's Asymptotic Decider, and Chernyaev's Marching Cube 33, although Chenyaev's method does not differ from Nielson and Hamann's method on Marching Squares. I tested these three techniques with my version of the Marching Squares algorithm, and I generated several failure cases.

There were several issues I had with implementing a Marching Squares algorithm. And this was only a 2D case. I believe that ambiguity problems and other concerns would be much worst for Marching Cubes.

I believe that Marching Square and Marching Cube Algorithms are better in visualizing surfaces in a fast and simpler way. However, if our priority is to visualize surfaces without errors, Marching Cube and Marching Square might not be a great choice.

8 An Improvement on Curve-Based High-Level Primitive Hamideh Vosoughpour

A curve-based high-level primitive is one of a few high-level primitive categories introduced in [3] to handle different levels of details automatically. This category of high-level primitives models thick 3D curves at different levels of details. In this project, a curve-based high-level primitives is implemented, and it is shown that the models created by the method in the paper have

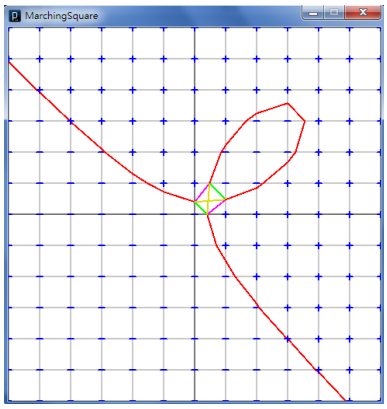
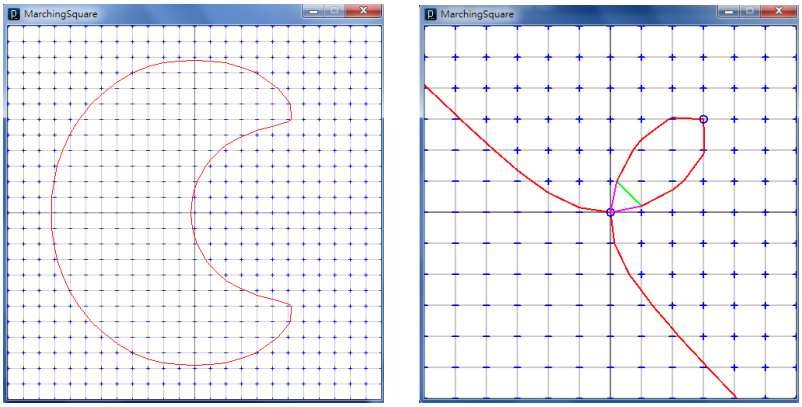
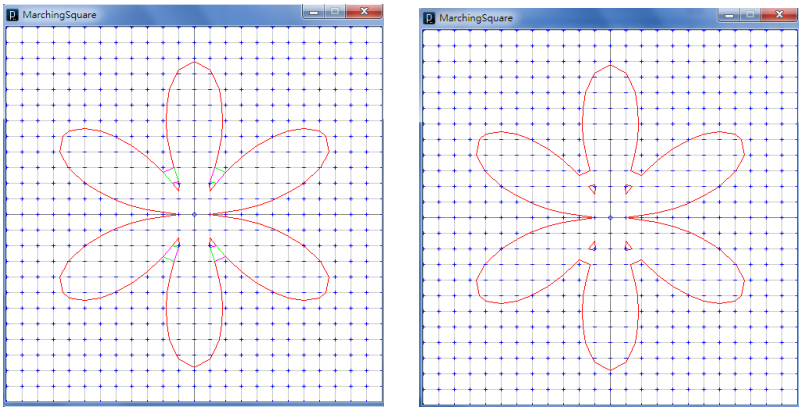


Figure 18: Marching Square fails on a self-intersecting implicit curve. The width of each square is 0.5 units. Folium of Descartes: $f(P) = (x - 0.2)^3 + (y - 0.2)^3(x - 0.2)(y - 0.2)$



(a) (b)



(c) (d)

Figure 19: Additional examples. **(a)** Applying Marching Square on a high degree implicit curve. The width of each square is 0.1 units. Moon: $f(P) = (x - y^2 - x^2 + 1)^4 + (y^2 + x^2)^4 - 0.9$; **(b)** Applying Marching Square on zero case (With Face Ambiguity). The width of each square is 0.5 units. Folium of Descartes: $f(P) = x^3 + y^3 - 3xy$; **(c)** Applying Marching Square on a high degree implicit curve (With Face Ambiguity). The width of each square is 0.5 units. Two Trifolium: $f(P) = (y^2 + x^2)^4 - 24(y^3 - 3x^2y)^2$; **(d)** A fail case when applying Marching Square on a high degree implicit curve (With Asymptotic Decider). The width of each square is 0.5 units. Two Trifolium: $f(P) = (y^2 + x^2)^4 - 24(y^3 - 3x^2y)^2$

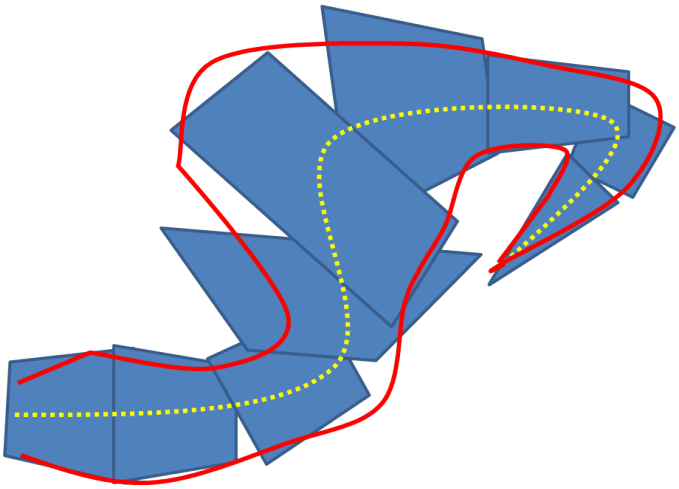


Figure 20: A curve modeled by a number of cones

some unwanted creases in the final surface. I proposed a modification to improve the model and used this category of primitives to model an object.

The model proposed in [3] for curve-based primitives is based on cutting the curve into a number of smaller curves and approximating each sub-curve by a truncated cone. Thicknesses of the curve at cut points determine the radii of cones where they are truncated.

At a higher level of detail the curve is cut into more segments which leads to a more accurate but more complex model. Figure 20 shows a rough, 2D example of a curve modeled with a number of cones.

In this project, the curve is given in a parametric form with parameter p:

$$x = f_x(p), \quad y = f_y(p), \quad z = f_z(p),$$

where $p \in [p_0, p_1]$.

The thickness of the curve is also assumed to be a function of p, so we have $t = t(p)$.

Using this form of equations we can cut the 3D curve by cutting the parameter interval. For a given level of detail α, the interval $[p_0, p_1]$ is divided into α equal sized subintervals, and the curve is approximated with a truncated cone in each of these small intervals. For a small interval $[p_s, p_e]$, the start and end points of the curve are indicated by $(x_s\ y_s\ z_s)$ and $(x_e\ y_e\ z_e)$, respectively, where

$$\begin{aligned} (x_s\ y_s\ z_s) &= (f_x(p_s)\ f_y(p_s)\ f_z(p_s)), \\ (x_e\ y_e\ z_e) &= (f_x(p_e)\ f_y(p_e)\ f_z(p_e)). \end{aligned}$$

The thickness of the curve at p_s and p_e are indicated by t_s and t_e, respectively. Figure 21 shows the truncated cone that should be rendered for the interval $[p_s, p_e]$.

The following equation for a truncated cone for $p \in [p_s, p_e]$ is used:

$$\frac{x^2 + y^2}{c^2} = (z - z_0)^2, \ c = \frac{t_s}{h}.$$

Figure 22 shows the cone for this equation. The axis of cone is the z-axis, and the base of the cone located on the xy-plane with radius t_s. The part of the cone we need to render is for $0 \le z \le l$, where $l = p_e - p_s$.

The matrix representation for the cone is

$$X^T A X = 0,$$

where

$$X = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \qquad A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -c^2 & 0 \\ 0 & 0 & 2z_0c^2 & -z_0^2 \end{bmatrix}.$$

The condition of $0 \le z \le l$ can be interpreted as forcing the cone equation to be between two planes: $z = 0$ and $z = l$. The plane equation has also a matrix form:

$$X^T P = 0,$$

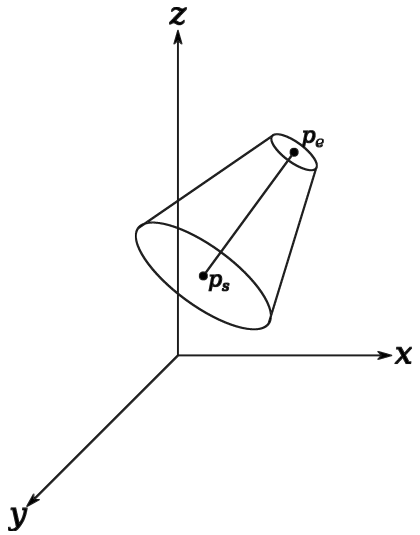


Figure 21: The truncated cone to model the 3D curve at $[p_s, p_e]$

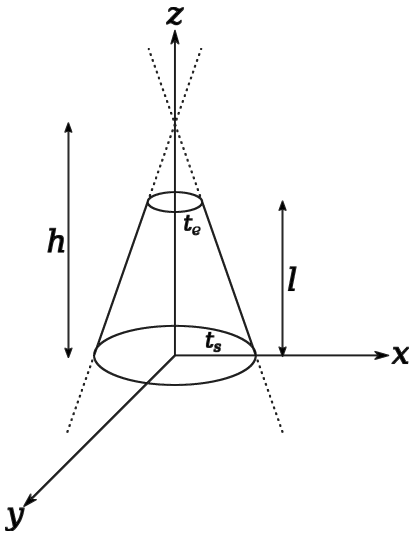


Figure 22: The cone in its normal configuration

where P is a 4×1 matrix. For the planes of $z = 0$ and $z = l$ we have:

$$P_1 = P_{z=0} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad P_2 = P_{z=l} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ -l \end{bmatrix}.$$

A point X lies between P_1 and P_2 if $X^T P_1$ and $X^T P_2$ have different signs.

To get a truncated plane of arbitrary location and orientation, we need to rotate the cone equation such that cone axis fits on the vector connecting $(x_s\ y_s\ z_s)$ to $(x_e\ y_e\ z_e)$; call this vector d. We also need to translate the shape such that the origin is located on p_s. The rotation consists of a rotation around the x-axis by α and then a rotation around y-axis by β. The angles α and β are shown in Figure 23. In this figure, the bold vector shows the vector d, which is going to be the new cone-axis. Therefore, we have

$$\begin{aligned} \sin(\alpha) &= \frac{-d_y}{\sqrt{d_x^2 + d_y^2 + d_z^2}}, \\ \cos(\alpha) &= \frac{\sqrt{d_x^2 + d_z^2}}{\sqrt{d_x^2 + d_y^2 + d_z^2}}, \\ \sin(\beta) &= \frac{d_x}{\sqrt{d_x^2 + d_z^2}}, \\ \cos(\beta) &= \frac{d_z}{\sqrt{d_x^2 + d_z^2}}. \end{aligned}$$

The transformation is the combination of a rotation by α around the x-axis, R_x, a rotation by β around the y-axis, R_y, and a translation, T that moves the origin to $(x_s\ y_s\ z_s)$:

$$M = T \times R_y \times R_x.$$

We need to change the equations of cones and truncating planes such that any point X maps to $X' = MX$ in transformed system. Setting $X' = MX$ gives $X = M^{-1}X'$, and replacing X in $X^T A X = 0$ we have:

$$\begin{aligned} (M^{-1}X')^T A (M^{-1}X') &= 0. \\ \Rightarrow X' T {M^{-1}}^T A M^{-1} X' &= 0. \end{aligned}$$

Therefore, the matrix for new quadratic equation is $A' = {M^{-1}}^T A M^{-1}$, and the truncating planes are $P_s' = {M^{-1}}^T P_s$ and $P_e' = {M^{-1}}^T P_e$.

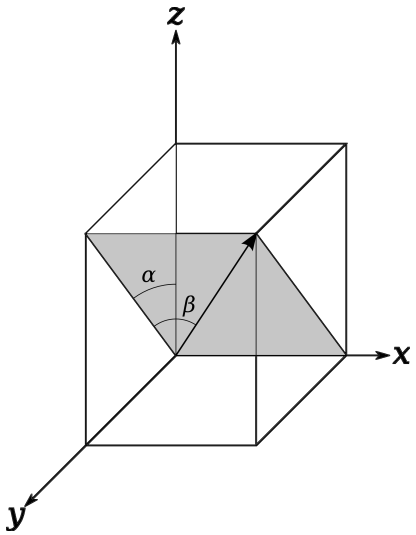


Figure 23: The rotations needed to fit z-axis on vector d

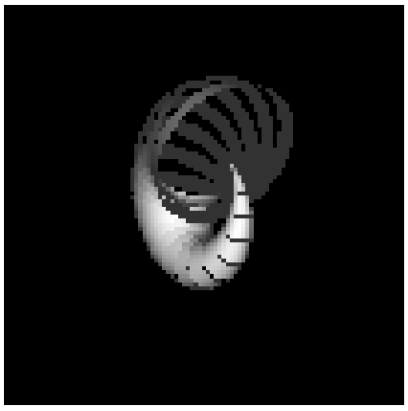


Figure 24: Nothing done to avoid gaps. See the unwanted gaps between cones in the convex areas.

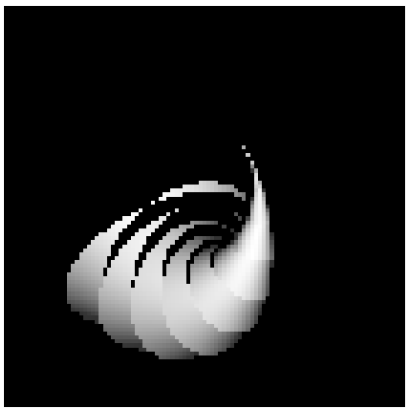


Figure 25: Using shared planes between cones. See the unwanted gaps between cones in the concave areas.

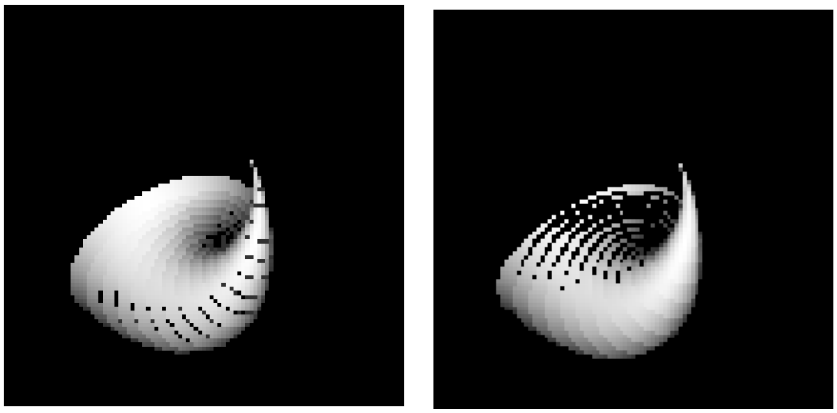
8.1 Ray Tracing

To ray tracing to these models, we need to intersect the ray with the truncated cones. The ray may intersect with more than one cone, it can intersect a cone in a point outside the truncated area, and it may intersect a cone in more than one point. If a ray intersects a cone, we apply test that the intersection point lies between the two truncating planes. In the case that a cone intersects the ray in a point outside its truncated area, we just ignore the intersection. With these modifications, the ray tracer described in Section 2.1 was used to ray trace the surfaces.

8.2 The Gap Between Cones

In modeling 3D thick curves with cones, the gap between cones needs to be handled. Figure 24 shows a sample object modeled with the curve-based high-level primitive without doing anything regarding the gaps. The curve in this example is a circle, its thickness is a linear function, and it is modeled with 20 cones. As is clear from the picture, the gaps between consecutive cones is because that the cones have different angles at the connection points. However, there is no gap in concave parts of the object because the cones fully cover it.

To solve this problem, one simple idea is to change the truncating planes such that they cover the gap between cones as much as possible. An implementation of this idea is shown in Figure 25. In this model the normal vector of truncating plane is the tangent vector of the curve at the connection point of two cones instead of the axis of the cone. This way, two consecutive cones shares a single truncating plane which makes the gap smaller. However, while the truncating planes in this model seem better in convex areas, they sometimes cause unwanted cuts in the concave parts. Figure 26 compares this method to the original model of cones. As shown in Figure 26, modifying truncating cones lead to a better result in convex areas, but makes unwanted



(a) (b)

Figure 26: Half circle with quadratic thickness modeled with 20 cones, with original, and modified truncating planes.

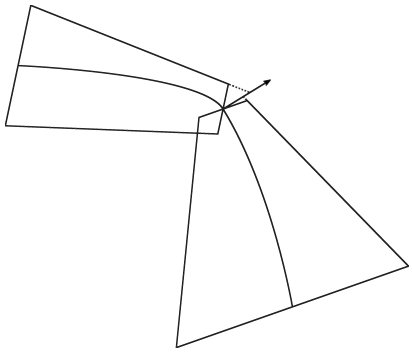


Figure 27: Modified truncating plane cannot fully cover the gap even in convex areas.

cut in concave parts.

The problem of the method of modifying truncating planes is not limited to concave areas. Figure 27 shows a case in which two cones may reach the truncating plane in different points, which forms a gap between them even in a convex area. Figure 28 shows another example of the spiral curve with both linear and quadratic thickness functions using the modified truncating planes method. In Figure 28 we can see the gap in both convex an concave parts of the surface. The idea proposed by Barbier et al. to avoid the gap is to put a sphere in each connection to make sure that the gap will be fully filled. Figure 29 shows how the sphere can fully cover the gap.

To avoid the intersection of the sphere with other cones, we need to truncated the connecting sphere, too. The only part of the sphere needed to fill the gap is the part between the second truncating plane of the first cone and the first truncating plane of the second cone, which can be easily identified with different signs of plane equations. Figure 30 shows the spiral with quadratic thickness modeled with this new technique in two different levels of detail. As we can see in the figure, there is no gap and this model fully covers the gaps.

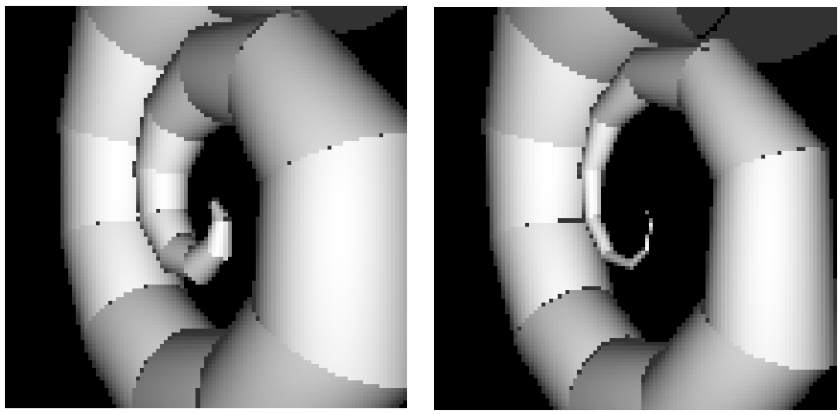
Although using spheres in the connections make bending smoother and more realistic, a problem may occur with spheres. As is visible in Figure 29, the sphere can form a concave crease in convex area of the surface which is undesirable. To avoid this problem and still cover the gap between cones, one idea is to replace sphere with an elliptic cylinder passing through the bases of consecutive cones. This model does not have the problem of sphere model, because the slope of this elliptic cylinder will be the mean of two cones' slopes.

To find the equation of the elliptic cylinder, similar to what we did for cones, we find the equation of an elliptic cylinder with the z-axis as the cylinder axis and transform it by appropriate rotation and translation matrices. Finally, the elliptic cylinder should be truncated with the same planes that form the base circles.

The equation for elliptic cylinder with z-axis as its axis is

$$\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 = 1,$$

where a and b are its two radii. Here a is equal to the radius of base circles, r, and b is smaller due to the angles between two base circles. This angle is determined by the angle between axes of cones. So b can be computed as follows:



(a) (b)

Figure 28: Spiral with linear and quadratic thickness. See the dark pixels due to the gap.

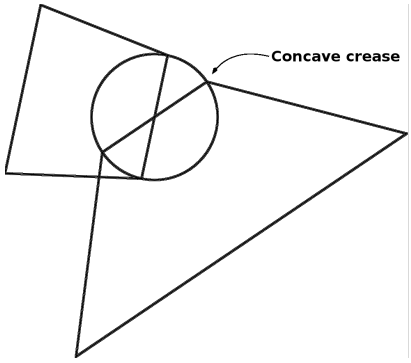
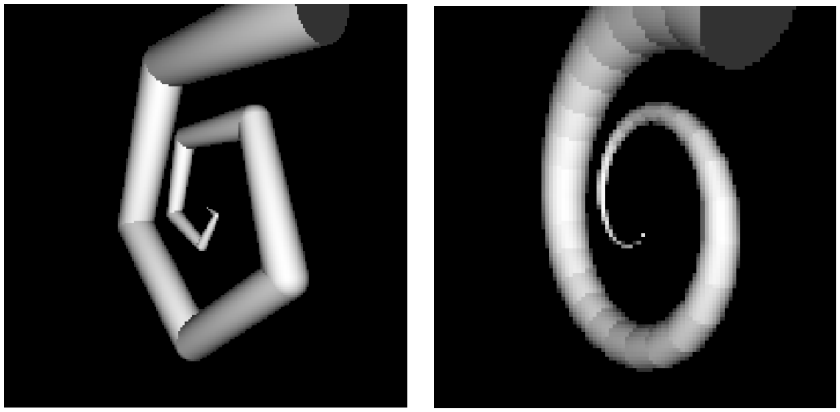
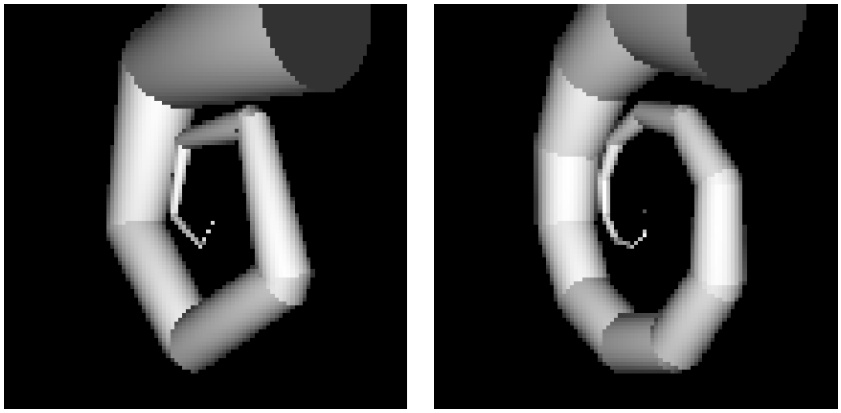


Figure 29: A sphere in each connection can fully cover the gap.



(a) (b)

Figure 30: Spiral with quadratic thickness with spheres at connections to fill the gaps, in two different levels of details.



(a) (b)

Figure 31: Spiral with quadratic thickness with elliptic cylinder at connections to fill the gaps, in two different levels of details.

$$\cos(2\theta) = \frac{\vec{z_1}.\vec{z_2}}{|z_1||z_2|}$$
$$b = r\cos(\theta).$$

In the elliptic cylinder, the radius b is along the angle bisector of $\vec{z_1}$ and $\vec{z_2}$, and a is perpendicular to the plane passing $\vec{z_1}$ and $\vec{z_2}$, so the vectors of base ellipsoid of the cylinder can be computed as:

$$\vec{b} = \frac{\frac{\vec{z_1}}{|z_1|} + \frac{\vec{z_2}}{|z_2|}}{|\frac{\vec{z_1}}{|z_1|} + \frac{\vec{z_2}}{|z_2|}|}, \quad \vec{a} = \frac{\vec{z_1} \times \vec{z_2}}{|\vec{z_1} \times \vec{z_2}|},$$

and the axis of cylinder is perpendicular to both $\vec{a}$ and $\vec{b}$, it can be computed as:

$$\vec{c} = \vec{a} \times \vec{b}.$$

Now, we should transform the axes to lie on $\vec{a}$, $\vec{b}$, and $\vec{c}$. We also should translate the origin to the connection point of the cones, p. The transformation matrix is

$$K = \begin{bmatrix} \vec{i}\cdot\vec{a} & \vec{i}\cdot\vec{b} & \vec{i}\cdot\vec{c} & p_x \\ \vec{j}\cdot\vec{a} & \vec{j}\cdot\vec{b} & \vec{j}\cdot\vec{c} & p_y \\ \vec{k}\cdot\vec{a} & \vec{k}\cdot\vec{b} & \vec{k}\cdot\vec{c} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where $\vec{i} = (1\ 0\ 0)$, $\vec{j} = (0\ 1\ 0)$, and $\vec{k} = (0\ 0\ 1)$ are the unit vectors of the coordinating system.

Figure 31 shows the spiral rendered with this model in two different levels of details.

Figure 32 shows a spiral rendered with these two models as well as without any object filling the gap.

8.3 Conclusion

The curve-based high-level primtive helps the designer to render the sphere in different levels of details automatically. Using this model, we can model any non-quadratic implicit surfaces by quadratic primitives. The only problem with this model is the transition between truncated cones. In this project three approaches were implemented and tested. The first one (extended truncated planes) could not cover the gaps between cones perfectly, and it also may cause some undesired cuts, but the two other models (using spheres or elliptic cylinders), are guaranteed to fill the gap. The first one makes a seamless transition from one cone to the next one, but can result in a concave crease in a convex area or a convex part of a sphere in the concave area, neither of which is desirable. The model of elliptic cylinders fills the gap between cones with a minimum information about the surface. It does not form any unwanted creases, and all discontinuities formed by it are in the direction of a natural crease between the cones.

9 Variational techniques in implicit surfaces — Khodakhast Bibak

The Radial Basis Function (RBF) method is one method for constructing 3D implicit surfaces. Roughly speaking, an RBF is a real-valued function whose value depends only on the distance from the origin or alternatively on the distance from some

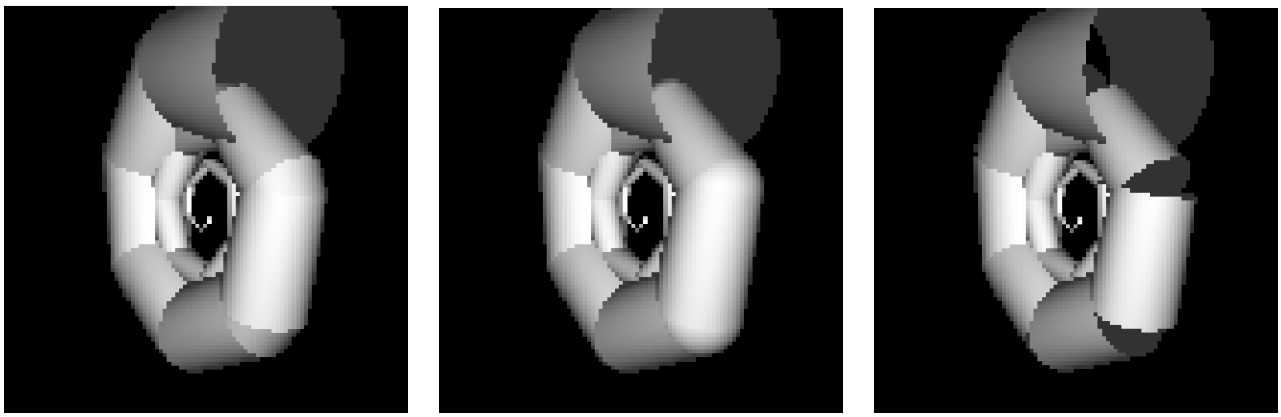


Figure 32: Spiral with quadratic thickness modeled with (left) spheres at connections; (center) with elliptic cylinders at connections; (right) without any object at connections.

other point called a *center*. RBFs apply to data fitting in neural networks and the theory of functional approximation [30]. One advantage of RBFs is their invariance under Euclidean transformations and their adjustable local influence when used for implicit modelling. Recent research has used RBFs to reconstruct implicit surfaces from scattered points. In this project, I will discuss some work using variational techniques to reconstruct surfaces from a point set. The next section describes reconstruction methods using RBFs with global support. Section 9.2 discuss methods allowing a higher number of constraints using greedy fast evaluation techniques of radial basis functions with global support. Section 9.3 presents methods using compactly supported radial basis functions, and thus also allow a higher number of constraints. In Section 9.4 looks at multi-level reconstruction techniques. Hermite Radial Basis Function (HRBF) implicits are discussed in Section 9.5.

9.1 Global Support

About 30 years ago, in an extensive survey, Franke [17] identified RBFs as one of the most accurate and stable methods to solve scattered data interpolation problems (this problem, roughly speaking, is stated as follows: Given n data points scattered on xy-plane along with corresponding scalar height values, find the smooth surface that interpolates each height at the given locations, i.e., find a smooth function f that passes through a given set of data points). The pioneering work to reconstruct implicit surfaces from given point sets using variational techniques can be attributed to Savchenko et al. [45]. In their method, the implicit surface is reconstructed by introducing a *carrier solid*, which in the simplest case can be a sphere. First, the radii of the carrier functions are calculated at all given points. The reconstruction problem is now shifted to finding a volume spline function that interpolates all values of radii. This method has some drawbacks. For example, volume interpolation requires heavy calculations (so the surface can neither be efficiently reconstructed nor efficiently evaluated). Another drawback of this method is that, besides the choice of the volume spline function, the carrier solid to use has to be defined. The choice of the carrier solid has a significant impact on the shape of the reconstructed surface, and it is a challenging problem how to determine an adapted carrier solid.

One reason for using a carrier solid for reconstructing implicits is to avoid the trivial solution of all coefficients being 0. Turk and O'Brien [53] gave another way to avoid the trivial solution of the system equations. In addition to the reconstruction constraints, they added additional constraints at off-surface points, which allows for the reconstruction of the implicit surface directly without defining a carrier solid. They introduced interior, exterior, and normal constraints. Then they applied the variational technique using the RBF $\phi(\mathbf{x}) = |\mathbf{x}|^3$. By a *boundary constraint* we mean points located on the surface. Positive-valued constraints were used to specify *interior constraints*, which are to specify a region that is inside to the surface. Likewise, negative-valued constraints were used to specify *exterior constraints*, which are to prevent the surface from overshooting into regions where the model should not exist. Also, *normal constraints* are placed very close to the boundary constraints, although the normal constraints are just a variation on interior and exterior constraints.

The Turk-O'Brien paper has some drawbacks. First, they did not mention the following items which are crucial in their paper: Energy function in 3D; the reason for appearance of $P(\mathbf{x})$ (however, P is required to guarantee that the linear system is solvable [57]); and also four orthogonality conditions for the system (see, e.g., [26]). Besides, their method, while it has several advantages, also has some drawbacks. For example, the additional constraints (or even a small change in even one constraint) can cause a large change in the resulting surface. Also, the off-surface points may contradict one another and some heuristics might be necessary to ensure that off-surface points produce a distance field consistent with the surface data. In addition, the linear system is dense and is tractable only for small data sets. In addition to the cost related to the solution of the equation system, a high-quality visualization of the isosurface requires the function to be evaluated at a large number of points. Because

of the global nature of variational implicit functions, all their terms must be used in computing the value function at any one point. Thus, each evaluation of the interpolated function has $O(n)$ time complexity. (Tests of the Turk-O'Brien scheme appear in a companion technical report [36].)

9.2 Fast Evaluation Techniques

The pioneering work to reconstruct implicit surfaces using radial basis functions with global support from a 'high' number of constraints can be attributed to Carr et al. [11]. They overcame the previously discussed computational cost and storage limitations by using a greedy fast evaluation technique of radial basis functions that allows for fast and storage-efficient computation of the matrix-vector products [5, 6], hence making iterative solutions attractive as solvers for the linear system. The greedy fast evaluation technique is based on the Fast Multiple Method of Greengard and Rohklin [21] that takes benefit of the fact, that when evaluating the radial basis function f at an argument x, infinite precision is not required and the evaluation can be approximated by dividing into far and near field expansions for a given argument x. On the one hand, all radial basis function terms in the near field, i.e., where the centers are close to the argument x, are computed directly and explicitly. On the other hand, many radial basis function terms of the far field whose centers are close to each other (but far away from the argument x) are approximated simultaneously by a Taylor series of truncated Laurent expansions. The accuracy, i.e., the length of the truncated Laurent expansion, can be preset, for example to a multiple of the machine precision of the computer in use. To identify the near and far fields of a given argument efficiently, the data has to be structured hierarchically.

Using the greedy evaluation technique, the evaluation cost drops from $O(n)$ to $O(1)$ after a $O(n \log n)$ setup, and calculating a matrix-vector product drops from $O(n^2)$ to $O(n)$. The computational cost to solve the linear system drops from $O(n^3)$ to $O(n \log n)$. Furthermore, since the involved matrix of the linear system never has to be calculated explicitly, storage requirements are also greatly reduced to only $O(n)$ compared to $O(n^2)$ before. Consequently, the method of Carr et al. can be considered efficient. Unfortunately, the far and near fields are complex to implement, and its derivation has to be done for every radial basis function separately. Besides the gain in computational and storage complexities, the reconstruction of the implicit surfaces is similar to Turk and O'Brien. Carr et al. also define off-surface constraints; however, they propose to add two new constraints at normal off-surface points on both sides of the surface for a subset of the initial points.

9.3 Compact Support

Morse et al. [38] propose another way to allow a higher number of constraints when reconstructing implicit surfaces by using Wendland's compactly supported radial basis functions of minimal degree [56]. In contrast to the methods presented so far, the resulting linear system is sparse. Similar to Turk et al. and Carr et al., Morse et al. use normal constraints to avoid the trivial solution of the linear system.

The sparsity has several advantages. For example, the linear system can be built up in $O(n \log n)$ time, solved in the range $O(n^{1.2})$ to $O(n^{1.5})$, and the evaluation drops to $O(\log n)$. Concerning storage complexity, by using sparse-matrix representations, only $O(n)$ storage is required.

The major drawback of this method is that the support radius has to be chosen globally, hence an additional parameter compared to globally supported radial basis functions is introduced. Choosing a large radius drops the performance, and may result in the same computational and storage complexities as by using globally supported radial basis functions. On the other hand, choosing a small radius may result in a too local reconstruction, possibly with the generation of holes in the surface. For these reasons, compactly supported radial basis functions should only be used to reconstruct implicit surfaces from quasi-uniformly distributed point sets. Another disadvantage is that the reconstructed implicit surface f is not the only set of zero-valued points in space, but there are extra zero-sets. Hence, point membership classifications, which are often used in constructive solid geometry (CSG) are not directly applicable to this kind of surfaces, as the common convention of the implicit surface to be positive inside and negative outside is not met. (Extra zero-sets are a common problem implicit surface reconstruction methods, and unless a method explicitly proves that there are no extra zero-sets, it should be assumed that the method creates them.)

The work of Morse et al. was further improved by Kojekine et al. [28] by organizing the sparse matrix into a band-diagonal sparse matrix in an efficient manner using an octree data structure. The resulting linear system can be solved more efficiently using a combination of block a Gaussian solution and a Cholesky decomposition [19]. Furthermore, Kojekine et al. use a carrier solid instead of normal constraints, which reduces the number of constraints and thus further improving efficiency. But Kojekine et al. state also that compactly supported radial basis functions are only suitable for 'moderately sized' 3D point sets.

Chen et al. [12] presented a method for efficiently creating compactly supported RBF-based implicit representations from the vertices of a polygonal model using hierarchical spatial partitioning. This method employs a hierarchical spatial partitioning that imposes a successive series of embedding functions constrained so that when they are added to one another, they interpolate the point set. The approach begins with the careful selection of a representative subset of the point set from which an interpolating implicit surface that provides a base model can be created. This base model interpolates the core subset of data points and serves as the foundation for the coarse-to-fine hierarchy. The data space is recursively divided into an octree with additional data points selected, and more detailed embedding functions are derived for each child octant that, when added to the

base model, accurately interpolate the more complete, higher resolution model. Unlike many other approaches that combine local interpolations through compactly supported blending functions, no explicit blending function is required—each level and partitioned patch is calculated so that a simple linear combination of them produces an exact interpolation.

9.4 Multi-level Methods

Compactly supported radial basis functions are also used by Ohtake et al. [42] to reconstruct implicit surfaces from larger point sets in a multi-resolution manner. Given a point cloud distributed along a surface, they first use spatial down sampling to construct a coarse-to-fine hierarchy of point sets. Then they interpolate the sets starting from the coarsest level. They interpolate a point set of the hierarchy, as an offsetting of the interpolating function computed at the previous level. Note that Ohtake et al. use the preconditioned biconjugate gradient method to solve the linear system, and since their function can be considered as a local carrier solid, no extra off-surface constraints are needed.

The multi-resolution approach of Ohtake et al. overcomes several limitations encountered before when compactly supported radial basis functions were used. First, the number of constraints can be significantly higher, i.e., Ohtake et al. reconstruct implicit surfaces from several hundreds of thousands of input points. Second, as the support radius varies throughout the different levels of the hierarchy, highly non-uniformly sampled point sets can be reconstructed allowing us to fill larger holes and repair incomplete data. On the other hand, similar to the drawback of traditional reconstruction methods using compactly supported radial basis functions, extra zero-sets are generated when the surface described by the sampled point set has thin parts. Furthermore, since the reconstruction of the finest levels of the hierarchy involves the solution of sparse linear systems with a high number of constraints, the number of input points is still limited.

To reconstruct implicit surfaces from a large number of points with associated normals, Ohtake et al. [41] use multi-level partition of unity implicits, i.e., an implicit surface with a global defining function that is reconstructed by partition of unity blending of local shape approximations. Basically, the reconstruction process starts by rescaling the point set into a bounding cube and creating an octree-based subdivision of this cube. At each cell of the octree, a local shape approximation is calculated, and while the local shape approximation is not accurate enough, the cell is subdivided recursively until a certain accuracy is achieved. Note that in this way the depth of the octree is dependent on the complexity of the reconstructed shape instead of on the number of points. Finally, local shape approximations are blended together in slightly overlapping domains using the partition of unity method. Since all local shape approximations are done by quadratic functions, multi-level partition of unity implicits can also be understood as piecewise defined algebraic surfaces. Multi-level partition of unity implicits are a powerful tool since implicit surfaces can be reconstructed from a large number of input points as well as from incomplete data, and they are one of the few methods that enables us to reconstruct implicit surfaces with sharp features [41].

Reuter et al. [44] obtained a multi-scale reconstruction method for large scattered data. To do this, they combined three well-known methods: variational techniques using RBFs are used to solve a set of small local reconstruction problems, thinning algorithms are used to obtain subsets of the data for intermediate resolutions, and the partition of unity (POU) method combines the local solutions to get the final reconstruction.[4] This combination is not only robust and efficient, but also offers a high level of scalability, and moreover it enables the adaptive selection of different precisions of the multi-scale reconstruction. In contrast to partition of unity implicits [41], where the local reconstructions of all the leaf nodes are blended together using the partition of unity method, Reuter et al. [44] used the partition of unity method also for the inner nodes of the hierarchy. Their method enables an adaptive evaluation of the defining function of the implicit surface after being reconstructed. This is particularly useful for a view-dependent visualization of the implicit surface, where surface parts closer to the viewpoint can be visualized more precisely compared to surface parts farther away from the viewpoint. Note that Reuter et al.'s method also requires the introduction of off-surface constraints, and takes a high reconstruction time compared to the methods of Ohtake et al. [41, 42].

9.5 Hermite Radial Basis Function Implicits

The offset requirement was avoided in some work deduced from a statistical-learning perspective [43], where normals were directly used in the variational problem. Pan et al. [43] incorporated normals directly in their regularized variational problem, where the reasoning is to consider the alignment of the gradient of the surface with the normals at the sample points. This amounts to solving a linear system for n point/normal pairs. However, this approach seems sensitive to non-uniform point distributions and does not ensure interpolated normals.

Hermite Radial Basis Function (HRBF) Implicits were introduced by Macêdo et al. [33, 34]. HRBF implicits interpolate, on its zero-level surface, simultaneously a given set of points and, unlike previous RBF approaches, their normal vectors. They are a special case of a generalized interpolation theory—Hermite-Birkhoff interpolation with RBFs—so that new variants of surface reconstruction methods can be designed for additional flexibility. HRBF implicits are an interpolant to first-order Hermite data based on radial basis functions and polynomials that are robust with respect to coarse and non-uniformly sampled data, deal

[4] The main idea of the partition of unity method is to divide the global domain of interest into smaller overlapping subdomains where the problem can be solved locally. More precisely, the global difficult problem is decomposed into several smaller local problems and their local solutions are combined using weighting functions that act as smooth blending functions to obtain the global solution.

effectively with close surface sheets, are able to produce detailed surface reconstructions, regularize independently points and normals and also reproduce polynomial-based surfaces [33, 34] (see also [9, 10]). Different from the previous RBF approaches, HRBF implicits do not depend on offset points to ensure existence and uniqueness of its interpolant. Intrinsic properties of this method allow the computation of implicit surfaces rich in details, with irregularly spaced points even in the presence of close sheets. Note that since HRBF implicits interpolate both points and normals, it not only ensures the existence of a non-null implicit function without the need of offset points, but it is also capable of generating effective results.

HRBF implicits has some drawbacks. For example, this model treated the 'pure' cases, but there is nothing to prevent 'hybrid' scenarios that might consist of data with different interpretations. For instance, it is not uncommon in sketch-based modeling applications to have consistently-oriented points as well as pairs of points and tangent vectors. Cases like this may be treated either as an instance of Hermite-Birkhoff interpolation and dealt with by solving a single linear system (since the Hermite data would rule out the trivial solution) or as an eigenproblem, or even a combination of both.

10 Interpolating and Approximating Implicit Surfaces Using Moving Least Squares

Zhujun (Grace) Yao

10.1 Introduction

Polygon models are widely used in computer graphics applications. However, many models have problems such as holes, gaps, T-junctions, self-intersections, and non-manifold structure. In contrast, other than self-intersection, implicit surface do not have these problems, and have other advantages such as accuracy, conciseness, affine invariance, arbitrary topology, guaranteed continuity, and efficient intersections. So it is meaningful to explore how to do the transformation from polygon models to implicit surfaces.

This project explores the idea of the Moving Least Squares algorithm introduced by Shen [47] to build interpolating and approximating implicit surfaces from triangle meshes. The user can choose either to interpolate the objects or approximate the objects.

10.2 Moving Least Squares

The idea of the moving least squares approximation is to solve for every point x a locally weighted least square problem. The influence of the data points is governed by a weight function ω: $R^d \rightarrow R$, which becomes smaller the further away its arguments are from each other. Ideally, ω vanishes for arguments $x, y \in R^d$ with $d(x, y)$ greater than a certain threshold. Such behavior can be modeled by using a translation-invariant weight function. Thus the standard moving least squares (taken from [57]) is defined as follows:

Definition 1 *For $x \in R^d$, the value $s_{f,X}(x)$ of the moving least squares approximation is given by $s_{f,X}(x)$ where*

$$s_{f,X}(x) = \min \left\{ \sum_{i=1}^{N} \left[f(x_i) - p(x_i) \right]^2 \omega(x, x_i) : p \in \pi_m(R^d) \right\}. \tag{3}$$

Use the following notation

$$\begin{aligned} \mathbf{b} &= [b_1, \ldots, b_M] \in R^M \\ \mathbf{B} &= [\mathbf{b}^\intercal(x_1), \ldots, \mathbf{b}^\intercal(x_N)]^\intercal \in R^{NM} \\ \mathbf{c} &= [c_1, \ldots, c_M] \in R^M \\ \mathbf{W} &= diag\left(\omega(x, x_i) : i \in [0, N]\right) \\ \boldsymbol{\phi} &= [f(x_1), \ldots, f(x_N)]^\intercal \in R^N \end{aligned}$$

the polynomial p can be defined in terms of basis functions $\mathbf{b}$ and coefficients $\mathbf{c}$ (which are functions of x) as

$$p = \sum_{j=1}^{M} c_j b_j \tag{4}$$

This reduces the minimization problem to finding the optimal coefficient vector $\mathbf{c}$. We then to minimize the function derived from Eq. 3

$$\begin{aligned} R(\mathbf{c}) &= \sum_{i=1}^{N} \left[f(x_i) - \sum_{j=1}^{M} (c_j b_j(x_i)) \right]^2 \omega(x, x_i) \\ &= \sum_{i=1}^{N} \left[\phi_i - \mathbf{B}_i \mathbf{c} \right]^2 \omega(x, x_i) \\ &= (\phi - \mathbf{B}\mathbf{c})^\intercal \mathbf{W} (\phi - \mathbf{B}\mathbf{c}) \\ &= (\phi^\intercal \mathbf{W} \phi) - 2\phi^\intercal \mathbf{W}\mathbf{B}\mathbf{c} + \mathbf{c}^\intercal \mathbf{B}^\intercal \mathbf{W}\mathbf{B}\mathbf{c}. \end{aligned}$$

Since $R(\mathbf{c})$ is a quadratic function in $\mathbf{c}$, we get a unique solution if $\mathbf{B}^\intercal\mathbf{W}\mathbf{B}$ is positive definite. Since $\mathbf{W}$ is a diagonal matrix whose diagonal elements $w(x, x_i)$ are always positive, hence $\mathbf{B}^\intercal\mathbf{W}\mathbf{B}$ is positive semi-definite. Moreover, $\mathbf{c}^\intercal\mathbf{B}^\intercal\mathbf{W}\mathbf{B}\mathbf{c} = 0$ means that $\mathbf{B}\mathbf{c} = 0$. Thus the polynomial $p = \sum_{j=1}^{M} c_j b_j$ vanishes on every x_i. Since this set is assumed to be $\pi_m(R^d)$-unisolvent, p and hence c must be zero. Since a unique solution exists we can use the necessary condition $\nabla R(\mathbf{c}) = 0$ to compute it. We find that

$$\nabla R(\mathbf{c}) = -2\mathbf{B}^\intercal\mathbf{W}\boldsymbol{\phi} + 2\mathbf{B}^\intercal\mathbf{W}\mathbf{B}\mathbf{c} = 0 \iff \mathbf{B}^\intercal\mathbf{W}\mathbf{B}\mathbf{c} = \mathbf{B}^\intercal\mathbf{W}\boldsymbol{\phi} \tag{5}$$

which gives $\mathbf{c} = (\mathbf{B}^\intercal\mathbf{W}\mathbf{B})^{-1}\mathbf{B}^\intercal\mathbf{W}\boldsymbol{\phi}$ and the problem is solved.

Shen took a non-standard approach to standard moving least squares in 3.5 General Matrix Form of his dissertation. He changed the matrix form to

$$\begin{bmatrix} \omega(\mathbf{x}, \mathbf{p}_1) & & \\ & \ddots & \\ & & \omega(\mathbf{x}, \mathbf{p}_N) \end{bmatrix} \begin{bmatrix} \mathbf{b}^\intercal(\mathbf{p}_1) \\ \vdots \\ \mathbf{b}^\intercal(\mathbf{p}_N) \end{bmatrix} \mathbf{c} = \begin{bmatrix} \omega(\mathbf{x}, \mathbf{p}_1) & & \\ & \ddots & \\ & & \omega(\mathbf{x}, \mathbf{p}_N) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_N \end{bmatrix}, \tag{6}$$

which according to our notation above is actually

$$\mathbf{W}\mathbf{B}\mathbf{c} = \mathbf{W}\boldsymbol{\phi}. \tag{7}$$

Compared to Eq. 5, Shen eliminate $\mathbf{B}^\intercal$ directly. Then to guarantee that Eq. 7 is solvable, Shen multiplies by $(\mathbf{W}\mathbf{B})^\intercal$:

$$(\mathbf{W}\mathbf{B})^\intercal\mathbf{W}\mathbf{B}\mathbf{c} = (\mathbf{W}\mathbf{B})^\intercal\mathbf{W}\boldsymbol{\phi} \tag{8}$$

$$\mathbf{B}^\intercal\mathbf{W}^2\mathbf{B}\mathbf{c} = \mathbf{B}^\intercal\mathbf{W}^2\boldsymbol{\phi} \tag{9}$$

Shen states in his dissertation that a constant basis function gives good enough results. When $f(x_i)$ is constant, from Eq. 9 the implicit function is

$$f(\mathbf{x}) = \frac{\sum_{i=1}^{N} \omega^2(\mathbf{x}, \mathbf{p}_i)\phi_i}{\sum_{i=1}^{N} \omega^2(\mathbf{x}, \mathbf{p}_i)} \tag{10}$$

which is a form of Sheppard's method [48, 57] (although Sheppard's method typically does not square the weight function).

It is clear that the weight function is an important element of the method. Shen uses the weight function

$$\omega(r) = \frac{1}{r^2 + \epsilon^2}, \tag{11}$$

which can provide both interpolating and approximating behavior by adjusting the parameter ϵ. When $\epsilon = 0$ the moving least-squares function will interpolate the sample points (constraint values). When ϵ is non-zero, the weight function is no longer singular at zero, and the moving least-squares function only approximates the constraint values. The following sections will be illustrated according to Shen's derivation of moving least squares.

10.3 True-Normal Constraints

Moving Least-Squares has the same problem seen earlier for interpolatory implicit surfaces (Section 9.1): if we only use sample points on the surface, then the we obtain the trivial solution of $f(x) = 0$. Shen looked at two options to solve this problem: the Pseudo-Normal Constraints of Turk-O'Brien [53][5] and what Shen calls True-Normal Constraints. I used Shen's True-Normal Constraints in my project [47].

[5] Turk and O'Brien called these "normal constraints"; Shen refers to them as "pseudo-normal constraints".

The general idea of True-Normal Constraints is as follows. A point constraint associate with a normal vector gives us information about the whole space instead of just the three points that are the original points and the two offset points. Assume we have a point constraint at $\boldsymbol{p}$ with a normal $\hat{n}$, Shen used the following shape function S:

$$\begin{aligned} S(x) &= \phi + (\boldsymbol{x} - \boldsymbol{p})^{\mathrm{T}} \hat{n} \\ &= \phi_0 + \phi_1 x_1 + \phi_2 x_2 + \phi_3 x_3, \end{aligned}$$

where $\boldsymbol{x} = (x_1, x_2, x_3)$.

With this shape function, the value of point x will not only depend on the inside and outside, but also be influenced by the distance to constraint point (which is on the surface and have constraint values ϕ as zero). The implicit function in Eq. 10 changes to:

$$f(\mathbf{x}) = \frac{\sum_{i=1}^{N} \omega^2(\mathbf{x}, \mathbf{p}_i) S(\mathbf{x})}{\sum_{i=1}^{N} \omega^2(\mathbf{x}, \mathbf{p}_i)} \tag{12}$$

10.4 Integrating Moving Least Squares

The previous method will interpolate and approximate the point cloud constraints well. However, we want to manipulate the data with triangles, and for each of these triangles we want to constrain the implicit function over its entire surface. If we were not interested in interpolating the triangles, we could approximate the desired effect with point constraints scattered over the surface of each polygon. Aside from potentially requiring a large number of points that require expensive computation, scattered point constraints work reasonably well for approximating surfaces. However, interpolating surfaces and surfaces that approximate closely will show undesirable bumps and dimples corresponding to the point locations. When we increase the number of constraint points on the surface, the result becomes better. However, this leads to more expensive computation and is infeasible. To get the infinite constraint points result, Shen integrated over each triangle. Assuming there are K triangles, let $\Omega_k, k \in [1 \dots K]$, be the domain for each K input triangle. Eq. 10 becomes

$$f(\mathbf{x}) = \frac{\sum_{i=1}^{k} \int_{\Omega_i} \omega^2(\mathbf{x} - \mathbf{p}_i) S(\mathbf{x}) d\mathbf{p}}{\sum_{i=1}^{k} \int_{\Omega_i} \omega^2(\mathbf{x} - \mathbf{p}_i) d\mathbf{p}}. \tag{13}$$

The shape function in Eq. 12 for triangles can be written as follows with $\mathbf{p}$ being regarded as the parametric representation relative to a triangle $\triangle \mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2$:

$$S(\mathbf{x}) = \hat{\mathbf{n}}^{\intercal} x - \hat{\mathbf{n}}^{\intercal}(\mathbf{p}_0 + (\mathbf{p}_1 - \mathbf{p}_0)s + (\mathbf{p}_2 - \mathbf{p}_0)t) \tag{14}$$

$$= \hat{\mathbf{n}}^{\intercal} x - \hat{\mathbf{n}}^{\intercal} \mathbf{p}_0 \tag{15}$$

$$\mathbf{p} = \mathbf{p}_0 + (\mathbf{p}_1 - \mathbf{p}_0)s + (\mathbf{p}_2 - \mathbf{p}_0)t. \tag{16}$$

From Eq. 15, we know $S(\mathbf{x})$ is constant over one triangle. Then Eq. 17 an be rewritten as

$$f(\mathbf{x}) = \frac{\sum_{i=1}^{k} S(\mathbf{x}) \int_{\Omega_i} \omega^2(\mathbf{x} - \mathbf{p}_i) d\mathbf{p}}{\sum_{i=1}^{k} \int_{\Omega_i} \omega^2(\mathbf{x} - \mathbf{p}_i) d\mathbf{p}}. \tag{17}$$

As $S(\mathbf{x})$ is easy to calculate from the normal of triangle and position of $\mathbf{p}_0$, the main issue is how to integrate the weight function. With the parameters t and s in Eq. 16, the eight function is

$$\omega(\mathbf{x}, s, t) = \omega(\mathbf{x}, \mathbf{p}_i) \tag{18}$$

$$= \frac{1}{\|\mathbf{x} - \mathbf{p}_i\|^2 + \epsilon^2} \tag{19}$$

$$= \frac{1}{\|\mathbf{x} - \mathbf{p}_0 + (\mathbf{p}_1 - \mathbf{p}_0)s + (\mathbf{p}_2 - \mathbf{p}_0)t\|^2 + \epsilon^2} \tag{20}$$

$$= \frac{C_i}{(t + A_i(s))^2 + B_i(s)}, \tag{21}$$

where

$$A_i(s) = \frac{(\mathbf{p}_2 - \mathbf{p}_0)^{\intercal} \cdot (\mathbf{p}_0 + (\mathbf{p}_1 - \mathbf{p}_0)s - \mathbf{x})}{\|\mathbf{p}_2 - \mathbf{p}_0\|^2} \tag{22}$$

$$B_i(s) = \frac{\|\mathbf{p}_0 + (\mathbf{p}_1 - \mathbf{p}_0)s - \mathbf{x}\|^2}{\|\mathbf{p}_2 - \mathbf{p}_0\|^2} - (A_i(s))^2 \tag{23}$$

$$C_i = \frac{1}{\|\mathbf{p}_2 - \mathbf{p}_0\|^2}. \tag{24}$$

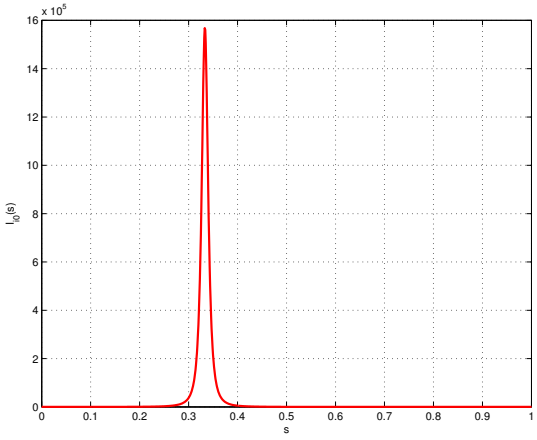


Figure 33: The function $I_{i0}(s)$

So the integral of the weight function is

$$Num(\mathbf{x}) = \int_{\Omega_i} \omega^2(\mathbf{x} - \mathbf{p}_i) d\mathbf{p} \tag{25}$$

$$= C_i^2 2Area_i \int_0^1 \int_0^{1-s} \frac{1}{[(t + A_i(s))^2 + B_i(s)]^2} \mathrm{d}t\mathrm{d}s \tag{26}$$

$$= C_i^2 2Area_i \int_0^1 I_{i0}(s)\mathrm{d}s \tag{27}$$

$$I_{i0}(s) = \int_0^{1-s} \frac{1}{[(t + A_i(s))^2 + B_i(s)]^2} \mathrm{d}t \tag{28}$$

$$= \frac{-\frac{A_i(s)\sqrt{B_i(s)}}{A_i(s)^2+B_i(s)} + \frac{\sqrt{B_i(s)}(1+A_i(s)-s)}{B_i(s)+(1+A_i(s)-s)^2} - \arctan\left[\frac{A_i(s)}{\sqrt{B_i(s)}}\right] + \arctan\left[\frac{1+A_i(s)-s}{\sqrt{B_i(s)}}\right]}{2B_i(s)^{3/2}}. \tag{29}$$

We know of no closed form solution to (27), so we need to compute the integral numerically. Unfortunately, integral of I_{i0} (shown in Figure 33) is difficult to compute using the trapezoid rule due to the shape of I_{i0}. In particular, even with a large number of samples, large errors in the numerical evaluation of the integral can lead to gross artifacts. To compute this integral accurately, Shen uses integration by substitution as detail in Appendix A; after doing the substitution, using the trapezoid rule to compute this integral gives good results.

10.5 Numerical Integration

In the two-dimensional integration problem which is required by integrating constraints over triangles, Shen presents a solution by transforming the integrands with singularities into flat ones using change of coordinates. The integrand is

$$I_{i0}(s) = \int_0^{1-s} \frac{1}{[(t + A_i(s))^2 + B_i(s)]^2} \mathrm{d}t. \tag{30}$$

To follow his change of coordinates method, we need to approximate $I_{i0}(s)$ by a function that captures its near-singularities and has a closed form integral. The crucial step to achieve a good approximation is to find the near-singularity point with maximum function value I_{i0}. We can choose unconstrained optimization methods to search the maximum point. As it is hard to derive the derivatives of I_{i0} and to find the step size that suitable to every I_{i0}, Gradient Descent and Newton's method are not the best approach to good results in limited steps. However, if we look into the properties of I_{i0} and the geometry aspect of the integration, a better way can be found to get the maximum point. The key of this method is to find the point within the triangle that is closest to the given point. According to classic graphics algorithms, such closest point p_m can be represented using

$$p_m = p_0 + (p_1 - p_0)s_m + (p_2 - p_0)t_m \tag{31}$$

where s_m and t_m are two parameters. In this case, the value of

$$w(t, s) = \frac{1}{[(t + A_i(s))^2 + B_i(s)]^2} \tag{32}$$

reaches its maximum at $p_m(t_m, s_m)$, and decreases as the point moves away from p_m. According to Eq. 32, if$s$ moves away from s_m, the value of $w(t_m, s_m)$ decreases at the rate four times $|s - s_m|$. At the same time, the change of integration interval is linear to $|s - s_m|$. When the largest exponent of s increases, the maximum position s_{max} will be closer to s_m. The fourth power is large enough so that $s_{max} - s_m < 1e - 6$ in most situations. Therefore it is feasible to simply use $s_{max} = s_m$ in the approximation except for the following situations:

- The triangle being integrated is an obtuse triangle, and the obtuse angle is on p_1. This is an exception because the integral interval grows a lot when moving from s_m.
- When $s_m = 1$, $w(t_m, s_m)$ is largest but the integral interval is zero. So s_{max} should be less than s_m and we cannot use $s_{max} = s_m$ here.

For these cases when s_{max} is significantly different from s_m, an unconstrained optimization algorithm has to be employed to find s_{max}. I implemented and tested three methods: Gradient Descent, Golden Search, and Fibonacci Search methods. A correct localization of s_m that fulfills the condition $|s_m - s_{max}| < 1e - 5$ will lead to a good approximation that captures the near-singularities of I_{i0} and has a closed-form integral.

From MATLAB experiments, Golden Search is found faster and more accurate than Gradient Descent and Fibonacci Search in most of the cases. However, there is no guarantee that any of these three methods will work well in all cases.

10.6 Fast Evaluation

It is infeasible to do integration on every triangle for each evaluation point. Approximation can be used when the distance between triangles and evaluation points are faraway. KD-Trees are a popular method to do fast evaluation. I implemented this but did not gain a significant speed-up. This is because building KD-Tree costs a lot of time. Since my test data was small, I used a simpler way to get a speed-up. As I need to calculate the nearest point in the numerical integration, it is easy to get the distance from the evaluation point to each triangle. I can use the distance information directly to classify the triangles as the evaluation point ON, SINGULAR, NEAR, or FAR:

- If the point of evaluation is less than ON= $5e^{-7}$ from the triangle, I return zero to the implicit function;
- If the point of evaluation farther than ON but closer than SINGULAR=0.05, I use Shen's method as described in Appendix A.
- If the point of evaluation farther than SINGULAR but closer than NEAR=0.5, $I_{i0}(s)$ will also have singularity issues even though it is not as serious. To get a better result in less time, we can use the information of s_m calculated before: Make s_m as the start point of the Newton-Cotes trapezoidal rule with irregularly spaced samples to capture the near-singularity points.
- If the point of evaluation is greater than FAR away from the triangle, then I use the centre of the triangle to approximate the contribution of the whole triangle's weight function, as we can see in Eq. 35:

$$\int_{\Omega_i} \omega^2(\mathbf{x} - \mathbf{p}_i) d\mathbf{p} \approx \int_{\Omega_i} \omega^2(\mathbf{x} - \mathbf{p}_c) d\mathbf{p} \tag{33}$$

$$= \omega^2(\mathbf{x} - \mathbf{p}_c) \int_{\Omega_i} d\mathbf{p} \tag{34}$$

$$= \omega^2(\mathbf{x} - \mathbf{p}_c) Area_i \tag{35}$$

10.7 Approximation, Interpolation, and Hole Filling

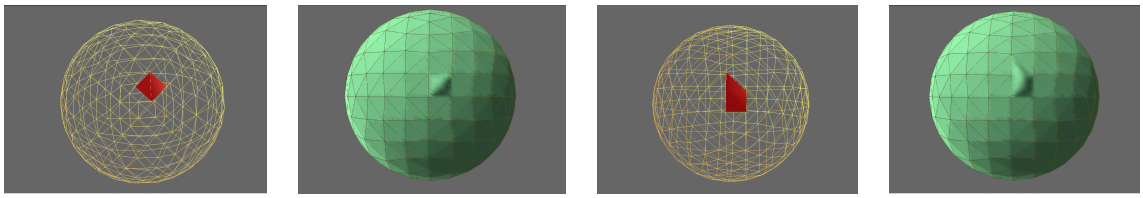
When the weighting function $\epsilon = 0$, moving least squares exactly interpolates the triangle mesh. If the input data has gaps or holes, this method would accomplish hole filling. However, this only resolves small holes. The result for small holes can be seen in Figure 34. When the hole size grows larger the protrusion at the missing face position grows larger and eventually extends to infinity; see Figure 35.

For $\epsilon \neq 0$, the value of weighting function is no longer singular at zero, and the moving least-squares function approximates the constraint values. Instead of dealing with the near-singular problem, we can directly use the trapezoid approximation method to compute the integral. Some results are shown in Figure 36.

The approximation method can also fill holes and give a better result than interpolation. But if the interpolation fill holes failed, the approximation will also failed, as shown in Figure 37.

10.8 Rendering Time

The rendering time for different polygon meshes are shown in Table 1.



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

Figure 34: Interpolate Surface with Different Sizes of Face Missing.

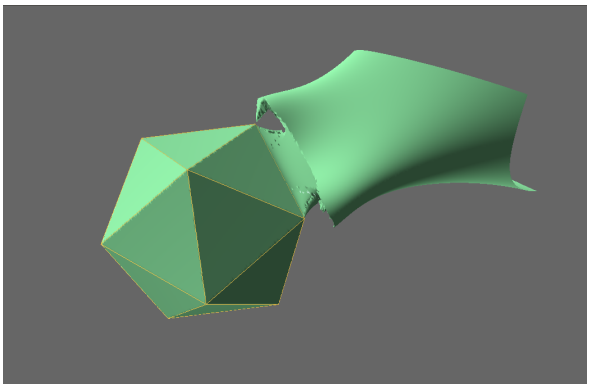


Figure 35: Large holes will not be filled correctly.

10.9 Reference Implicit Surface Rendering Results

The three surfaces in Figure 38 are the Implicit Surface directly rendered from their implicit function. The rendered cube has edges of 1, which is the same as the cube size I generated in previous sections. The sphere has a radius of 1. These numbers

Figure 36: Interpolation and approximation examples for Shen's method.

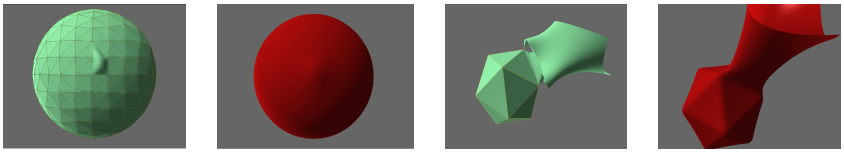
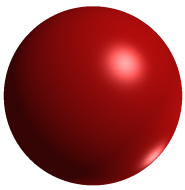


Figure 37: Approximation for Face Missing Meshes

Table 1: Rendering Time for Different Meshes

	Polys.In	ϵ	Tri.Out	Time(s)	ϵ	Tri.Out	Time(s)	ϵ	Tri.Out	Time(s)
Cube	6	0	18720	2	0.02	18720	7	0.2	26480	11
Icosahedron	20	0	137164	18	0.02	137564	103	0.2	146468	115
LetterA	31	0	120524	39	0.02	121456	225	0.2	136864	287
Mushroom	448	0	24360	58	0.02	24856	390	0.2	36812	577
Sphere	512	0	44520	84	0.02	44644	826	0.2	52272	953

0.06 seconds
18720 triangles
(a)



0.26 seconds
45092 triangles
(b)

0.01 seconds
6972 triangles
(c)

Figure 38: Implicit surface directly rendered from implicit function

should serve as a base-line for the time and triangle numbers I got for my implementation of Shen's method.

11 Turk-O'Brien—Animation — Stephen Mann

Turk and O'Brien suggest using their variational minimization method (described in Section 5.2) to create animations of morphings between two 2D curves or between two 3D surfaces [52]. For their curve method, each curve is described by a set of points on the curve. Turk and O'Brien embed the curves in a space of one higher dimension. Calling the extra dimension t, the points for curve C_i is embedded at $t = t_i$ for some set of increasing values t_i. A single surface is then fit to the set of all the points in 3D. To construct an animation, you extract the level set of the 2D curve at a sequence of t values.

To test this method, we created some animations using Koch snowflakes as the curves. Turk and O'Brien give no suggestions on how far apart to space the t_i values, so we tried several settings. Figure 39 shows our results. Several things can be observed from the figure. First, as seen in the top row, if the curves are spaced too far apart, then the variational minimization creates a surface that (in effect) smooths out the intermediate curves (as seen in the left column of Figure 39).

On the other hand, if we place the curves too close together, then neighboring curves have too strong an influence on one another. The right column of Figure 39 illustrates this; compare the bottom curve in this row to the bottom curve in the top row of the figure. In this example, the middle row has the best results, but even here, the top curves have noticeable influence on the bottom rows. Note that in these examples that the large difference in the density of points between adjacent levels likely leads to the strong influence of the denser curve to the less dense one. Regardless, each curve of data to interpolate will be influence by the data in all the curves to be interpolated, since their method is a global method.

As an alternative, we note that if we construct two implicit curves $C_0(P)$ and $C_1(P)$ using the Turk-O'Brien curve method, then we can affinely blend the implicit functions to obtain a time varying implicit surface: $C(t, P) = (1-t)C_0(P) + tC_1(P)$. A similar approach also works for surfaces. Figure 40 shows a comparison of this approach to that of Turk-O'Brien. In this figure, the Turk-O'Brien figures correspond to the slices in Figure 39, top row. Note that the normal constraints were placed a distance of 0.01 from the corresponding point to be interpolated.

The linear blend has several advantages over the Turk-O'Brien method. First, with the affine blend method, the curves/surfaces constructed appear at the initial/final steps of the blend (which is not true of the Turk-O'Brien method); second, the method is computationally less expensive, since the combined cost of inverting an $(n+4) \times (n+4)$ matrix and an $(m+4) \times (m+4)$ matrix is significantly less than that of inverting an $(n+m+4) \times (n+m+4)$ matrix. On the other hand, the affine blend approach can make no claims about the variation minimization at the intermediate values of the interpolation, and if a sequence of more than two curves/surfaces are blended, then there will be a discontinuities in the changing deformation at locations of the curves/surfaces being blended.

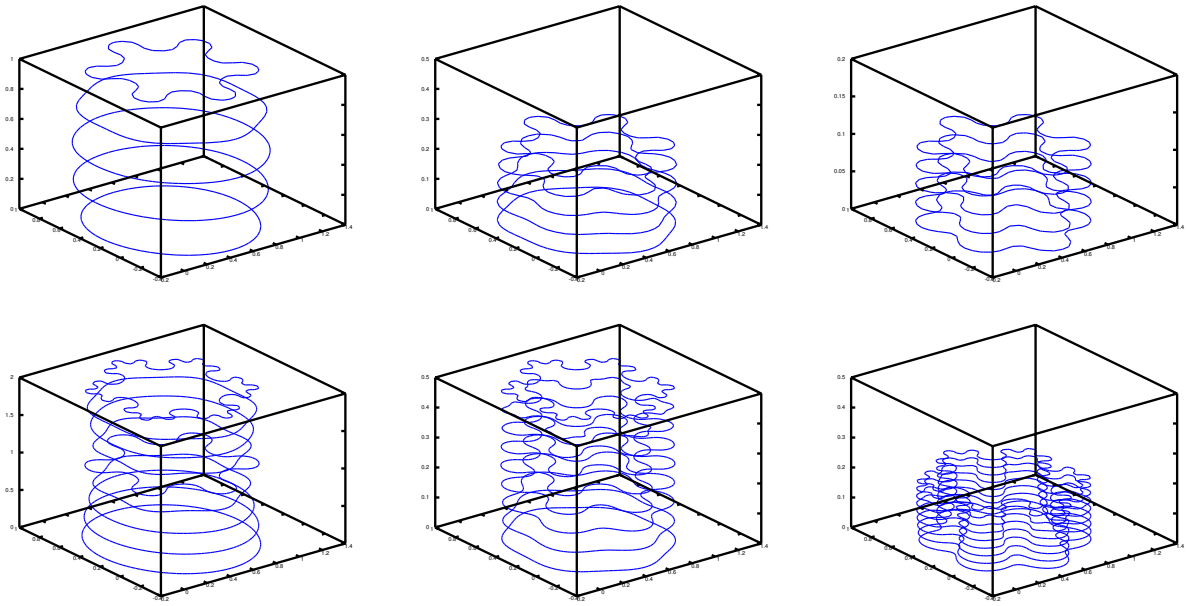


Figure 39: Slices through Turk-O'Brien animations of the Koch snowflake. Left column: spacing of 1; center column: spacing of 0.25; right column: spacing of 0.1. Top row: going from K_0 to K_1. Bottom row: going from K_0 to K_1 to K_2.

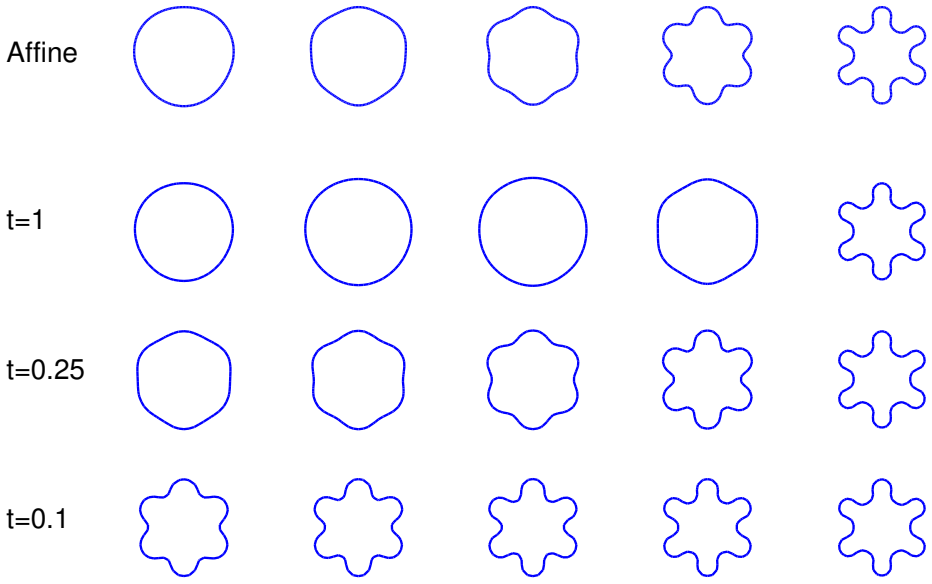


Figure 40: Comparison of affine blend to Turk-O'Brien blend. Top row: affine blend. Bottom three rows: Turk-O'Brien, with left image extra coordinate at $t = 0$ and right image extra coordinate at t value appearing in figure. In all images, columns correspond uniformly space t values.

12 Summary

We saw four types of implicit surfaces: Algebraic surfaces; A-patches; CSG-style; and Data fitting. Algebraic surfaces were mathematically the cleanest, but seemed the hardest to model with. A-patch and CSG-style implicits were similar to parametric patches and CSG modeling; while there were some advantages to using the implicit form, the modeling effort of the implicit was similar to the modeling effort of the non-implicit form, with somewhat similar results. The data fitting implicits gave the most elaborate models, although this was because you can fit the implicit to scanned data, and scanned real world objects are typically more elaborate than those modeled from scratch.

The choice among the four methods likely depends on your modeling and other needs, although it seems unlikely that one would use algebraics beyond possibly simple algebraics as leaf nodes in a CSG style modeler. While there were both triangle and rectangular surface patches in A-patch style, we did not study the rectangular style patches to decide if one is preferred to the others. Still, the A-patch style surfaces seem like a less suitable choice than the CSG-style or data fitting implicits.

Of the CSG-style methods, the Blob-tree seems the most developed. Whether it is better to use the Blob-tree implicit form of modeling over more standard CSG modeling often found in CAD packages is a harder question, mainly because there are more users of standard CAD packages than of the Blob-tree. Possibly this suggests that the standard CAD packages are better suited for CAD than the Blob-tree is, but we are not aware of any direct comparisons.

Of the data fitting methods, the two we studied most closely were the one of Turk-O'Brien and the one of Shen (a more detailed study of the two appears in a companion tech report [36]). Of the two, the Turk-O'Brien method gave better shaped surfaces with less effort than Shen's methods, although Shen is able to interpolate polygons while Turk-O'Brien are limited to interpolating points. Regardless, more recent methods are likely an improvement over both methods.

References

[1] Lionel Alberti, Georges Comte, and Bernard Mourrain. Meshing implicit algebraic surfaces: the smooth case. In Schumaker, editor, *Mathematical Methods for CAGD: Tromso 2004*, pages 11–26. Nashboro, 2005.

[2] Chandrajit L. Bajaj, Jindon Chen, and Guoliang Xu. Modeling with cubic A-patches. *ACM Trans. Graph.*, 14(2):103–133, April 1995.

[3] Aurlien Barbier, Eric Galin, and Samir Akkouche. Complex skeletal implicit surfaces with levels of detail. In *Proceedings of WSCG*, page 2004, 2004.

[4] Loïc Barthe, Brian Wyvill, and Erwin de Groot. Controllable binary CSG operators for "soft objects". *International Journal of Shape Modeling*, 10(2):135–154, 2004.

[5] R. K. Beatson and W. A. Light. Fast evaluation of radial basis functions: methods for two-dimensional polyharmonic splines. *IMA Journal of Numerical Analysis*, 17(3):343–372, 1997.

[6] R. K. Beatson and G. N. Newsam. Fast evaluation of radial basis functions. *Computational Mathematics and Applications*, 24(12):7–20, 1992.

[7] James F. Blinn. A generalization of algebraic surface drawing. *ACM Trans. Graph.*, 1(3):235–256, July 1982.

[8] J. Bloomenthal. Polygonalization of implicit surfaces. *Computer Aided Geometric Design*, 5:341–355, 1988.

[9] E. V. Brazil, I. Macêdo, M. C. Sousa, L. Velho, and L. H. de Figueiredo. Sketching variational hermite-rbf implicits. *Eurographics Symposium on Sketch-Based Interfaces and Modeling*, 2010.

[10] E. V. Brazil, I. Macêdo, M. C. Sousa, L. Velho, and L. H. de Figueiredo. Shape and tone depiction for implicit surfaces. *Computers & Graphics*, 35:43–53, 2011.

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

[12] D. T. Chen, B. S. Morse, B. C. Lowekamp, and T. S. Yoo. Hierarchically partitioned implicit surfaces for interpolating large point set models. In *GMP '06*, pages 553–562, 2006.

[13] Evgeni V. Chernyaev. Marching cubes 33: Construction of topologically correct isosurfaces. Technical Report CN 95-17, CERN, 1995.

[14] Erwin de Groot, Brian Wyvill, and Huub van de Wetering. Locally restricted blending of Blobtrees. *Computers & Graphics*, 33(6):690–697, 2009.

[15] Daouda Niang Diatta, Bernard Mourrain, and Olivier Ruatta. On the isotopic meshing of an algebraic implicit surface. *J. Symb. Comput.*, 47(8):903–925, August 2012.

[16] William Donnelly. Per-pixel displacement mapping with distance functions. In Sweeney Pharr, Fernando, editor, *GPU Gems 2*, chapter 8, pages 123–136. Addison-Wesley, 2005.

[17] R. Franke. Scattered data interpolation: tests of some methods. *Mathematics of Computation*, 38(157):181–200, 1982.

[18] B. Fry and C. Reas. Processing. http://processing.org/.

[19] A. George and J. W. H. Liu. *Computer solution of large sparse positive definite systems*. Prentice-Hall, NJ, 1981.

[20] Abel Gomes, Irina Voiculescu, Joaquim Jorge, Bryan Wyvill, and Callum Galbraith. *Implicit Curves and Surfaces: Mathematics, Data Structures and Algorithms*. Springer Verlag, 2009.

[21] L. Greengard and V. Rokhlin. A fast algorithm for particle simulation. *Journal of Computational Physics*, 73:325–348, 1987.

[22] Andrew Guy and Brian Wyvill. Controlled blending for implicit surfaces. In *Implicit Surfaces '95*, April 1995.

[23] John C. Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12:527–545, 1996.

[24] Paul S. Heckbert, editor. *Graphics Gems IV*. Academic Press Professional, Inc., San Diego, CA, USA, 1994.

[25] Xiaogang Jin, Hanqiu Sun, and Qunsheng Peng. Subdivision interpolating implicit surfaces. *Computers and Graphics*, 27:763–772, 2003.

[26] X. Jina, H. Sunb, and Q. Peng. Subdivision interpolating implicit surfaces. *Computers & Graphics*, 27:763–772, 2003.

[27] M.W. Jones and M. Chen. A new approach to the construction of surfaces from contour data. In *Eurographics '94 Conference Proceedings*, volume 13, pages 75–84, 1994.

[28] N. Kojekine, I. Hagiwara, and V. Savchenko. Software tools using CSRBFs for processing scattered data. *Computers & Graphics*, 27(2):311–319, 2003.

[29] F. Levet, X. Granier, and C. Schlick. Fast sampling of implicit surfaces by particle systems. In *Proceedings of the IEEE International Conference on Shape Modeling and Applications*, SMI, pages 39–44, 2006.

[30] Q. Li, D. Wills, R. Phillips, W. J. Viant, J. G. Griffiths, and J. Ward. Implicit fitting using radial basis functions with ellipsoid constraint. *Computer Graphics Forum*, 23(1):55–69, 2004.

[31] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, August 1987.

[32] Curtis Luk. Tessellating algebraic curves and surfaces using A-patches. Master's thesis, University of Waterloo, 2008.

[33] I. Macêdo, J. P. Gois, and L. Velho. Hermite interpolation of implicit surfaces with radial basis functions. In *Proceedings of XXII Brazilian symposium on computer graphics and image processing (SIBGRAPI '09)*, pages 1–8, 2009.

[34] I. Macêdo, J. P. Gois, and L. Velho. Hermite radial basis functions implicits. *Computer Graphics Forum*, 30(1):27–42, 2011.

[35] Stephen Mann. Using A-patches to tessellate algebraic curves and surfaces. Technical Report CS-2009-21, University of Waterloo, 2009.

[36] Stephen Mann. A study of two implicit data interpolation schemes. Technical Report CS-2013-09, Cheriton School of Computer Science, University of Waterloo, September 2013.

[37] Sergey V. Matveyev. Approximation of isosurface in the marching cube: ambiguity problem. In *Proceedings of the conference on Visualization '94*, VIS '94, pages 288–292, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

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

[39] Gregory M. Nielson and Bernd Hamann. The asymptotic decider: resolving the ambiguity in marching cubes. In *Proceedings of the 2nd conference on Visualization '91*, VIS '91, pages 83–91, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.

[40] Paul Ning and Jules Bloomenthal. An evaluation of implicit surface tilers. *IEEE Comput. Graph. Appl.*, 13(6):33–41, November 1993.

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

[42] Y. Ohtake, A. Belyaev, and H.-P Seidel. Multi-scale approach to 3d scattered data interpolation with compactly supported basis functions. In *Proceedings of Shape Modeling International*, pages 153–161, 2003.

[43] R. Pan, X. Meng, and T. Whangbo. Hermite variational implicit surface reconstruction. *Science in China Series F: Information Sciences*, 52(2):308–315, 2009.

[44] P. Reuter, C. Schlick, and I. Tobor. Reconstructing multi-scale variational partition of unity implicit surfaces with attributes. *Graphical Models*, 68(1):25–41, 2006.

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

[46] Vladimir Savchenko, Er A. Pasko, Oleg G. Okunev, and Tosiyasu L. Kunii. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum*, 14:181–188, 1995.

[47] Chen Shen. *Building Interpolating and Approximating Implicit Surfaces Using Moving Least Squares*. PhD thesis, University of California, Berkeley, 2007. UCB/EECS-2007-14.

[48] D. Sheppard. A two dimension interpolation function for irregularly spaced data. In *Proceedings of ACM National conference*, pages 517–524. ACM, 1968.

[49] Masamichi Sugihara, Brian Wyvill, and Ryan Schmidt. WarpCurves: A tool for explicit manipulation of implicit surfaces. *Computers & Graphics*, 34(3):282–291, 2010. Shape Modeling International (SMI) 2010.

[50] R. Szeliski and D. Tonnesen. Surface modeling with oriented particle systems. Technical report, Digital Equipment Corporation, 1991. CRL–91–14.

[51] U. Tiede, K. Heinz, M. Bomans, A. Pommert, M. Riemer, and G. Wiebecke. Investigation of medical 3-d-rendering algorithms. *IEEE Computer Graphics and Applications*, pages 41–53, 1990.

[52] Greg Turk and James F. O'Brien. Shape transformation using variational implicit functions. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 335–342, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[53] Greg Turk and James F. O'Brien. Modelling with implicit surfaces that interpolate. *ACM Trans. Graph.*, 21(4):855–873, 2002.

[54] L. Velho, J. de Miranda Gomes, and D. Terzopoulos. Implicit Manifolds, Triangulations, and Dynamics. *Journal of Neural, Parallel, and Scientific Computation*, 5(1–2):103–120, 1997. Special issue on CAGD.

[55] Z.L. Wang, J.C.M. Teo, C.K. Chui, S.H. Ong, C.H. Yan, S.C. Wang, H.K. Wong, and S.H. Teoh. Computational biomechanical modeling of the lumbar spine using marching-cubes surface smoothened finite element voxel meshing. *Computer Methods and Programs in Biomedicine*, 80(1):25–35, 2005.

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

[57] Holger Wendland. *Scattered Data Approximation*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2005.

[58] Andrew P. Witkin and Paul S. Heckbert. Using particles to sample and control implicit surfaces. In *SIGGRAPH*, pages 269–277, 1994.

[59] Brian Wyvill, Eric Galin, and Andrew Guy. Extending the CSG tree. Warping, blending and boolean operations in an implicit surface modeling system. *Computer Graphics Forum*, 18(2):149–158, June 1999.

[60] Brian Wyvill, Craig McPheeters, and Geoff Wyvill. Animating *soft* objects. *The Visual Computer*, 2(4):235–242, 1986.

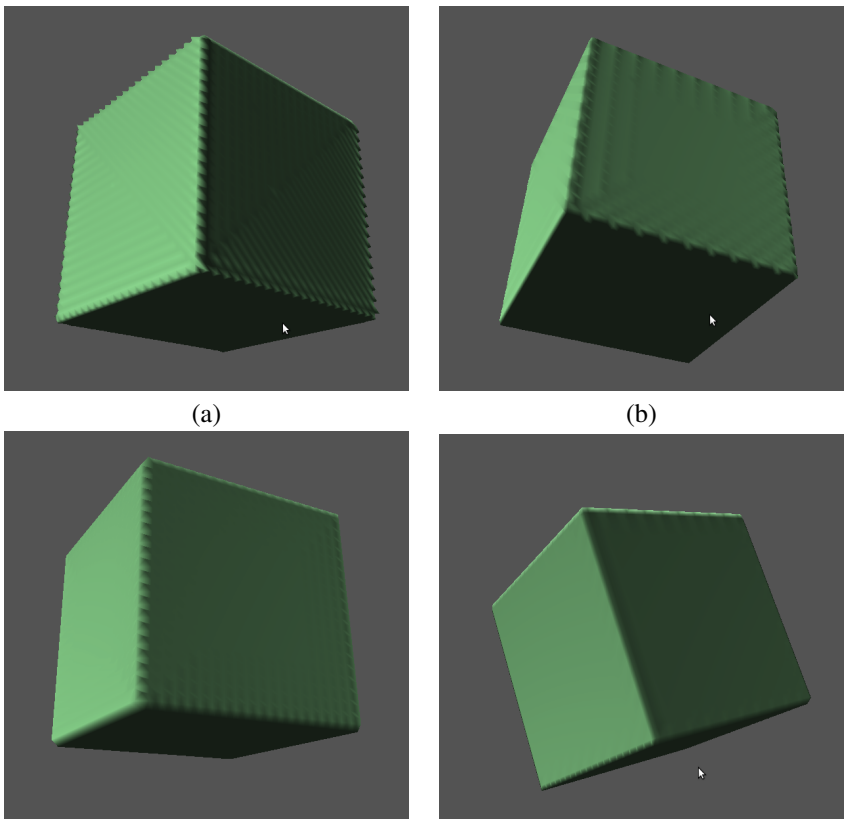
[61] Geoff Wyvill, Craig McPheeters, and Brian Wyvill. Data structure for *soft* objects. *The Visual Computer*, 2(4):227–234, 1986.

A Shen's Method: Transforming from singularities to flat ones

From Eq.(11), the integral is singular when $\epsilon = 0$ and the point of evaluation is on the surface, and near singular when the point of evaluation is near the surface. So the trapezoid rule (or trapezium rule) for approximating the definite integral has significant error with a limited number of steps. Even though we can decrease the error by increasing the step (in Figure 41), the resulting surface may have bumps.

To reduce this error, Shen used integration by substituion. When we map the function we want to integrate onto a different space and use a compensating multiplicative factor, we want to make sure that the new function satisfies two properties:

- Accurate: The area under the new function is the same as the area under the original, badly behaved function.
- Well-behaved: the new function should be roughly constant over the domain, and thus can be integrated much more accurately than a function with a singularity.



(a) (b)

(c) (d)

Figure 41: The comparison of 20, 50, 100, and 150 iterations

What we need to to calculate is

$$\int_0^1 I_{i0}(s)\mathrm{d}s. \tag{36}$$

To achieve this, we define another function in a different parameter space,

$$s = s(j), \quad j \in [j_0, j_1] \tag{37}$$

where $j_0 = s^{-1}(0)$ and $j_1 = s^{-1}(1)$. Then Eq.(36) can be written as

$$\int_0^1 I_{i0}(s)\mathrm{d}s = \int_{s^{-1}(0)}^{s^{-1}(1)} I_{i0}(s(j))\mathrm{d}s(j) \tag{38}$$

$$= \int_{s^{-1}(0)}^{s^{-1}(1)} I_{i0}(s(j))s'(j)\mathrm{d}j. \tag{39}$$

The part we want to be flat is

$$h(j) = I_{i0}(s(j))s'(j) \approx 1. \tag{40}$$

With this flat function Eq.(40), we want to choose $s(j)$ so that the $h(j)$ would satisfy the second property (to be well-behaved and easy to integrate):

$$s'(j) \approx \frac{1}{I_{i0}(s(j))} \tag{41}$$

Unfortunately, $I_{i0}(s)$ does not have an analytical solution. To get $s'(j)$, we can to approximate $I_{i0}(s)$ by a function that can capture the near-singularities and has a closed-form integral. The curve of $I_{i0}(s)$ is plotted in Figure 33.

As we know the shape of $I_{i0}(s)$, we can use the following function to approximate it:

$$I_{i0}^{\text{approx}}(s) = \frac{1}{c_2 s^2 + c_1 s + c_0} \tag{42}$$

To fit $I_{i0}^{\text{approx}}(s)$ to I_{i0}, which will capture the near-singularity part, the essential point is to find the maximum of I_{i0} as accurately as possible. It is not easy to get the maximum point directly; see Section 10.5 for my approach to computing it. With this maximum point and two near points on the left and right called s_m, s_-, s_+ we can solve for the coefficients c_0, c_1, c_2 of (42):

$$\begin{bmatrix} s_-^2 & s_- & 1 \\ s_m^2 & s_m & 1 \\ s_+^2 & s_+ & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{I_{i0}(s_-)} \\ \frac{1}{I_{i0}(s_m)} \\ \frac{1}{I_{i0}(s_+)} \end{bmatrix} \tag{43}$$

Figure 42(b) shows the approximate function, compared to the accurate one plotted in Figure 42(a). The comparasion of both are in Figure 42(d)

$I_{i0}^{\text{approx}}(s)$ will make $s(j)$ a closed-form integral, and we can get $s(j)$ and s' easily as follows,

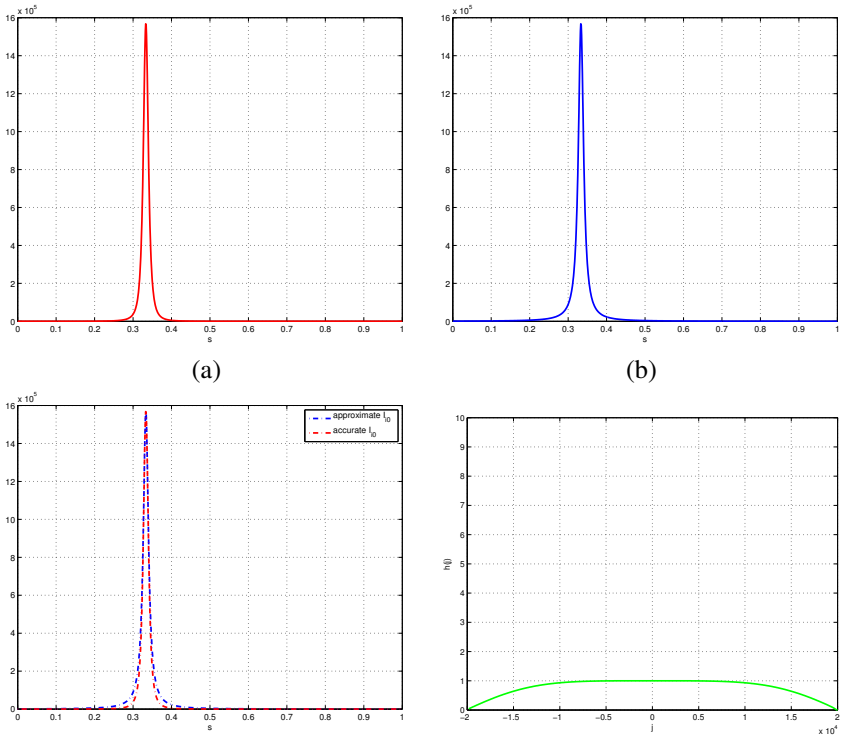
$$s(j) = \frac{-c_1 + \sqrt{-c_1^2 + 4c_0c_2}\tan\left[\frac{1}{2}\sqrt{-c_1^2 + 4c_0c_2}\right]}{2c_2} \tag{44}$$

$$s'(j) = \frac{\sec^2\left[\frac{1}{2}j\sqrt{-c_1^2 + 4c_0c_2}\right](-c_1^2 + 4c_0c_2)}{4c_2} \tag{45}$$

Then $h(j)$ is given by Eq.(40); an example appears in Figure 42(d). As we can see in Figure 42(d), the function we are going to integrate is well-behaved and we can use 15 steps of the trapezoid rule to get a much better result than applying the trapazoid rule to compute (36) directly. We can see the result of 200 steps of naive trapezoid approximation and the 15 steps of the flatten transforming method in Figure 43. As we can see in the results, the rendering time decreases significantly and the uneven problem in Figure 43(a) is also solved in Figure 43(b).

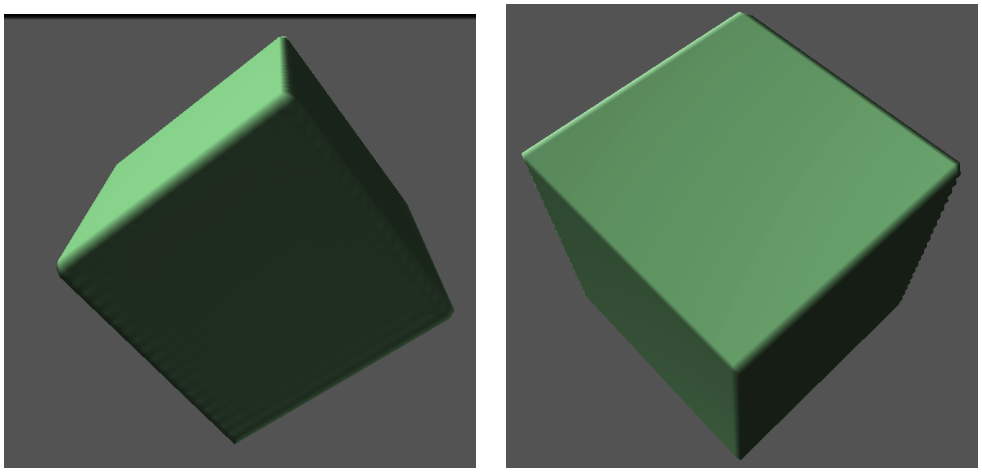
To summarize, the process for computing (36) is as follows:

1. Use optimization to find the maximum of I_{i0}
2. Approximate I_{i0} with I_{i0}^{approx}
3. Compute s and s' using (44) and (45)
4. Use s and s' to compute (36) using (39).



(a) (b)

Figure 42: The transform procedure: (a) is the plot of $I_{i0}(s)$, (b) is the approximate $I_{i0}^{\text{approx}}(s)$, (c) is a comparison of (a) and (b), (d) is the flatten function h.



(a) (b)

Figure 43: Comparison of 200 steps (a) of naive trapezoid approximation (120 seconds) and (b) 15 steps of the flatten transforming method (48 seconds). The number triangles after marching cubes is 18720.