Surface Reconstruction from Unorganized Points

by

Hugues Hoppe

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Washington

1994

Approved by____
(Chairperson of Supervisory Committee)

____

____

Program Authorized
to Offer Degree____

Date____

University of Washington

Abstract

Surface Reconstruction from Unorganized Points

by Hugues Hoppe

Chairperson of Supervisory Committee: Professor Tony DeRose
Department of
Computer Science and Engineering

This thesis describes a general method for automatic reconstruction of accurate, concise, piecewise smooth surfaces from unorganized 3D points. Instances of surface reconstruction arise in numerous scientific and engineering applications, including reverse-engineering—the automatic generation of CAD models from physical objects.

Previous surface reconstruction methods have typically required additional knowledge, such as structure in the data, known surface genus, or orientation information. In contrast, the method outlined in this thesis requires only the 3D coordinates of the data points. From the data, the method is able to automatically infer the topological type of the surface, its geometry, and the presence and location of features such as boundaries, creases, and corners.

The reconstruction method has three major phases: 1) initial surface estimation, 2) mesh optimization, and 3) piecewise smooth surface optimization. A key ingredient in phase 3, and another principal contribution of this thesis, is the introduction of a new class of piecewise smooth representations based on subdivision. The effectiveness of the three-phase reconstruction method is demonstrated on a number of examples using both simulated and real data.

Phases 2 and 3 of the surface reconstruction method can also be used to approximate existing surface models. By casting surface approximation as a global optimization problem with an energy function that directly measures deviation of the approximation from the original surface, models are obtained that exhibit excellent accuracy to conciseness trade-offs. Examples of piecewise linear and piecewise smooth approximations are generated for various surfaces, including meshes, NURBS surfaces, CSG models, and implicit surfaces.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I wish to express sincere appreciation to my advisor Tony DeRose, Statistics professors Werner Stuetzle and John McDonald, and Mathematics professor Tom Duchamp for their help and guidance in pursuing this research. The multidisciplinary nature of the research group was a tremendous benefit both to me and to the outcome of the research, as each discipline contributed crucial ideas to this thesis. In particular, I want to thank Tony for much needed encouragement over the last four years, and for his admirable integrity.

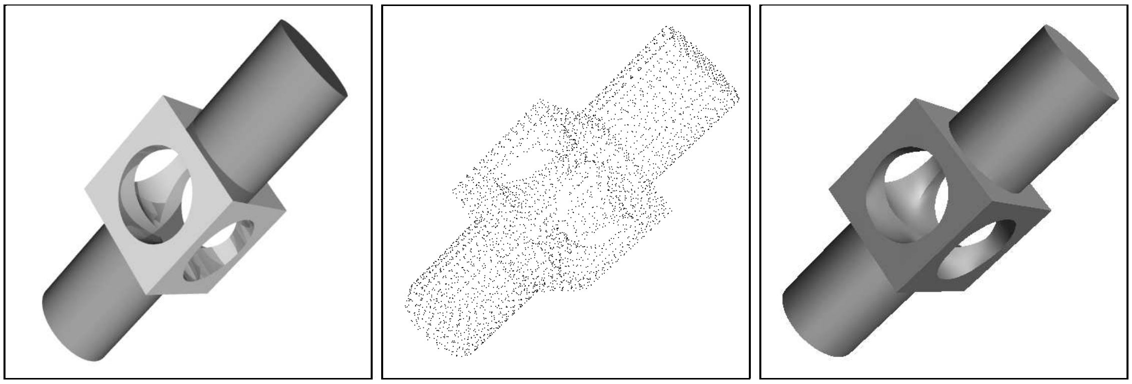
I wish to thank Steve Mann for introducing me to computer graphics and geometric modeling, for many exciting discussions, and for several entertaining graphics projects outside my thesis work.

I want to thank Michael Lounsbery, Jean Schweitzer, and the other graphics students in the U.W. Computer Science & Engineering Department for making my graduate student experience very pleasurable.

My research was funded in part by the National Science Foundation and by an IBM Graduate Fellowship. I would also like to thank Technical Arts Co. and especially Ken Birdwell for providing several sets of real data for my experiments.

Chapter 1

INTRODUCTION



(a) Unknown surface U (b) Sampled points X (c) Reconstructed surface S

Figure 1.1: Example of surface reconstruction.

1.1 Problem statement

Computer-aided geometric design and computer-aided manufacturing systems are used in numerous industries to design and create physical objects from digital models. However, the reverse problem, that of inferring a digital description from an existing physical object, has received much less attention. We refer to this problem as reverse-engineering or, more specifically, 3D scanning. There are various properties of a 3D object that one may be interested in recovering, including its shape, its color, and its material properties. This thesis addresses the problem of recovering 3D shape, also called *surface reconstruction*.

The goal of surface reconstruction can be stated as follows: Given a set of sample points X assumed to lie on or near an unknown surface U, create a surface model S approximating U (see Figure 1.1).

As we shall see shortly, previous reconstruction methods have usually been crafted to exploit characteristics of specific problem instances.

In contrast, this thesis examines the surface reconstruction problem in a general form that makes few assumptions about the sample X and the unknown surface U. In the *general surface reconstruction problem* we consider, the points X may be noisy, and no structure or other information is assumed within them. The surface U (assumed to be a manifold[1]) may have arbitrary topological type[2], including boundaries, and may contain sharp features such the creases and corners present in the surface of Figure 1.1a. Since the points X may be a noisy sampling, we do not attempt to interpolate them, but will instead find an approximating surface.

Of course, a surface reconstruction procedure cannot guarantee recovering U exactly, since it is only given information about U through a finite set of sample points. The reconstructed surface S should have the same topological type as U, and be everywhere close to U. In this thesis we will evaluate the reconstruction method by considering examples where the true underlying surface U is known and can be compared visually and quantitatively to the reconstruction.

1.2 Motivating applications

We were led to consider the general surface reconstruction problem stated above by a number of application areas in science and engineering, including 3D scanning, surface reconstruction from contours, and surface sketching. The next three sections discuss each of these application areas in more detail.

1.2.1 3D scanning

One of the most important applications of surface reconstruction is *3D scanning*—the measurement and modeling of shape and other visual properties.

As shown in Table 1.1, many other physical forms such as images, movies, and sound, can be measured using common consumer devices like document scanners, videocameras,

---

[1] Intuitively speaking, a *manifold* is a surface that does not intersect itself. More precisely, a manifold (possibly with boundary) embedded in $\mathbf{R}^3$ is a set everywhere locally homeomorphic to either a disk or a half-disk, where a *homeomorphism* is a continuous invertible map whose inverse is also continuous [47].

[2] The topological type of a surface refers to its genus, the presence of boundaries, etc. (cf. [47]).

and microphones. These multimedia devices have had an enormous impact because the models they generate can be used in many ways that physical forms cannot. For example, such models can be transmitted digitally, stored in databases, edited and analyzed with software, and used as templates for making digital or physical copies. Another set of technologies also allows for the re-instantiation of these same physical forms, using devices such as printers, televisions, and speakers.

Table 1.1: Modeling of physical forms.

physical form	acquisition	representation	instantiation
image	document scanner	TIFF image, postscript	printer
movies	videocamera	videotape, laserdisc	television
sound	microphone	compact disc, digital audio tape	speaker
shape	3D scanner + **surface reconstruction**	**concise representation**	NC milling, stereo lithography

Our vision is to put shape on an equal footing with these other media. We would like to acquire, represent, analyze, and recreate 3D shapes with ease.

3D scanning technology There are numerous methods for acquiring shape information. For instance, in computer vision, registration of landmarks in multiple views is used to infer object shape. In a different technique called shape from shading, the intensity of light reflected from the object's surface provides knowledge of surface orientation, and with further processing, the global shape of the object.

In the manufacturing industries, mechanical touch probes mounted on coordinate measuring machines are used to record points on surfaces such as car bodies and airplane wings. The resulting measurements are very accurate, but the technique is extremely slow and limited to materials that can withstand mechanical contact. Cheaper, less accurate hand-held 3D digitizing probes determine position using magnetic fields (Polhemus Corp.)

or ultrasound (Science Accessories Corp.). However, this type of “digitization” requires significant human intervention.

Recently, mechanical probes are being replaced by laser range scanners. Laser range scanners illuminate the object with a laser beam, and measure distance using either triangulation, interference, or time-of-flight principles (for an extensive survey of range imaging sensors, see Besl [4]). Laser range scanners are promising because they can provide dense, accurate range data at high bandwidths.

Range scanning systems typically produce *range images*—rectangular grids of distances from the sensor to the object being scanned. If the sensor and object are fixed, only objects that are “point viewable” can be fully digitized. More sophisticated systems, such as those produced by Cyberware Laboratory, Inc., are capable of digitizing cylindrical objects by rotating either the sensor or the object. To adequately scan objects of more complicated topological type, such as the object depicted in Figure 1.1a (a surface of genus 3), multiple range images must be generated. Although the resulting data contains structure within each range image, merging the data to reconstruct a useful surface representation is a non-trivial task [38, 72].

3D scanning applications The development of fast, inexpensive 3D scanning systems opens up a vast range of applications, including:

Reverse engineering: Computer-aided designs often begin with a physical object. Many industries have a large catalog of traditional parts, created without CAD tools, for which there may not even be paper engineering drawings, and which must be incorporated or modified into new designs.

Industrial design: Current CAD systems are far from providing the tactile and visual advantages of traditional media such as wood and clay. CAD systems are used to design three-dimensional shapes, but, with few exceptions, only two dimensional input and output devices are used. Certain subtle but important features—such as the facial features in Figure 4.15f—can be difficult to achieve. 3D scanning allows the transfer of manually sculptured shapes into CAD systems.

Analysis and simulation: Digital descriptions can be analyzed and used in computer simulations. It then becomes possible, for example, to calculate the drag coefficient of a car body sculpted in clay by the designer.

Populating virtual environments: Creating virtual environments simulating the physical world around us requires models for the objects populating this world. Current virtual realities tend to have a cartoon-like character partly due to lack of realistic models of everyday things. 3D scanning can be used to efficiently obtain such models.

3D faxing: The emerging technologies of *solid free-form fabrication* (SFF), which allow the quick prototyping of 3D objects, together with 3D scanners and efficient surface reconstruction algorithms, may allow "3D faxing"—the scanning, transmission, and re-instantiation of 3D shape.

To fully realize the potential of 3D scanning, it is essential to develop general, automatic, efficient, and robust surface reconstruction algorithms for converting the data points that 3D scanners produce into useful models.

1.2.2 Contour data

Another application area involves the reconstruction of surfaces from contours. In many medical studies it is common to slice biological specimens into thin layers with a microtome. The outlines of the structures of interest are then digitized to create a stack of contours. In manufacturing, similar stacks of contours are also produced by cross-section CAT scans of mechanical parts. The *surfaces from contours* problem attempts to recover the three-dimensional structures from the stacks of parallel two-dimensional contours. Although the problem has received a good deal of attention [8, 39, 40], there remain severe limitations with current methods. Perhaps foremost among these is the difficulty of automatically dealing with branching structures. While algorithms addressing the general surface reconstruction problem may not be as successful as methods specialized for contour data, they need not consider such special cases.

In a related problem, ultrasound sensing is used to study the shape of the heart [28]. Contour images of the heart are obtained after insertion of a probe into the esophagus. Unlike the microtome data, the ultrasound contours are not parallel. Moreover, the probe is able to generate many sets of contours from different directions and from different positions. Algorithms for solving the surfaces from contours problem cannot be easily applied to this type of data.

1.2.3 Surface sketching

A number of researchers, including Schneider [59] and Eisenman [19], have investigated the creation of curves in $\mathbf{R}^2$ by tracing the path of a stylus or mouse as the user sketches the desired shape. Sachs *et al.* [54] describe a system, called 3-Draw, that permits the creation of free-form curves in $\mathbf{R}^3$ by recording the motion of a stylus fitted with a Polhemus sensor. This can be extended to the design of free-form surfaces by ignoring the order in which positions are recorded, allowing the user to move the stylus arbitrarily back and forth over the surface. The problem is then to construct a surface representation faithful to the unordered collection of points.

1.3 Previous work

Previous surface reconstruction algorithms addressing these application areas have typically been crafted on a case by case basis to exploit additional knowledge such as topological type of the surface, structure in the data, orientation information, or absence of noise.

1.3.1 Algorithms assuming fixed topological type

A common restriction of surface reconstruction methods is that they assume that the topological type of the surface is known a priori.

Parametric reconstruction methods Parametric methods represent the reconstructed surface as an embedding $f(\Lambda) \subset \mathbf{R}^3$ of a 2-dimensional parameter domain Λ. Previous work has concentrated on domain spaces with simple topological type, i.e. the plane and the sphere. Hastie and Stuetzle [27], and Bolle and Vemuri [7] discuss reconstruction of surfaces by a topological embedding $f(\Lambda)$ of a planar region Λ into $\mathbf{R}^3$. Brinkley [9] considers the reconstruction of surfaces that are slightly deformed spheres, and thus chooses Λ to be an approximation to a sphere. Schmitt *et al.* [57, 58] fit embeddings of cylinders. Goshtasby [25] works with embeddings of cylinders and tori.

Since the domain Λ and the surface $f(\Lambda)$ are homeomorphic, parametric reconstruction methods inherently require knowledge of the topological type of the surface. Moreover, to converge correctly, they also require an initial embedding $f_0(\Lambda)$ that is “sufficiently close” to U. Equivalently, they assume a “good” initial parameterization of the points X in Λ. This presents a problem since such an initial parameterization may be difficult to construct.

There is also extensive literature on smooth interpolation of triangulated data of arbitrary topological type using parametric surface patches; see Lounsbery *et al.* [36] for a survey. These schemes are designed to interpolate sparse data, rather than to fit dense, noisy point sets of the type obtained from range scanners. Some more recent examples include Veltkamp [73] and Szeliski *et al.* [67].

Function reconstruction Terms like "surface fitting" appear in reference to two distinct classes of problems: surface reconstruction and function reconstruction. The goal of surface reconstruction was stated earlier. The goal of function reconstruction may be stated as follows: Given a surface D, a set $\{\mathbf{x}_i \in D\}$, and a set $\{y_i \in \mathbf{R}\}$, determine a function $f : D \to \mathbf{R}$, such that $f(\mathbf{x}_i) \approx y_i$.

The domain surface D is most commonly a plane, in which case the problem is a standard one considered in approximation theory. The case where D is a sphere has also been treated extensively . Foley [22] defines radial basis functions centered on points scattered over a sphere. Schudy and Ballard [61, 62] use spherical harmonics to fit a surface as a function over a spherical domain. Sclaroff and Pentland [65] describe a hybrid implicit/parametric surface fitting method that involves fitting a function over a deformed superquadric. Some recent work under the title *surfaces on surfaces* addresses the case when D is a general curved surface such as the skin of an airplane [2, 46].

Function reconstruction methods can be used for surface reconstruction in simple, special cases, where the surface to be reconstructed is, roughly speaking, the graph of a function over a *known* surface D. It is important to recognize just how limited these special cases are—for example, not every surface homeomorphic to a sphere is the graph of a function over the sphere. The point is that function reconstruction must not be misconstrued to solve the general surface reconstruction problem.

Constriction methods Constriction methods attempt to find a mesh interpolating a set of data points. They first construct a 3-dimensional triangulation T_0 of the points (T_0 is often chosen to be the Delaunay Triangulation).[3] The boundary $B(T_0)$ of the triangulation is a mesh that corresponds to the convex hull of the points. Since many surfaces are not convex, $B(T_0)$ in general only interpolates a subset of the points. Therefore, they apply an iterative constriction technique that, from a triangulation T_i, creates a new triangulation

[3] Intuitively, a 3-dimensional triangulation consists of a set of tetrahedra pasted together along their faces.

T_{i+1} by removing a tetrahedron adjacent to $B(T_i)$. As T progressively shrinks, the boundary mesh $B(T)$ interpolates an increasing number of data points. In deciding which tetrahedron to remove next from T_i, O'Rourke [48] uses a criterion based on minimal area of $B(T_{i+1})$, and Veltkamp [73] uses a criterion based on maximal interior angle of $B(T_i)$. However, these methods are restricted in that they always produce a closed surface of genus zero.

1.3.2 Algorithms exploiting structure information

Many surface reconstruction algorithms exploit structure in the data. For instance, algorithms solving the surfaces from contours problem (Section 1.2.2) make heavy use of the fact that the data points are organized into contours, and that the contours lie in parallel planes.

Similarly, algorithms to reconstruct surfaces from multiple range images typically exploit the adjacency relationship of the data within each range image. Merriam [38] suggests two methods for merging range images: a *virtual milling* technique that intersects polyhedra constructed from the different range images, and a pruning technique that first constructs the 3D Delaunay Triangulation of the points and then prunes away tetrahedra "exposed" in the various range images. Turk and Levoy [72] describe a *mesh zippering* approach, in which overlapping surfaces (the range images) are "stitched" together.

These approaches have the drawback that they must deal with special cases using ad hoc techniques. It is therefore difficult to apply them to similar but different problems. For instance, methods solving the surfaces from contours problem cannot be used when presented with several sets of intersecting contours.

1.3.3 Algorithms exploiting orientation information

Knowledge of the orientation of the surface at each data point is extremely valuable in surface reconstruction. In fact, automatically determining such orientation is one of the main challenges in our method, as we shall see in Section 2.2.3. Many previous reconstruction methods assume that such orientation information is supplied with the data.

When the data points X are obtained from volumetric data, the gradient of this data can provide orientation information that helps guide the reconstruction. For instance, Moore and Warren [42] fit a piecewise polynomial implicit surface to a set of points, and make use of auxiliary volumetric samples (off the surface) to assign correct orientations to the surface pieces and to prevent spurious surface sheets. Likewise, Miller *et al.* [41] describe a procedure for fitting meshes to isosurfaces of volumetric data, and make use of volumetric information.

Other reconstruction procedures assume that each data point $\mathbf{x}_i$ is also provided with a normal vector $\hat{\mathbf{n}}_i$. For example, algorithms for reconstructing surfaces from range data typically exploit the fact that each surface point $\mathbf{x}_i$ is known to be visible from the sensor, and make use of these direction vectors in orienting the surface. Szeliski and Tonnesen [68] reconstruct a surface using an optimization problem involving oriented particles. By local interaction, these particles align themselves on a 2-dimensional manifold. The initial orientations of the particles is crucial to the success of their method, and must be specified as input. Muraki [43] fits an implicit function f that is a linear combination of three-dimensional Gaussian kernels with different means and spreads. His goodness-of-fit function measures how close the values of f at the data points are to zero, and how well the direction of the gradient of f matches normals $\hat{\mathbf{n}}_i$ estimated from the data.

1.3.4 Algorithms for triangulating noise-free data

Some recent computational geometry methods come close to addressing the general surface reconstruction problem. They find meshes of arbitrary topological type that interpolate sets of unorganized points. Since they interpolate the data, their main limitation is that they require the data to be noise-free.

Edelsbrunner and Mücke [18] generalize the notion of convex hull to that of *alpha hull* (α-hull). The convex hull of a set X can be thought of as the complement of the union of all half-spaces not containing X. The α-hull is defined to be the complement of the union of all α-spheres (spheres of radius α) not containing X. Thus, the convex hull equals the α-hull with $\alpha = \infty$. Edelsbrunner and Mücke also introduce the *alpha shape* (α-shape), obtained by substituting simplicial elements (segments and triangles) for the curved boundary elements of the α-hull. More recently, Edelsbrunner [17] has extended this notion to *weighted alpha shapes*, in which the data points can be assigned scalar weights to cope with non-uniform samplings.

The α-shape approach has great potential in addressing the general surface reconstruction problem. However, if the sample X is noisy, or if the underlying surface U is not sufficiently smooth, the α-shape of X will in general have finite thickness, and not be, as one would desire, a 2-dimensional manifold. It may be possible, as a post-process, to "flatten" such an α-shape into a surface.

Favardin [21] presents a modified "gift-wrapping" algorithm for triangulating a set of points. Gift-wrapping is a standard algorithm from computational geometry for computing the convex hull of a set of points. It creates a triangulation by successively pivoting planes about boundary edges of the triangulation until these plane encounter other points. Favardin modifies the standard procedure to allow the creation of non-convex surfaces by only considering points in a local neighborhood of the pivoting edge. Favardin also describes a heuristic for detecting and dealing with surface boundaries.

There appears to be a close connection between the two previous methods. By appropriately defining the local neighborhood of Favardin's modified gift-wrapping algorithm, we conjecture that it can in fact generate the boundary of the α-shape. Specifically, the neighborhood should be defined as the union of all α-spheres incident to both vertices of the edge.

1.3.5 Implicit surface fitting algorithms

Several methods fit algebraic implicit surfaces (zero sets of polynomial functions) to sets of points [50, 69]. However, the intent of these methods is not to reconstruct surfaces but to either recognize objects or infer their orientations in a scene. These fitting methods cannot be used directly for surface reconstruction because the topological type of algebraic surfaces is highly unpredictable; in most cases, fitting an algebraic surface to a set of points results in numerous surface sheets that happen to pass near the data but only connect up far away (e.g. Figure 1.2).

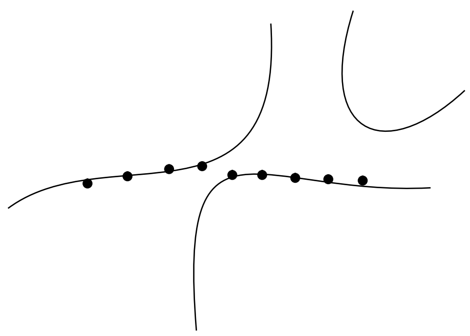


Figure 1.2: Problem with fitting an algebraic surface.

One approach to controlling the topological type of an implicit surface is to triangulate space and define a *piecewise* algebraic function of low degree over the resulting simplices.

Moore and Warren [42] use a piecewise polynomial function of degree 3. By keeping the degree low and using additional volumetric samples (Section 1.3.3), they are able to avoid extraneous surface sheets when the data is dense and nearly linear within each simplex. Taubin and Ronfard [70] use a piecewise linear function in their *implicit simplicial model* representation. The topological type of their surface is easy to predict, since the degree 1 algebraic surface can have at most one surface sheet within each simplex. Using this representation, they have designed a curve reconstruction method that should in principle generalize to surface reconstruction. A unique aspect of their method is that they infer both the topological type of the curve and its geometry in a single process. Also, the use of an implicit representation guarantees that the curves they generate never intersect themselves.

1.4 Overview of the surface reconstruction method

As seen in the previous section, surface reconstruction algorithms have typically been designed to exploit additional knowledge in specific problem instances.

In contrast, our approach is to pose a unifying general problem. This approach has both theoretical and practical merit. On the theoretical side, abstracting to a general problem often sheds light on the truly critical aspects of the problem. On the practical side, a single algorithm that solves the general problem can be used to solve any specific problem instance.

We have developed a method for automatically reconstructing an accurate, concise piecewise smooth surface S from a set of of points X, where

- X is an unorganized, noisy sample of an unknown surface U;
- the unknown surface U can have arbitrary topological type (including boundaries), and may contain tangent plane discontinuities such as creases and corners;
- no other information, such as structure in the data or orientation information, is provided.

A major difficulty in this general surface reconstruction problem is that the topological type of U is not known a priori and must be inferred from the points. To tackle this difficulty, we have partitioned the reconstruction problem: we first robustly determine the topological type of the surface, and only then concern ourselves with the accuracy and conciseness of the model. Our reconstruction method consists of three successive phases, as illustrated in Figures 1.3 and 1.4.

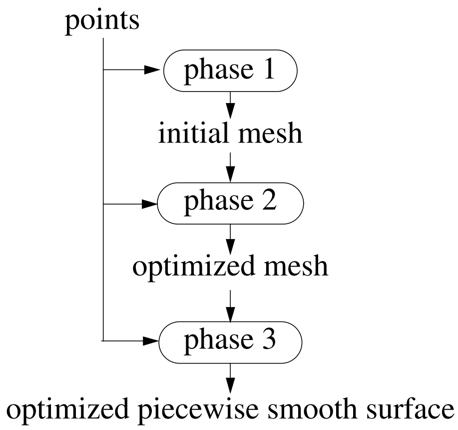
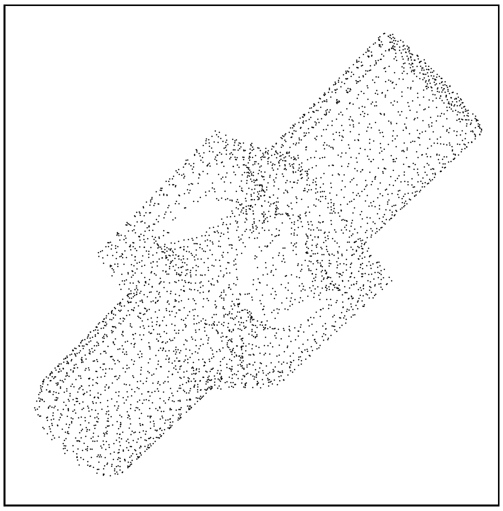


Figure 1.3: The three phases of our surface reconstruction method.

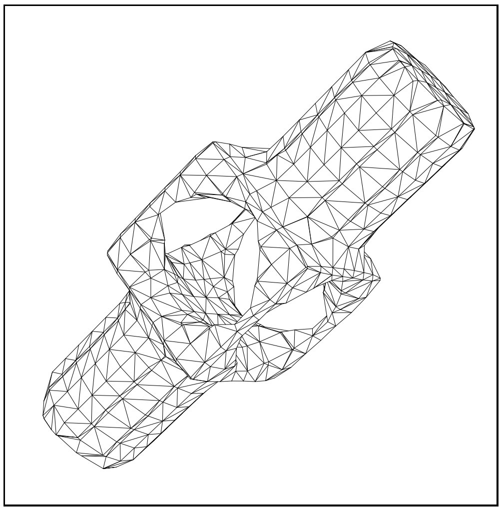
Phase 1: initial surface estimation (Chapter 2 and [4]): From an unorganized set of points (Figure 1.4a), phase 1 constructs an initial dense mesh (Figure 1.4b). The goal of this phase is to determine the topological type of the surface, and to produce an initial estimate of its geometry.

Phase 2: mesh optimization (Chapter 3 and [3, 2]): Starting with the dense mesh created in phase 1, phase 2 reduces the number of faces and improves the fit to the data points (Figure 1.4c). We cast this problem as optimization of an energy function that explicitly models the trade-off between the competing goals of accuracy and conciseness. The free variables in the optimization are the number of vertices in the mesh, their connectivity, and their positions.

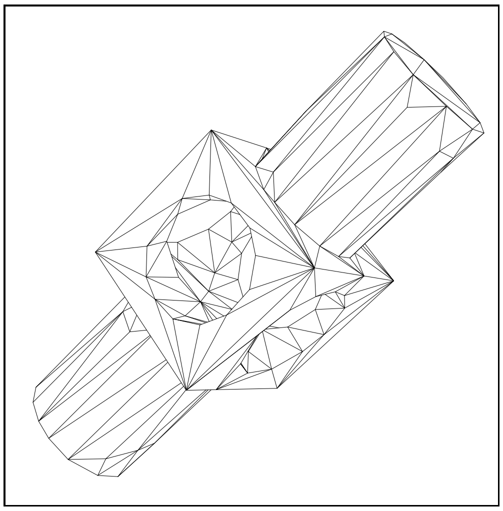
Phase 3: piecewise smooth surface optimization (Chapter 4 and [1]): In phase 3, the surface representation is changed from a piecewise linear one (meshes) to a piecewise smooth one. We introduce of a new piecewise smooth representation based on subdivision. These surfaces are ideal for surface reconstruction, as they are simple to implement, can model sharp features concisely, and can be fit using an extension of the phase 2 optimization algorithm.



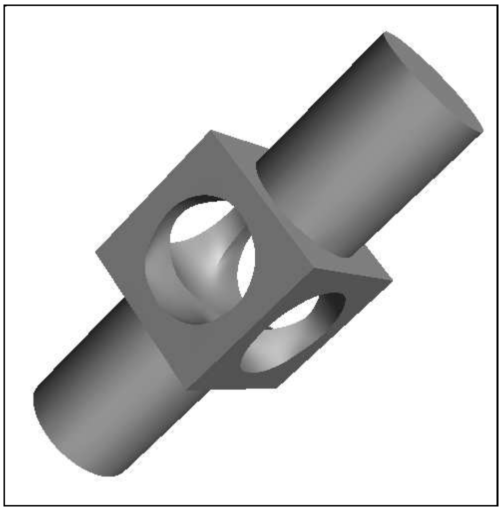
(a) Unorganized points X



(b) Result of phase 1: initial dense mesh



(c) Result of phase 2: optimized mesh



(d) Result of phase 3: piecewise smooth surface

Figure 1.4: Example of the three phases of the surface reconstruction method.

Starting with the optimized mesh produced in phase 2, phase 3 fits an accurate, concise piecewise smooth subdivision surface (Figure 1.4d), again by optimizing an energy function that trades off accuracy and conciseness. In addition to varying the geometry and size of the surface representation, phase 3 also optimizes over the number and locations of sharp features. The automatic detection and recovery of sharp features in the surface is an essential part of phase 3.

Phase 2 could in principle be eliminated, but has proven useful for two reasons: first, it is computationally more efficient to optimize over a piecewise linear surface in the early stages of optimization, and second, initial estimates of sharp features are much more robust when obtained from the phase 2 mesh.

1.5 Contributions

The principal contributions of this thesis are:

- It presents a robust algorithm (phase 1) for reconstructing surfaces of arbitrary topological type from unorganized points. From the data points, the algorithm automatically infers the topological type of the surface (including the presence of boundary curves).

- It presents an algorithm (phase 2, mesh optimization) for fitting a mesh of arbitrary topological type to a set of data points. The fitting problem is cast as an energy minimization problem over all meshes of a given topological type, with an energy function that directly represents the trade-off of accuracy and conciseness.

- It introduces a new representation for piecewise smooth surfaces of arbitrary topological type. The new surface representation generalizes Loop's subdivision surface scheme [33] by introducing additional subdivision rules that allow the modeling of sharp surface features such as creases and corners.

- It presents an algorithm (phase 3) for fitting piecewise smooth surfaces to sets of points. The algorithm is a generalization of mesh optimization to piecewise smooth subdivision surfaces, with the addition of a new set of free variables, the set of sharp surface features. By casting the fitting problem as an optimization over all piecewise smooth subdivision surfaces of a given topological type, the algorithm is able to find accurate, concise piecewise smooth surfaces wherein sharp features are recovered automatically.

- It demonstrates how the optimization algorithms of phases 2 and 3 can be used effectively for the approximation of surfaces by piecewise linear and piecewise smooth models (Chapter 5). By casting surface approximation as a global optimization problem with an energy function that directly measures deviation of the approximation from the original surface, we obtain models with excellent accuracy to conciseness trade-offs. One commonly encountered instance of surface approximation is the problem of mesh simplification—the accurate approximation of a dense mesh by a more concise one.

1.6 Overview of thesis

The three phases of the surface reconstruction method are discussed and demonstrated in Chapters 2, 3, and 4, respectively. The application of this work to the related problem of surface approximation is presented in Chapter 5. Finally we analyze the shortcomings of the surface reconstruction method as a whole and highlight directions for future work in Chapter 6.

1.7 Terminology and notation

By a *surface* we mean a "compact, connected, orientable two-dimensional manifold, possibly with boundary, embedded in $\mathbf{R}^3$" (cf. O'Neill [47]). A surface without boundary will be called a *closed surface*. If we want to emphasize that a surface possesses a non-empty boundary, we will call it a *bordered surface*. Similarly, a *curve* will refer to a "compact, connected one-dimensional manifold, possibly with boundary, embedded in $\mathbf{R}^d$". We use $||\mathbf{x}||$ to denote the Euclidean length of a vector $\mathbf{x}$, and we use $d(X, Y)$ to denote the Hausdorff distance between the sets of points X and Y (the Hausdorff distance is simply the Euclidean distance between the two closest points of X and Y).

1.7.1 Mesh representation

Intuitively, a *mesh* is a piecewise linear surface, consisting of triangular faces pasted together along their edges. For our purposes it is important to maintain the distinction between the connectivity of the mesh and its geometry. Formally, a mesh M is a pair (K, V), where: K is a *simplicial complex* representing the connectivity of the vertices, edges, and faces, thus determining the topological type of the mesh; $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$, $\mathbf{v}_i \in \mathbf{R}^3$ is a set of vertex positions defining the shape of the mesh in $\mathbf{R}^3$ (its geometric realization).

Simplicial complex K
vertices: $\{1\}$, $\{2\}$, $\{3\}$
edges: $\{1, 2\}$, $\{2, 3\}$, $\{1, 3\}$
faces: $\{1, 2, 3\}$

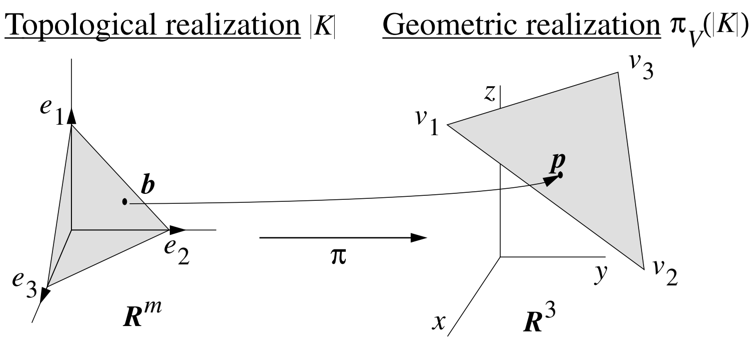


Figure 1.5: Mesh representation: an example of a mesh consisting of a single face.

A simplicial complex K consists of a set of simplices $\{1\}, \ldots, \{m\}$, called vertices, together with a set of simplices formed by unions of these vertices, such that every non-empty subset of a simplex in K is again a simplex in K (cf. Spanier [66]). The 1-simplices $\{i, j\} \in K$ are called edges, and the 2-simplices $\{i, j, k\} \in K$ are called faces.

A geometric realization of a mesh as a surface in $\mathbf{R}^3$ can be obtained as follows. For a given simplicial complex K, form its *topological realization* $|K|$ in $\mathbf{R}^m$ by identifying the vertices $\{\{1\}, \ldots, \{m\}\}$ with the standard basis vectors $\{\mathbf{e}_1, \ldots, \mathbf{e}_m\}$ of $\mathbf{R}^m$. For each simplex $s \in K$ let $|s|$ denote the convex hull of its vertices in $\mathbf{R}^m$, and let $|K| = \cup_{s \in K} |s|$. Let $\pi : \mathbf{R}^m \to \mathbf{R}^3$ be the linear map that sends the i-th standard basis vector $\mathbf{e}_i \in \mathbf{R}^m$ to $\mathbf{v}_i \in \mathbf{R}^3$ (see Figure 1.5).

The *geometric realization* of M is the image $\pi_V(|K|)$, where we write the map as π_V to emphasize that it is fully specified by the set of vertex positions $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$. The map π_V is called an *embedding* if it is 1-1, that is if $\pi_V(|K|)$ is not self-intersecting. Only a restricted set of vertex positions V result in π_V being an embedding.

If π_V is an embedding, any point $\mathbf{p} \in \pi_V(|K|)$ can be parameterized by finding its unique pre-image on $|K|$. The vector $\mathbf{b} \in |K|$ with $\mathbf{p} = \pi_V(\mathbf{b})$ is called the *barycentric coordinate vector* of $\mathbf{p}$ (with respect to the simplicial complex K). Note that barycentric coordinate vectors are convex combinations of standard basis vectors $\mathbf{e}_i \in \mathbf{R}^m$ corresponding to the vertices of a face of K. Any barycentric coordinate vector has at most three non-zero entries; it has only two non-zero entries if it lies on an edge of $|K|$, and only one if it is a vertex.

Our implementation represents the simplicial complex using a half-edge data structure (cf. Weiler [75]). Points stored at vertices of this data structure determine the geometric realization of the mesh.

1.7.2 Neighborhoods on meshes

It is also useful to define neighborhoods on a simplicial complex. For this purpose, we define a *face* of a simplex s to be any subset of s, and define the *simplicial neighborhood* of a set of simplices $J \subset K$ as the set of all simplices s such that s is a face of a simplex $s' \in K$ that has a face s'' in J:

$$\mathrm{nbhd}(J; K) = \{s \in K \,:\, \exists s'' \in J,\ s' \in K \text{ such that } s'' \cup s \subset s'\}.$$

Figure 1.6 shows examples of $\mathrm{nbhd}(J; K)$ where J consists of a single simplex. Finally, we inductively define $\mathrm{nbhd}^k(J; K) = \mathrm{nbhd}(\mathrm{nbhd}^{k-1}(J; K))$, $k > 1$.

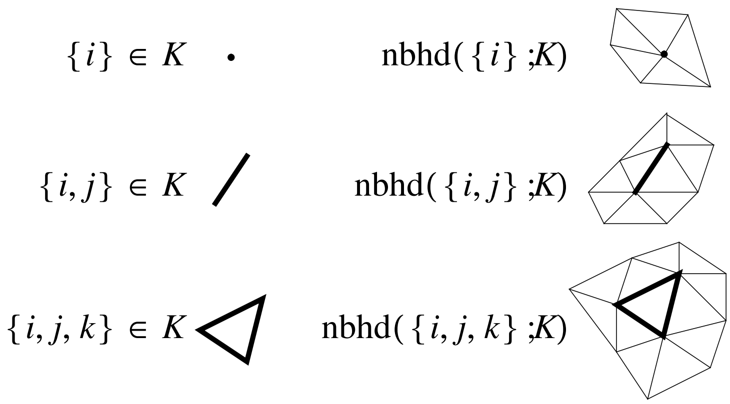


Figure 1.6: Simplices and their corresponding simplicial neighborhoods.

Chapter 2

PHASE 1: INITIAL SURFACE ESTIMATION

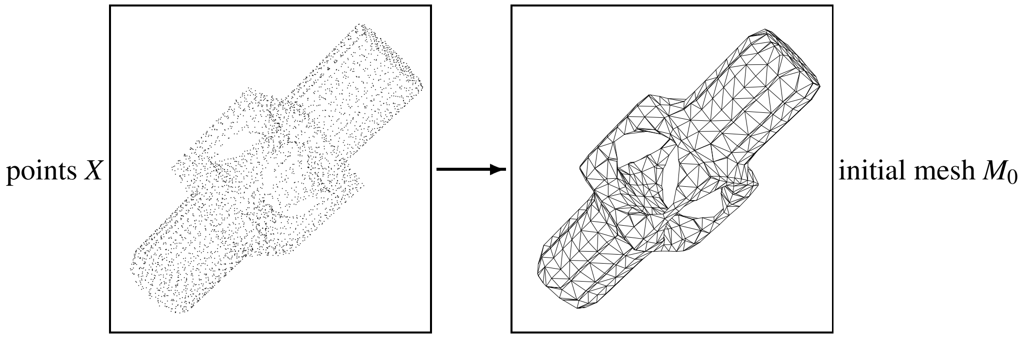


Figure 2.1: Phase 1: estimation of an initial mesh from a set of points.

2.1 Introduction

Phase 1 of the surface reconstruction procedure constructs an initial estimate for the surface. From an unorganized set of points $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ sampled from some unknown surface U, phase 1 creates a mesh M_0 that approximates U. A major difficulty is that the topological type of U is unknown and must be inferred from the points. The goal of phase 1 is to provide a robust method for correctly inferring this topological type. At the same time, phase 1 creates a geometric approximation to the surface, albeit a crude one. In the example of Figure 2.1, from a set of 4,102 points, phase 1 creates an initial mesh of the correct topological type (closed surface of genus 3). As is evident in the example, the approximating mesh M_0 typically has an excessive number of faces and is a poor fit to the data. Phases 2 and 3 are responsible for improving the accuracy and the conciseness of the surface.

In contrast to previous surface reconstructions schemes (Section 1.3), the phase 1 reconstruction algorithm makes relatively few assumptions about the set of points X and the underlying surface U from which these were sampled:

- the points may be noisy;
- no structure is assumed in the points;
- no information is required at each point beyond its (x,y,z) coordinates;
- the surface U may have arbitrary topological type (including boundaries);
- this topological type is not known a priori;
- the surface U is not assumed to be smooth.

We do require U to be a manifold (a non-intersecting surface), and at present require it to be orientable. These are not severe restrictions in practice since surfaces that are boundaries of physical objects satisfy these requirements.

Of course, if the only knowledge of U the algorithm is given is the finite set of sample points X, a correct surface reconstruction cannot generally be guaranteed, since the topological type of U cannot be deduced. Additional assumptions must therefore be made concerning the relationship between the surface U and the process that created the sample X. To be practically useful, these assumptions must be sufficiently general to be widely applicable, but sufficiently concrete to allow the algorithm to use them effectively. We make two main assumptions: one concerns the sampling process, and the other concerns the size of features in U.

Assumptions on the sampling process To capture the error in most sampling processes, each of the points $\mathbf{x}_i \in X$ is assumed to be of the form $\mathbf{x}_i = \mathbf{y}_i + \mathbf{e}_i$, where $\mathbf{y}_i \in U$ is a point on the unknown surface U and $\mathbf{e}_i \in \mathbf{R}^3$ is an error vector. Such a sample X is called *δ-noisy* if $||\mathbf{e}_i|| \leq \delta$ for all i. Of course, it is impossible to recover features of U in regions where insufficient sampling has occurred. In particular, if U is a bordered surface, such as a sphere with a disc removed, it is impossible to distinguish holes in the sample from holes in the surface. To capture the intuitive notion of sampling density we need to make another definition: Let $Y = \{\mathbf{y}_1, \ldots, \mathbf{y}_n\} \subset U$ be a (noiseless) sample of a surface U. The sample Y is said to be *ρ-dense* if any sphere with radius ρ and center in U contains at least one sample point in Y. A δ-noisy sample $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbf{R}^3$ of a surface U is said to be

ρ-dense if there exists a noiseless ρ-dense sample $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\} \subset U$ such that $\mathbf{x}_i = \mathbf{y}_i + \mathbf{e}_i$, $\|\mathbf{e}_i\| \leq \delta$, $i = 1, \ldots, n$.

With these definitions, let us consider when, given the set of sample points X, an arbitrary point $\mathbf{p} \in \mathbf{R}^3$ could be a point of U. If there is no noise, we can deduce that $\mathbf{p}$ with $d(\mathbf{p}, X) > \rho$ cannot be a point of U since that would violate X being ρ-dense. Intuitively, the sample points do not leave holes of radius larger than ρ. If the sample is δ-noisy, the radius of the holes may increase, but by no more than δ. We therefore conclude that, given a ρ-dense, δ-noisy sample X of U, a point $\mathbf{p}$ cannot be a point of U if $d(\mathbf{p}, X) > \rho + \delta$.

Assumptions on the size of features in U Features of U that are small compared to either ρ or δ are obviously not recoverable. While it may be acceptable to leave out small details of U from the reconstruction, it is critical that the topological type of U be inferred correctly. To realize this, we assume that no two "sheets" of U come "too close together" (Figure 2.2). Specifically we assume that points sampled from two different sheets of surface are separated by a distance of at least $\rho + \delta$. Otherwise there would be no way to resolve the underlying topological type of U. Taking sampling noise into account, an equivalent condition is that no two sheets of U may come within distance $\rho + 3\delta$ of each other.

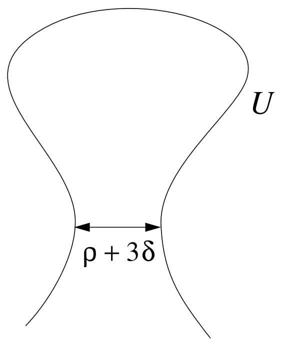


Figure 2.2: Assumption on the size of features in U.

An approximation of $\rho + \delta$ is provided to the algorithm as a user-specified parameter. A value for the noise magnitude δ can be estimated in most applications (e.g., the accuracy of the laser scanner). Similarly, analysis of the scanning procedure can also provide an estimate for the sampling density ρ.

2.2 Description of the algorithm

2.2.1 Overview of the algorithm

From a set of data points $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ assumed to be on or near an unknown surface U, the phase 1 algorithm generates a mesh approximating U.

The key idea in phase 1 is to estimate from X the signed distance to U. The signed distance from an arbitrary point $\mathbf{p} \in \mathbf{R}^3$ to a known closed surface U is defined as $d_U(\mathbf{p}) = s(\mathbf{p}) \cdot d(\mathbf{p}, U)$, where $s(\mathbf{p}) = \pm 1$, depending on which side of the surface $\mathbf{p}$ lies. If U is a bordered surface, a continuous signed distance can be defined if one stays within a tubular neighborhood D of the surface[1]. For our purposes, it is important to note that knowing the signed distance function d_U is equivalent to knowing the surface U; an implicit representation for U is given by the zero set $Z(d_U) = \{\mathbf{p} : d_U(\mathbf{p}) = 0\}$. Although we know neither U nor d_U, our strategy will be to first estimate d_U from the data points and then extract an approximation of its zero set.

More concretely, the phase 1 algorithm consists of two stages (Figure 2.3). The first stage defines a function $\tilde{d}_U : D \to \mathbf{R}$, where $D \subset \mathbf{R}^3$ is a region near the data points, such that $\tilde{d}_U$ estimates the signed distance d_U. To handle bordered surfaces, the algorithm leaves $\tilde{d}_U(\mathbf{p})$ undefined when $\mathbf{p} \notin D$. Since $\tilde{d}_U$ estimates d_U, its zero set $Z(\tilde{d}_U) = \{\mathbf{p} : \tilde{d}_U(\mathbf{p}) = 0\}$ is our estimate for U. In the second stage we use a contouring algorithm to extract an approximation to $Z(\tilde{d}_U)$ in the form of a mesh.

Although the *unsigned* distance function $|d_U|$ would be easier to estimate, zero is not a regular value of $|d_U|$. Zero is, however, a regular value of d_U, and the implicit function theorem thus guarantees that the set $Z(d_U)$ is a manifold.

The key ingredient to estimating the signed distance function is to associate an oriented plane with each of the data points. These estimated *tangent planes* serve as local linear approximations to the surface. Although the construction of the tangent planes is relatively simple, the selection of their orientations so as to define a globally consistent orientation for the surface is one of the major obstacles facing the algorithm. These oriented tangent planes, shown in Figure 2.4a, are then used to define the signed distance function to the surface. An example of the mesh obtained by contouring the zero set of the signed distance function is shown in Figure 2.1. The next several sections develop in more detail the successive steps of the algorithm.

---

[1] Informally, the tubular neighborhood of a surface S is the set of points $\mathbf{p}$ such that $d(\mathbf{p}, S)$ is small.

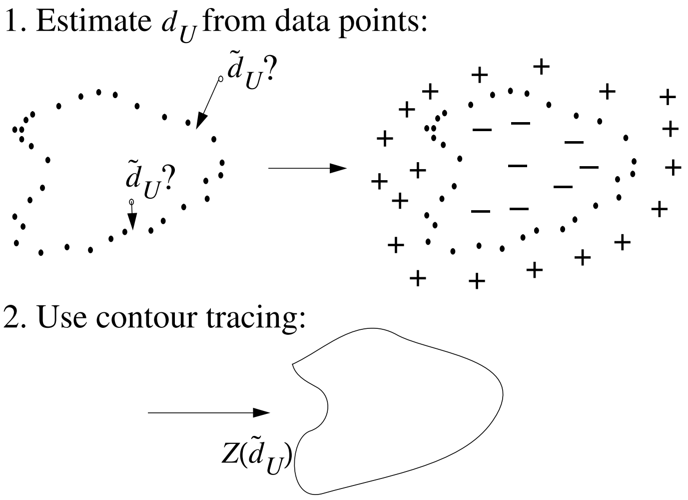


Figure 2.3: Illustration of the two stages in the phase 1 algorithm.

2.2.2 Tangent plane estimation

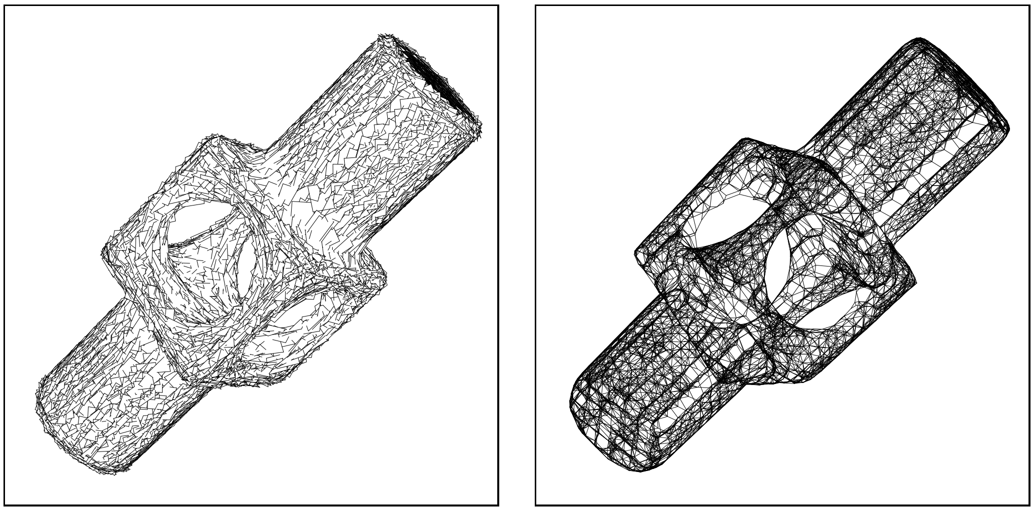
The first step toward estimating a signed distance function is to compute an oriented tangent plane for each data point (Figure 2.4a). The tangent plane $Tp(\mathbf{x}_i)$ associated with the data point $\mathbf{x}_i$ is represented as a point $\mathbf{o}_i$, called the center, together with a unit normal vector $\hat{\mathbf{n}}_i$. The signed distance of an arbitrary point $\mathbf{p} \in \mathbf{R}^3$ to $Tp(\mathbf{x}_i)$ is defined to be $\mathrm{dist}_i(\mathbf{p}) = (\mathbf{p} - \mathbf{o}_i) \cdot \hat{\mathbf{n}}_i$. The center and normal for $Tp(\mathbf{x}_i)$ are determined by gathering together the group of points of X within distance $\rho + \delta$ of $\mathbf{x}_i$ (where ρ and δ are parameters estimating the sampling density and noise); this set is denoted by $Nbhd(\mathbf{x}_i)$ and is called the neighborhood of $\mathbf{x}_i$. The center and unit normal are computed so that the plane $\{\mathrm{dist}_i(\mathbf{p}) = 0\}$ is the least squares best fitting plane to $Nbhd(\mathbf{x}_i)$. That is, the center $\mathbf{o}_i$ is taken to be the centroid of $Nbhd(\mathbf{x}_i)$, and the normal $\hat{\mathbf{n}}_i$ is determined using *principal component* analysis. To compute $\hat{\mathbf{n}}_i$, the covariance matrix of $Nbhd(\mathbf{x}_i)$ is formed. This is the symmetric 3×3 positive semi-definite matrix

$$\mathrm{CV}_i = \sum_{\mathbf{y} \in Nbhd(\mathbf{x}_i)} (\mathbf{y} - \mathbf{o}_i) \otimes (\mathbf{y} - \mathbf{o}_i)$$

where $\otimes$ denotes the outer product vector operator.[2] If $\lambda_i^1 \geq \lambda_i^2 \geq \lambda_i^3$ denote the eigenvalues of CV_i associated with unit eigenvectors $\mathbf{v}_i^1, \mathbf{v}_i^2, \mathbf{v}_i^3$, respectively, $\hat{\mathbf{n}}_i$ is chosen to be either

[2] If **a** and **b** have components a_i and b_j respectively, then the matrix $\mathbf{a} \otimes \mathbf{b}$ has $a_i b_j$ as its *ij*-th entry.

$\mathbf{v}_i^3$ or $-\mathbf{v}_i^3$. The selection determines the orientation of the tangent plane, and it must be done so that nearby planes are "consistently oriented".



(a) Estimated tangent planes $Tp(\mathbf{x}_i)$ (b) Riemannian Graph on $\mathbf{o}_i$

Figure 2.4: Estimated tangent planes and Riemannian Graph.

2.2.3 Consistent tangent plane orientation

Suppose two data points $\mathbf{x}_i, \mathbf{x}_j \in X$ are geometrically close. Ideally, when the data is dense and the surface is smooth, the corresponding tangent planes $Tp(\mathbf{x}_i) = (\mathbf{o}_i, \hat{\mathbf{n}}_i)$ and $Tp(\mathbf{x}_j) = (\mathbf{o}_j, \hat{\mathbf{n}}_j)$ are nearly parallel, i.e. $\hat{\mathbf{n}}_i \cdot \hat{\mathbf{n}}_j \approx \pm 1$. If the planes are consistently oriented, then $\hat{\mathbf{n}}_i \cdot \hat{\mathbf{n}}_j \approx +1$; otherwise, either $\hat{\mathbf{n}}_i$ or $\hat{\mathbf{n}}_j$ should be flipped. The difficulty in finding a consistent global orientation is that this condition should hold between all pairs of "sufficiently close" data points.

We can model this problem as graph optimization. Let the undirected graph $G = (V, E)$ contain a vertex $i \in V$ for each tangent plane $Tp(\mathbf{x}_i)$, and edges $(i, j) \in E$ connecting two tangent planes if their centers $\mathbf{o}_i$ and $\mathbf{o}_j$ are sufficiently close. Two tangent planes are deemed sufficiently close if the corresponding data points lie in each other's neighborhood, or equivalently, if $\|\mathbf{x}_i - \mathbf{x}_j\| < \rho + \delta$. This graph G (Figure 2.4b), which we call the *Riemannian Graph*, is thus constructed to encode geometric proximity of the tangent plane centers. (The graph may consist of more than one connected component if the underlying surface U is not connected, in which case the following algorithm is applied to each connected component of the graph.)

In the graph optimization problem, the cost on an edge $e = (i, j)$ encodes the degree to which $Tp(\mathbf{x}_i)$ and $Tp(\mathbf{x}_j)$ are consistently oriented, and is taken to be $w(e) = \hat{\mathbf{n}}_i \cdot \hat{\mathbf{n}}_j$. The problem is then to make a binary choice $b_i \in \{-1, 1\}$ for each vertex i, selecting tangent plane orientation $b_i\hat{\mathbf{n}}_i$, so as to maximize the cost metric

$$\sum_{(i,j)\in E} b_i b_j w(i, j) .$$

Unfortunately, this combinatorial optimization problem is NP-hard, as shown via a reduction from the NP-complete MAXCUT [23] decision problem.

Proof:

MAXCUT problem: Given graph $G = (V, E)$, weights $w(e) \in Z^+$ for each $e \in E$, and positive integer K, is there a partition of V into disjoint sets V_1 and V_2 such that the sum of the weights of the edges from E that have one endpoint in V_1 and one endpoint in V_2 is at least K?

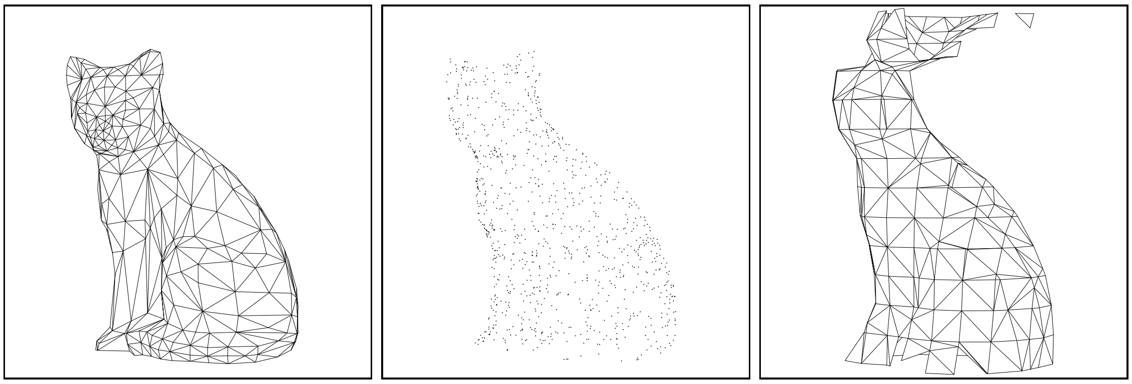
ORIENTATION problem: Given graph $G = (V, E)$ with weight $w(e) \in \mathbf{R}$ for each $e \in E$, determine an assignment $b(v) \in \{-1, +1\}$ for each $v \in V$ that maximizes

$$C_w(b) = \sum_{e=(v_1,v_2)\in E} b(v_1)b(v_2)w(e) .$$

To solve MAXCUT(G, w, K) using ORIENTATION, let $w' = -w$ (negate the weights on all edges), and call ORIENTATION(G, w') to obtain assignments b that maximize $C_{w'}(b)$. Partition V into two sets V_- and V_+ according to these assignments $b(v)$. Let W'_{--}, W'_{++}, and W'_{-+} represent the sum of the weights of edges with endpoints completely in V_-, completely in V_+, and in both, respectively. Note that the total weight of edges in the graph, $T = W'_{--} + W'_{++} + W'_{-+}$, is independent of assignments b, and that $C_{w'}(b) = W'_{--} + W'_{++} - W'_{-+}$. Thus, $C_{w'}(b) = T - 2W'_{-+}$, and maximizing $C_{w'}(b)$ is equivalent to minimizing W'_{-+}. Moreover, since the weights have been negated, $W_{-+} = -W'_{-+}$ and maximizing $C_{w'}(b)$ is equivalent to maximizing W_{-+}, the sum of the weights—in the MAXCUT problem—of edges with one endpoint in V_- and one endpoint in V_+. Since ORIENTATION provides a solution to this maximization problem, we can then trivially decide if there exists a partition of V with $W_{-+} > K$. Q.E.D.

Since our graph optimization problem is NP-hard, no solution method can guarantee finding its exact solution in reasonable time. One approach to approximating the solution is to use a simulated annealing method. For instance, Taubin and Ronfard [70] encounter a similar graph problem when determining signs in their implicit curve reconstruction method. They discover that this discrete optimization problem has a physically-based counterpart, the Ising model of large populations of particles with spin, for which simulated annealing schemes are well documented. Our approach is instead to use an efficient, greedy approximation algorithm. Although this greedy algorithm is not likely to perform well on arbitrary graphs, it has been successful on the Riemannian Graphs we have encountered.

A relatively simple-minded greedy algorithm to orient the tangent planes would be to arbitrarily choose an orientation for some plane, then "propagate" the orientation to neighboring planes in the graph. In practice, we have found that the order in which orientation is propagated is important. Figure 2.5c shows what may result when propagating orientation solely on the basis of geometric proximity; a correct reconstruction from the points in Figure 2.5b is shown in Figure 2.11l.



(a) Original mesh (U)

(b) Sampled points X

(c) Incorrect reconstruction resulting from naive orientation propagation

Figure 2.5: Importance of careful propagation of surface orientation.

Intuitively, we would like to choose an order of propagation that favors propagation from $Tp(\mathbf{x}_i)$ to $Tp(\mathbf{x}_j)$ if the unoriented planes are nearly parallel. This can be accomplished by assigning to each edge (i,j) in the Riemannian Graph the cost $1 - |\hat{\mathbf{n}}_i \cdot \hat{\mathbf{n}}_j|$. In addition to being non-negative, this assignment has the property that a cost is small if the unoriented tangent planes are nearly parallel. A favorable propagation order can therefore be achieved by traversing the *minimal spanning tree* (MST) of the resulting graph (shown in Figure 2.6). This order is advantageous because it tends to propagate orientation along directions of low curvature in the data, thereby largely avoiding ambiguous situations encountered when trying to propagate orientation across sharp edges (as at the tip of the cat's ears in Figure 2.5a).

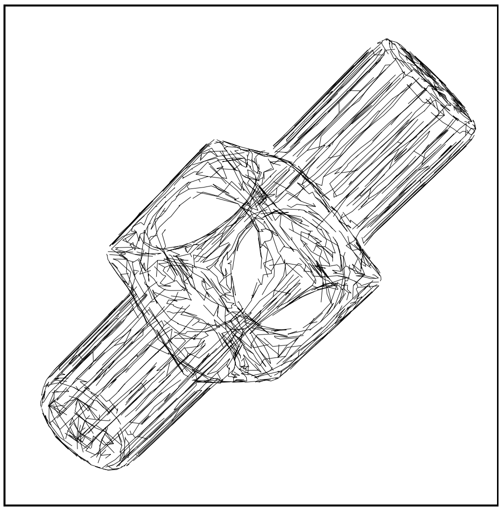


Figure 2.6: Orientation propagation path (minimal spanning tree).

To assign orientation to an initial plane, the unit normal of the tangent plane whose center has the largest z coordinate is made to point toward the $+z$ axis. This assignment is not critical but is convenient in obtaining predictable surface orientations. Then, rooting the minimal spanning tree at this initial node, the tree is traversed in depth-first order, each tangent plane being assigned an orientation consistent with that of its parent. That is, if during traversal, the current tangent plane $Tp(\mathbf{x}_i)$ has been assigned the orientation $\hat{\mathbf{n}}_i$ and $Tp(\mathbf{x}_j)$ is the next tangent plane to be visited, then $\hat{\mathbf{n}}_j$ is replaced with $-\hat{\mathbf{n}}_j$ if $\hat{\mathbf{n}}_i \cdot \hat{\mathbf{n}}_j < 0$.

The orientation algorithm described above has been used in all the examples and has produced correct orientations in all the cases we have run.

2.2.4 Signed distance function

The signed distance $d_U(\mathbf{p})$ from an arbitrary point $\mathbf{p} \in \mathbf{R}^3$ to a known closed surface U is the distance between $\mathbf{p}$ and the closest point $\mathbf{z} \in U$, multiplied by ± 1, depending on which side of the surface $\mathbf{p}$ lies. In reality U is not known, but we can mimic this procedure using the oriented tangent planes as follows. First, we find the tangent plane $Tp(\mathbf{x}_i)$ whose center $\mathbf{o}_i$ is closest to $\mathbf{p}$. This tangent plane is a local linear approximation to U, so we take the estimated signed distance $\tilde{d}_U(\mathbf{p})$ to U to be the signed distance between $\mathbf{p}$ and its projection $\mathbf{z}$ onto $Tp(\mathbf{x}_i)$ (Figure 2.7); that is,

$$\tilde{d}_U(\mathbf{p}) = \mathrm{dist}_i(\mathbf{p}) = (\mathbf{p} - \mathbf{o}_i) \cdot \hat{\mathbf{n}}_i \,.$$

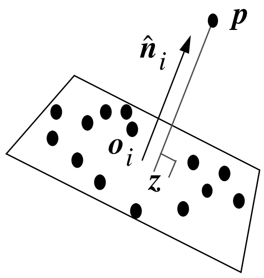


Figure 2.7: Signed distance from a point $\mathbf{p}$ to the nearest tangent plane.

If U is known to be a closed surface, this simple rule works well. However, the rule must be extended to accommodate surfaces that might have boundaries. As discussed in Section 2.1, because X is assumed to be a δ-noisy, ρ-dense sampling of U, the points do not leave unsampled holes of radius larger than $\rho + \delta$. Therefore if the projection $\mathbf{z}$ of $\mathbf{p}$ onto the closest tangent plane has $d(\mathbf{z}, X) > \rho + \delta$, $\mathbf{z}$ cannot be a point of U, and we take $\tilde{d}_U(\mathbf{p})$ to be undefined. Undefined values are used by the contouring algorithm (described in Section 2.2.5) to identify boundaries.

Stated procedurally, the signed distance function is defined as:

$i \leftarrow$ index of tangent plane whose center is closest to $\mathbf{p}$

{ *Compute* $\mathbf{z}$ *as the projection of* $\mathbf{p}$ *onto* $Tp(\mathbf{x}_i)$ }
$\mathbf{z} \leftarrow \mathbf{o}_i - ((\mathbf{p} - \mathbf{o}_i) \cdot \hat{\mathbf{n}}_i)\, \hat{\mathbf{n}}_i$

if $d(\mathbf{z}, X) < \rho + \delta$ **then**
    $\tilde{d}_U(\mathbf{p}) \leftarrow (\mathbf{p} - \mathbf{o}_i) \cdot \hat{\mathbf{n}}_i$ $\{= \pm\|\mathbf{p} - \mathbf{z}\|\}$
else
    $\tilde{d}_U(\mathbf{p}) \leftarrow$ **undefined**
endif

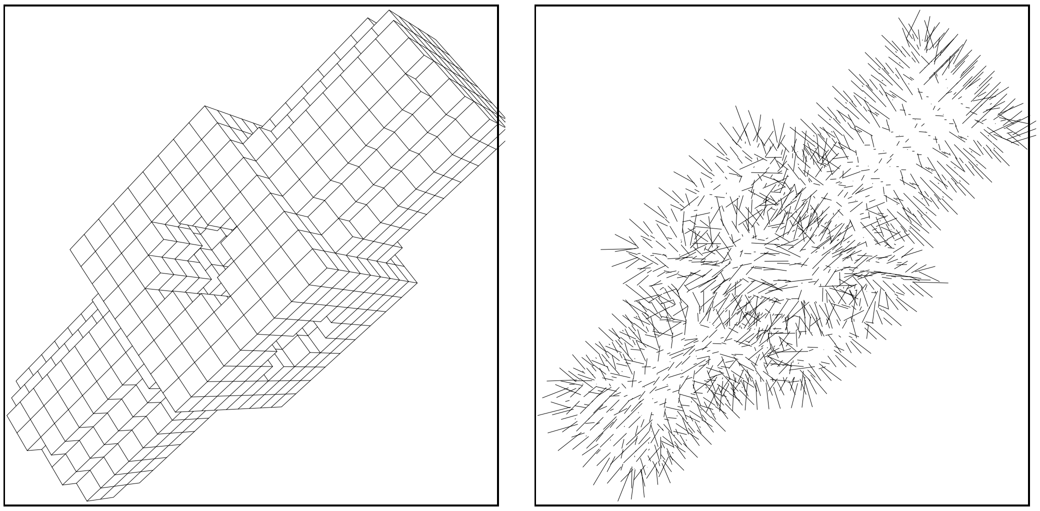
The simple approach outlined above creates a signed distance function $\tilde{d}_U$ whose zero set $Z(\tilde{d}_U)$ is piecewise linear but contains discontinuities. The discontinuities result from the implicit partitioning of space into regions within which a single tangent plane is used to define $\tilde{d}_U$. (These regions are in fact the Voronoi regions associated with the centers $\mathbf{o}_i$.) Fortunately, the discontinuities do not adversely affect the algorithm. The contouring algorithm discussed in the next section discretely samples the function $\tilde{d}_U$ over a portion of a 3-dimensional grid near the data and reconstructs a *continuous* piecewise linear approximation to $Z(\tilde{d}_U)$.

2.2.5 Contour tracing

Contour tracing, the extraction of an isosurface from a scalar function, is a well-studied problem [1, 13, 76]. Most contour tracing algorithms partition space into cubes or simplices (tetrahedra), evaluate the scalar function at the vertices of these volume elements, and for each element, from the values at its vertices, infer a linear approximation to the surface. Phase 1 currently uses the algorithm of Wyvill *et al.* [76], because it is simple to implement and, unlike the standard "marching cubes" algorithm [34], does not suffer from ambiguous configurations. Also, it produces sparser representations (for a given accuracy) than the method of Allgower and Schmidt [1] which uses tetrahedral decompositions of cubes. (A disadvantage of the method of Wyvill *et al.* is that it does not generalize to higher dimensions.)

In our implementation of the algorithm of Wyvill *et al.*, we handle degenerate zero evaluations ($\tilde{d}_U(\mathbf{p}) = 0$) by arbitrarily perturbing them to a small positive value, in order to guarantee that the output is a surface. The algorithm only visits cubes that intersect the zero set by pushing onto a queue only the appropriate neighboring cubes (Figure 2.8a). No intersection is reported within a cube if the signed distance function is undefined at any vertex of the cube, thereby giving rise to surface boundaries. As a result, the signed distance function $\tilde{d}_U$ is evaluated only near the data. In Figure 2.8b, evaluation of the signed distance function is shown graphically as line segments between the query points $\mathbf{p}$ (at the cube vertices) and their projections $\mathbf{z}$ onto the nearest tangent plane.

To accurately estimate the surface and to properly infer its topological type, the cube size should be set so that cube edges are of length $\leq \rho + \delta$. In practice it has been convenient to simply let the cube size equal $\rho + \delta$, to free the user from having to specify another parameter.



(a) Cubes visited by marching cubes

(b) Estimated signed distance (shown as $\mathbf{p} \leftrightarrow \mathbf{z}$)

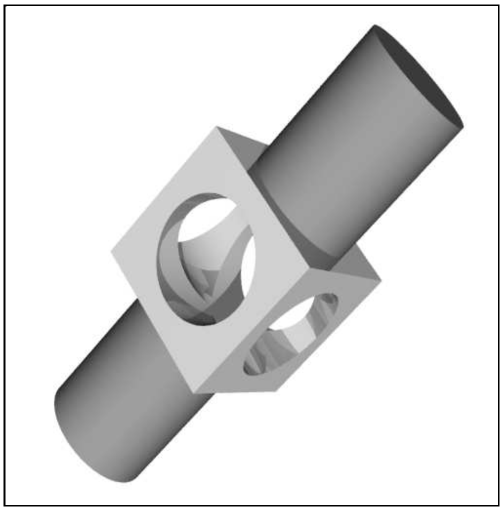
Figure 2.8: Contour tracing of $Z(\tilde{d}_U)$.

The result of phase 1 is a mesh approximating $Z(\tilde{d}_U)$. Drawn as a wireframe (Figure 2.1), the mesh is seen to be very dense (it has numerous vertices, edges, and faces). The mesh has the same topological type as the original surface U shown in Figure 2.9a, but as is evident in the shaded version of Figure 2.9b, the result of phase 1 is far from faithful to the geometry of the original model. This deficiency will be addressed in the next two chapters.

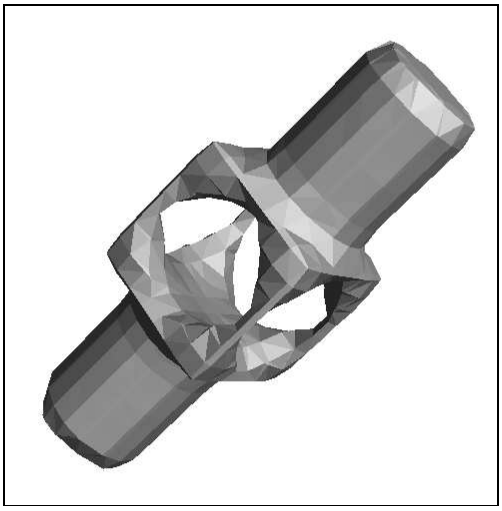
2.3 Results

We have experimented with data sets obtained from several different sources. In all cases, any structure (such as ordering) that might have been present in the point sets was discarded.

Simulated range data : To simulate laser range imaging from multiple view points, Constructive Solid Geometry (CSG) models were ray traced from multiple eye points. The ray tracer recorded the point of first intersection along each ray. Eight eye points (the vertices of a large cube enclosing the object) were used to generate the 4,102 points shown in Figure 2.1. This is the point set used in Section 2.2 to illustrate the steps of the algorithm.

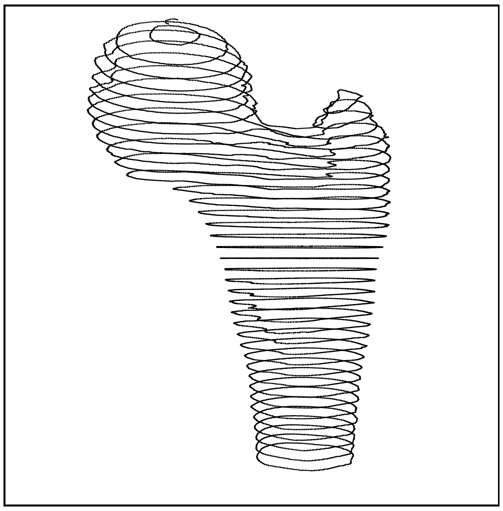


(c) Original surface U

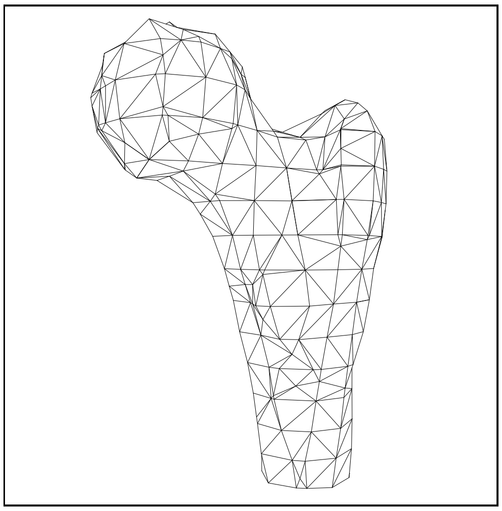


(d) Shaded phase 1 mesh

Figure 2.9: Comparison of the original surface U with the result of phase 1.



(a) Contour data from a CT scan



(b) Phase 1 reconstruction

Figure 2.10: Phase 1 of surface reconstruction on contour data.

Contours : Points from 39 planar slices of the CT scan of a femur (Figure 2.10a) were combined to reconstruct the surface of Figure 2.10b. Note how the branching problem (where a single contour forks into two contours) was handled correctly without any special code.

Figure 2.11a shows 30,937 points on 794 cross-sections of an oil pump (courtesy of Ford Motor Company). The surface reconstructed in phase 1 is shown in Figure 2.11b.

Real range data : Points were sampled from two physical objects by Technical Arts Co. (Redmond, WA) using a laser scanning head mounted on a coordinate measuring machine: 12,745 points from a Nissan distributor cap (Figure 2.11c) and 12,772 points from a mannequin head (Figure 2.11e). The phase 1 reconstructions are shown in Figures 2.11d and 2.11f. The holes present in the surface of Figure 2.11d are artifacts of the data, as self-shadowing prevented some regions of the surface from being scanned. Adaptive selection of scanning paths preventing such shadowing is an interesting area of future research. In this case, we manually closed the holes by introducing vertices at their centroids and adding new faces. (We did leave one boundary at the bottom of the distributor cap.)

Existing surfaces : By sampling points from a variety of existing surface models, we were able to compare results with known references.

The 10,000 points shown in Figure 2.11g were sampled from a swept surface (knot courtesy of Rob Scharein), and were used to reconstruct the surface in Figure 2.11h. This surface is an example of a surface with simple topological type (that of a torus) but a complex geometric embedding.

The 26,103 points in Figure 2.11i were sampled from the Utah teapot, defined as a set of NURBS patches. Since in the NURBS definition, the patches intersect each other (for instance, the spout penetrates inside the body), we had to filter the sample points to make them lie on a manifold. Phase 1 of surface reconstruction yields the single genus 1 surface (with one boundary at the spout) shown in Figure 2.11j.

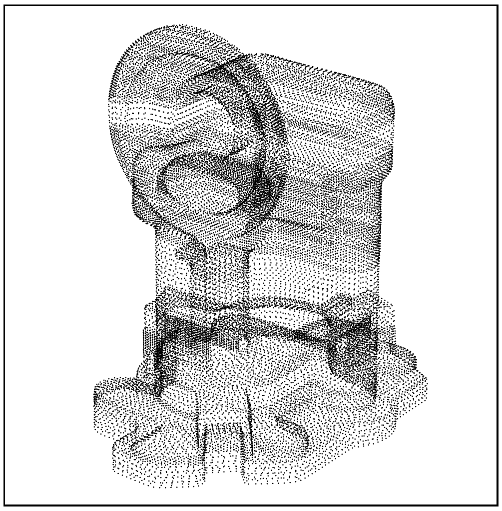
A piecewise linear surface, the mesh of Figure 2.5a, was randomly sampled to yield the sparse set of 1,000 points shown in Figure 2.11k. From these points, the surface in Figure 2.11l was reconstructed. This particular case illustrates the behavior of the method on a bordered surface (the cat has no base and is thus homeomorphic to a disc).

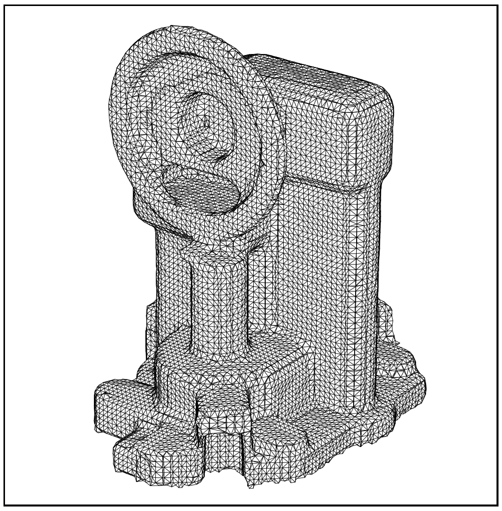
Table 2.1: Phase 1 sampling parameters and execution times.

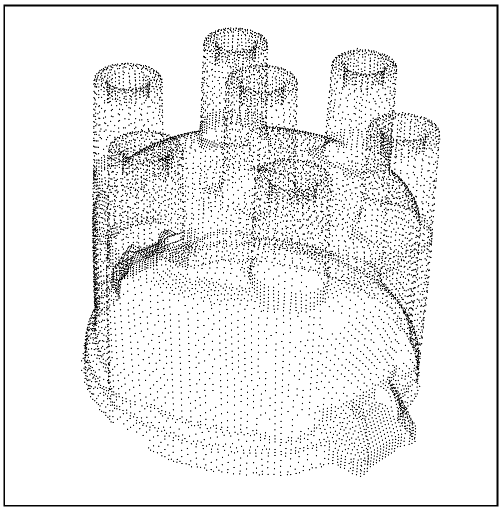
Figure	Object	# points n	Parameter $\rho + \delta$	Time (seconds)
2.1b	mechpart	4,102	.035	14
2.10b	femur	18,224	.06	190
2.11b	oilpmp	30,937	.011	104
2.11d	distcap	12,745	.02	40
2.11f	mannequin	12,772	.015	52
2.11h	knot	10,000	.025	75
2.11j	teapot	26,103	.02	133
2.11l	cat	1,000	.08	4
6.2a	curve1	200	.05	.6
6.2b	curve2	200	.08	.6

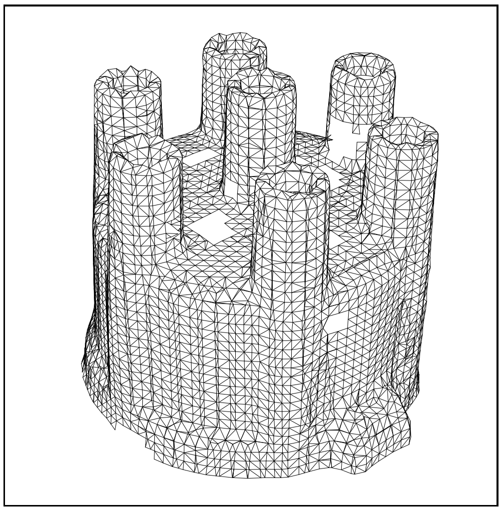
Table 2.1 lists the sampling parameter used in each example, expressed as a fraction of the object's size, and execution times obtained on an SGI Indigo workstation.

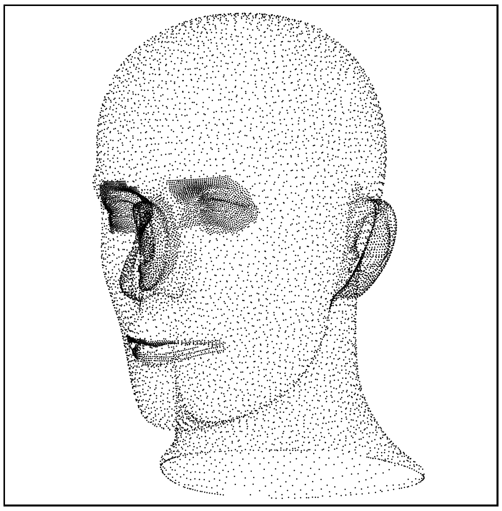
As shown in Figure 2.12, the method is not overly sensitive to the value of the parameter $\rho + \delta$. However, for parameter values that are too low, holes appear in the surface; and, for parameter values that are too high, the genus of the surface is not inferred correctly.


(a) Cross-section data


(b) Phase 1 reconstruction


(c) Laser range data


(d) Phase 1 reconstruction


(e) Laser range data

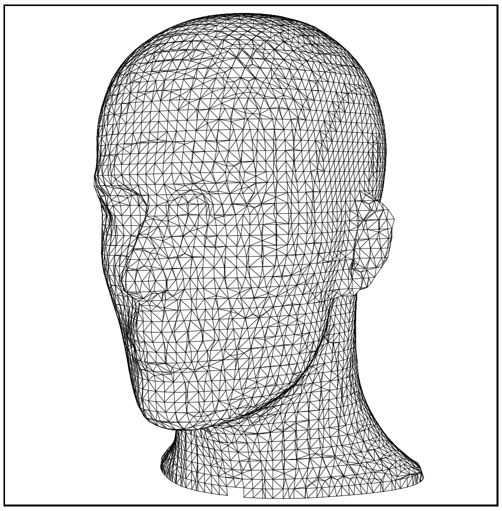
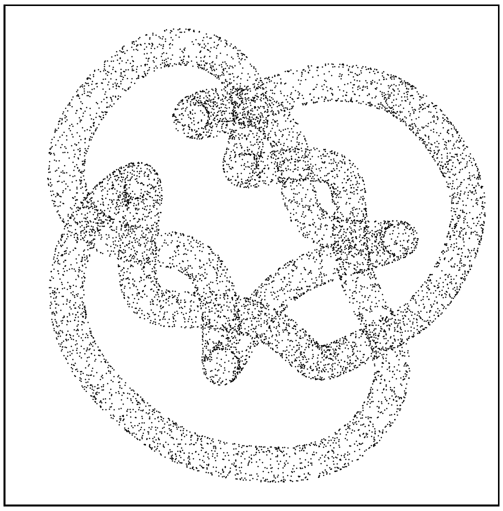
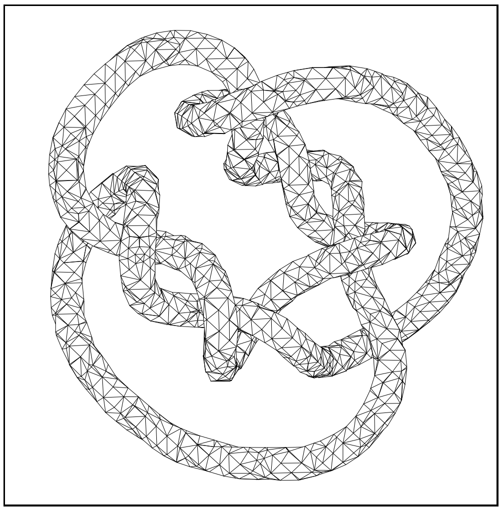

(f) Phase 1 reconstruction

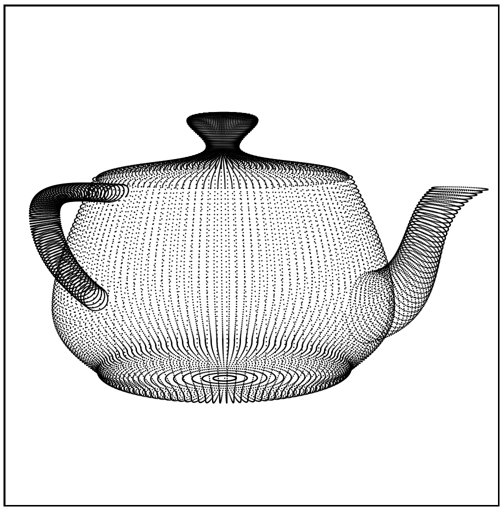
Figure 2.11: Results of phase 1 (initial surface estimation).



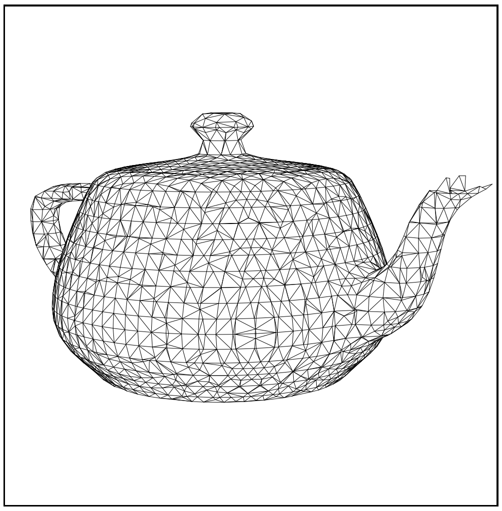
(g) Points from a swept surface



(h) Phase 1 reconstruction

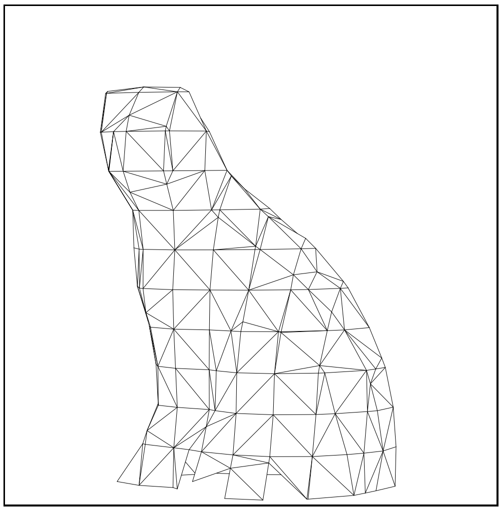


(i) Points from a set of NURBS patches



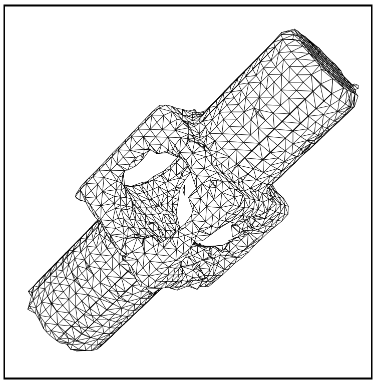
(j) Phase 1 reconstruction

(k) Points from a mesh

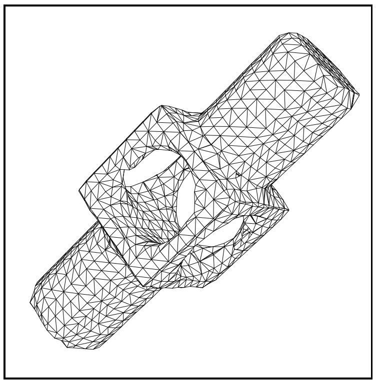


(l) Phase 1 reconstruction

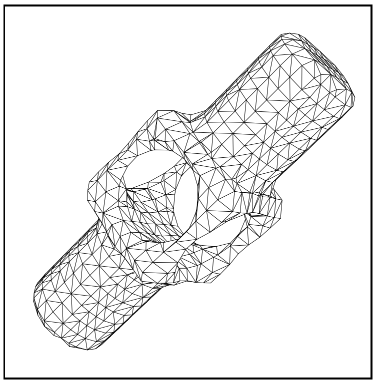
Figure 2.11: (continued)



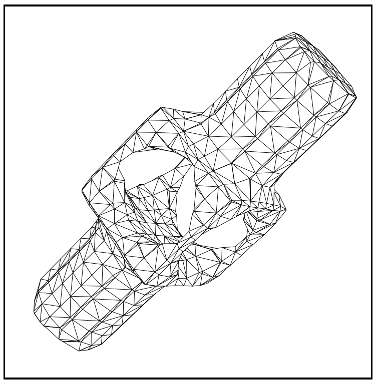
(a) $\rho + \delta = .020$



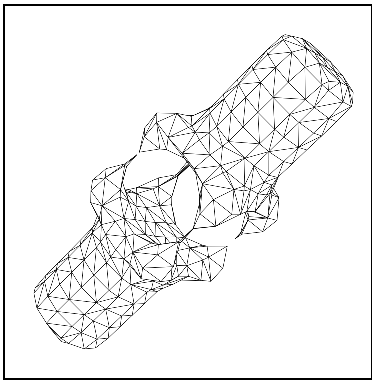
(b) $\rho + \delta = .025$



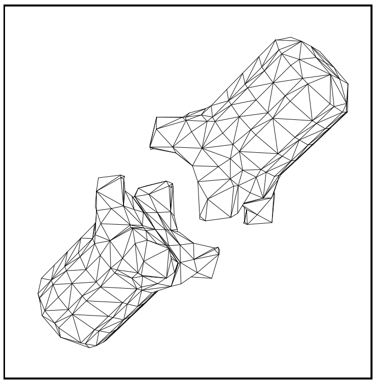
(c) $\rho + \delta = .030$



(d) $\rho + \delta = .035$



(e) $\rho + \delta = .040$



(f) $\rho + \delta = .045$

Figure 2.12: Sensitivity of the phase 1 reconstruction to the parameter $\rho + \delta$.

2.4 Discussion

Parameter setting Because the assumptions on the sampling and the surface are interlinked (Section 2.1), the single parameter $\rho + \delta$ is used for several purposes in phase 1:

- It determines the neighborhoods $Nbhd(\mathbf{x}_i)$ of points used in estimating tangent planes to the surface;
- It determines geometric proximity of the tangent planes $Tp(\mathbf{x}_i)$ (connectivity in the Riemannian Graph);
- It determines if, in the evaluation of the estimated signed distance $\tilde{d}_U(\mathbf{p})$, the projection of $\mathbf{p}$ onto the nearest tangent plane lies beyond a boundary of the surface;
- It is used to set the resolution of the contour tracing algorithm.

When scanning multiple objects with a scanner of a given technology, the parameter $\rho + \delta$ can likely be set to a common value. Specifying a global value for $\rho + \delta$ works well for data sets with uniform sampling density, uniform noise, and uniform surface features. But ideally this parameter should be adapted locally to variations in sampling, and should be inferred from the points themselves.

One possible scheme for automatically adapting the sampling density parameter is to examine the shape of the neighborhoods $Nbhd(\mathbf{x}_i)$ as points are added in order of increasing distance. The "shape" of a neighborhood can be described by the *principal frame* $(\mathbf{o}, \sqrt{\lambda^1}\mathbf{v}^1, \sqrt{\lambda^2}\mathbf{v}^2, \sqrt{\lambda^3}\mathbf{v}^3)$ obtained from principal component analysis (Section 2.2.2). For small sets of points, data noise tends to dominate, the eigenvalues $\lambda^{\cdot}$ are similar, and the eigenvectors $\mathbf{v}^{\cdot}$ may not reveal the surface's true tangent plane. At the other extreme, as number of points grows larger, the neighborhoods become less localized and the surface curvature tends to increase the "thickness" λ^3 of the neighborhood. Another possible criterion is to compare λ^3 to some local or global estimate of data noise. Although we have done some initial experimentation in this direction, we have not yet fully examined these options. One difficulty is that some structured data sets (such as the contour data in Figure 2.10a) have anisotropic sampling density.

Complexity analysis It is difficult to analyze the asymptotic complexity of the phase 1 algorithm as a function of the number n of points, because the value of the parameter $\rho + \delta$ greatly affects the complexity of several subproblems. However, empirical observation reveals the following:

- the size of the neighborhoods $Nbhd(\mathbf{x}_i)$ tends to remain constant as n increases, since the parameter $\rho + \delta$ is adjusted to a smaller value;
- therefore the Riemannian Graph has size $O(n)$;
- therefore the computation of the MST on the Riemannian Graph requires $O(n \log n)$ time.

The most expensive subproblems then become:

1. For each point $\mathbf{x}_i$, determining the neighborhood $Nbhd(\mathbf{x}_i)$, i.e. the $O(1)$ set of points within distance $\rho + \delta$. As there are n such neighborhoods, this problem would require $O(n^2)$ time overall if done by brute force.
2. For each point $\mathbf{p} \in \mathbf{R}^3$ at which $\tilde{d}_U(\mathbf{p})$ is evaluated, finding the nearest tangent plane origin $\mathbf{o}_i$. The number of such evaluations is proportional to n. To see this, note that the contour tracing cube size is set equal to $\rho + \delta$, and is thus proportional to the average spacing between the points. Therefore the number of contour tracing cubes occupied by the n data points is proportional to n. Thus this problem would also require $O(n^2)$ if implemented by brute force.

Hierarchical spatial partitioning schemes such as octrees [55] and k-D trees [3] can greatly speed up these spatial searching problems. As we assume uniform sampling density in our data, a simpler scheme, based on uniform cubic partitioning, has worked effectively. The axis-aligned bounding box of the points is partitioned by a cubical grid. Points are entered into sets corresponding to the cube to which they belong, and these sets are accessed through a hash table indexed by the cube indices (a similar scheme is described in Wyvill *et al.* [76]). It is difficult to analyze the resulting improvements analytically, but empirically, the time complexity of the above problems is effectively reduced from $O(n^2)$ to $O(n)$.

Specialization to range data If the data points are obtained from range images, there exists some knowledge of surface orientation at each data point. Indeed, each data point is known to be visible from at least one viewing direction[3], so that, unless the surface grazing angle is large, a point's tangent plane orientation can usually be inferred from the viewing direction. The current algorithm can exploit this additional information in the tangent plane orientation step (Section 2.2.3). Let the Riemannian Graph be augmented with one additional vertex γ and n additional edges from γ to each tangent plane vertex i. Assign to each new edge (γ, i) the cost $1 - |\hat{\mathbf{n}}_i \cdot \hat{\mathbf{v}}_i|$ where $\hat{\mathbf{n}}_i$ is the normal of $Tp(\mathbf{x}_i)$ and $\hat{\mathbf{v}}_i$ is the viewing direction at point $\mathbf{x}_i$. Then compute the MST of this new graph and propagate orientation as before. This strategy was implemented but never used, as we were always able to correctly infer orientation without resorting to it.

Culling of outliers Another possible use for the principal component analysis of the neighborhoods $Nbhd(\mathbf{x}_i)$ is the culling of outlier data points. If the principal frame is not sufficiently "flat" (i.e. if λ^3 is large) or if its origin $\mathbf{o}_i$ is distant from $\mathbf{x}_i$ (as measured in principal frame coordinates), then $Nbhd(\mathbf{x}_i)$ does not appear to lie on a 2-manifold and the point $\mathbf{x}_i$ should be removed. It may be safer to remove data points than to proceed with uncertain tangent plane estimates. But fortunately we did not have to filter our data, as it was free of such outliers.

[3] In a ranger scanner based on triangulation, it is known that the surface point is visible from both the light source and the camera.

Chapter 3

PHASE 2: MESH OPTIMIZATION

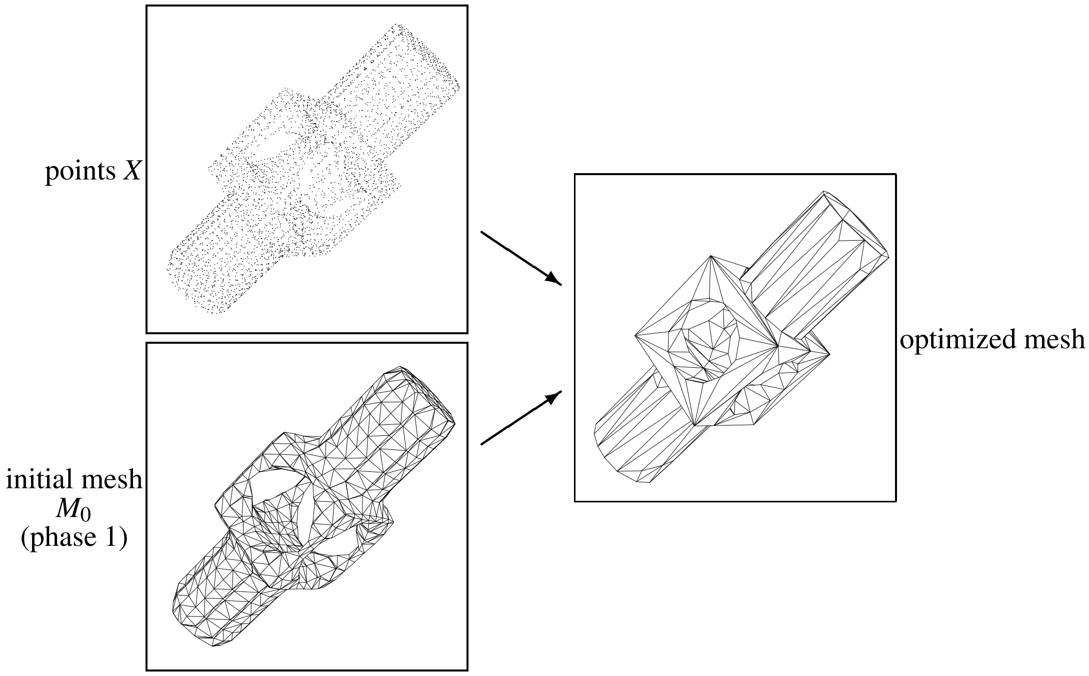


Figure 3.1: Phase 2: optimization of the phase 1 mesh to fit the points X.

3.1 Introduction

Phase 1, described in the previous chapter, creates an initial surface approximation from a set of points X, in the form of a dense mesh. Phase 2, the topic of this chapter, seeks to improve the accuracy and conciseness of this mesh. This is achieved by solving a *mesh optimization* problem, stated roughly as follows: Given a collection of data points X in $\mathbf{R}^3$ and an initial mesh M_0 near the data, find a mesh M of the same topological type as M_0 that both fits the data well and has a small number of vertices. A "perfect" fit to the data would involve interpolating the points. However, since X may be a noisy sampling, we do not seek an interpolating surface, as such a surface would contain many unwanted folds and undulations.

In the example of Figure 3.1, from the set of 4,102 points and the initial mesh created in phase 1, mesh optimization creates a new, more concise, more accurate mesh. Notice that the sharp edges and corners indicated by the data points have been faithfully recovered and that the number of vertices has been significantly reduced (from 886 to 163).

To solve the mesh optimization problem we minimize an *energy function* that captures the competing desires of tight geometric fit and compact representation. Using the input mesh M_0 as a starting point, an optimization algorithm minimizes this non-linear energy function by varying the number of mesh vertices, their positions, and their connectivity (under the constraint that the topological type be maintained). In essence, the search space of the optimization consists of *all* meshes of the same topological type as M_0. Although we can give no guarantee of finding a global minimum, we have run the method on a wide variety of data sets, and we have obtained good results in all cases, as demonstrated in Section 3.4.

Most previous methods for surface fitting only consider surfaces of simple topological type (e.g. rectangular or spherical domains), as described in Section 1.3.1.

In contrast, the mesh optimization algorithm allows fitting of a parametric surface of arbitrary topological type to a set of points. In addition to fitting the mesh, the optimization also varies the number of vertices and their connectivity, thereby locally tailoring the degrees of freedom in the representation to the geometry of the data points.

The principal contributions of phase 2 are:

- It presents an algorithm for fitting a mesh of arbitrary topological type to a set of data points (as opposed to volume data, etc.).

- It casts mesh fitting as the minimization of an energy function that embodies the competing goals of accuracy and conciseness.

- It defines a set of mesh transformations, and conditions under which they can be applied, that allow consideration of all meshes of a given topological type.

- It shows that the resulting non-linear optimization problem can be made tractable through the use of a nested optimization algorithm and a set of approximations.

- It demonstrates how the algorithm's ability to recover sharp edges and corners can be exploited to automatically segment the final mesh into smooth connected components.

Another application of mesh optimization, that of finding concise piecewise linear approximations to existing surfaces, is discussed in Chapter 5.

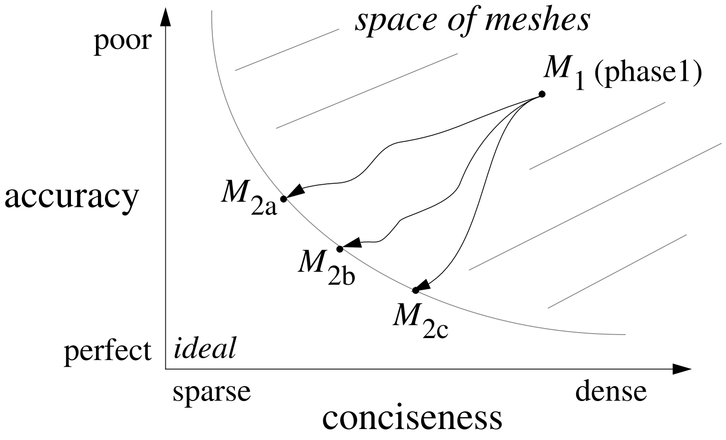
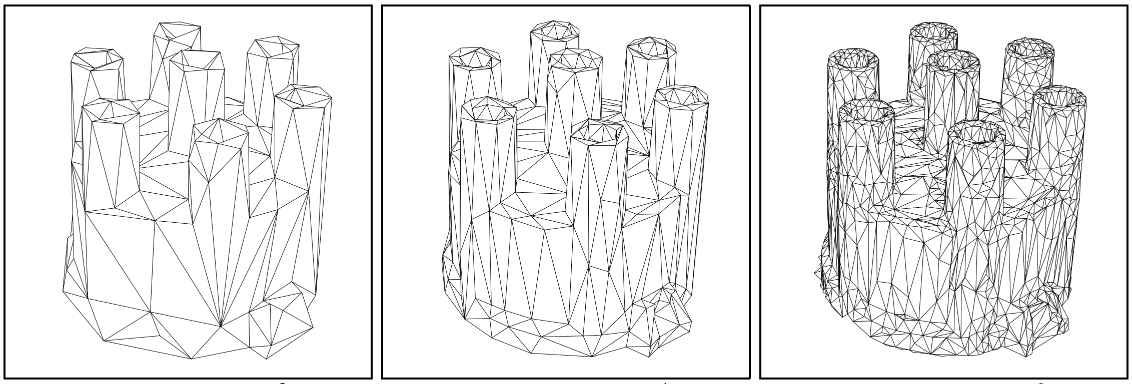


Figure 3.2: Trade-off between accuracy and conciseness in phase 2.

Trade-off between accuracy and conciseness The two goals of mesh optimization, those of accuracy and conciseness, are competing goals, as illustrated in the graph of Figure 3.2. An ideal surface representation would lie in the lower left corner of this graph, as it would both be concise and have perfect fit to the data. However, since we are only considering piecewise linear surfaces (meshes), there is a bound on the accuracy that a representation of a given size can achieve. The meshes $M_{2\mathrm{a}}$, $M_{2\mathrm{b}}$, and $M_{2\mathrm{c}}$, which lie on the boundary of this "space of meshes", are all in a sense optimal; they exhibit a trade-off between accuracy and conciseness. In mesh optimization, this trade-off between geometric fit and compact representation is controlled via a user-selectable parameter c_{rep}. A large value of c_{rep} indicates that a coarse representation is to be strongly preferred over a dense one, usually at the expense of degrading the fit.

As an example, Figure 3.3 shows three optimized meshes obtained with different values of c_{rep}. At one extreme, the first mesh (obtained with a high value of c_{rep}) is concise but a poor geometric fit; at the other extreme, the last mesh (obtained from a low value of c_{rep}) is dense but much more accurate.



(a) $M_{2\mathrm{a}}$ with $c_{rep} = 10^{-3}$ (b) $M_{2\mathrm{b}}$ with $c_{rep} = 10^{-4}$ (c) $M_{2\mathrm{c}}$ with $c_{rep} = 10^{-6}$

Figure 3.3: Three optimized meshes obtained with different values of c_{rep}.

3.2 Definition of the energy function

In the following discussion, it may be useful to refer back to Section 1.7.1 for notation and terminology.

Recall that the goal of mesh optimization is to obtain a mesh that both provides a good fit to the point set $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ and has a small number of vertices. We find a simplicial complex K and a set of vertex positions $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$ defining a mesh $M = (K, V)$ that minimizes an energy function carefully chosen to meet our two stated goals:

$$E(K, V) = E_{dist}(K, V) + E_{rep}(K) + E_{spring}(K, V)\,.$$

The first two terms measure the accuracy and conciseness of the mesh; the third term is motivated below.

The distance energy E_{dist} is equal to the sum of squared distances from the points to the mesh,

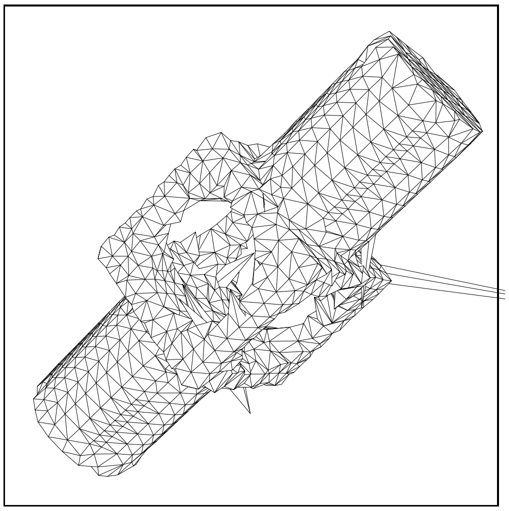
$$E_{dist}(K, V) = \sum_{i=1}^{n} d^2(\mathbf{x}_i, \pi_V(|K|))\,.$$

The representation energy E_{rep} penalizes meshes with a large number of vertices. It is set to be proportional to the number m of vertices of K:

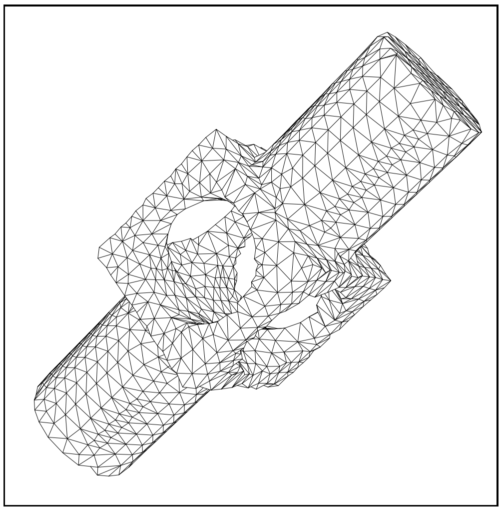
$$E_{rep}(K) = c_{rep} m\,.$$

The optimization allows vertices to be both added to and removed from the mesh. When a vertex is added, the distance energy E_{dist} is likely to be reduced; the term E_{rep} charges a penalty to this operation so that vertices are not added indefinitely. Similarly, one wants to remove vertices from a dense mesh even if E_{dist} increases slightly; in this case E_{rep} acts to encourage the vertex removal. The user-specified parameter c_{rep} provides a controllable trade-off between fidelity of geometric fit and parsimony of representation.

We discovered, as others have before us [37], that minimizing $E_{dist} + E_{rep}$ alone does not produce the desired results. As an illustration of what can go wrong, Figure 3.4a shows the result of minimizing E_{dist} alone. The estimated surface has several spikes in regions where there is no data. These spikes are a manifestation of the fundamental problem that a minimum of $E_{dist} + E_{rep}$ may not exist. If, in the course of minimization, few points project onto the neighborhood of a vertex, then the optimal position of this vertex may become ill-defined, and the vertex may therefore wander arbitrarily far from the data.



(a) Minimizing E without E_{spring}



(b) Minimizing E with E_{spring}

Figure 3.4: Minimization of E for fixed K without and with spring energy.

To guarantee the existence of a minimum [32], we add the third term, the spring energy E_{spring}. It places on each edge of the mesh a spring of rest length zero and spring constant κ:

$$E_{spring}(K, V) = \sum_{\{j,k\} \in K} \kappa \|\mathbf{v}_j - \mathbf{v}_k\|^2 .$$

Minimizing E with the spring energy term produces a much more stable result, as shown in Figure 3.4b.

It is worthwhile emphasizing that the spring energy is not a smoothness penalty. Our intent is not to penalize sharp dihedral angles in the mesh, since such features may be present in the underlying surface and should be recovered. We view E_{spring} as a regularizing term that helps guide the optimization into a desirable local energy well. As the optimization converges to the solution, the magnitude of E_{spring} can be gradually reduced. We return to this issue in Section 3.3.5.

For some applications, the procedure should be scale-invariant, equivalent to defining a unitless energy function E. To achieve invariance under Euclidean motion and uniform scaling, the points X and the initial mesh M_0 are uniformly pre-scaled to fit in a unit cube. After optimization, a post-processing step can undo this initial transformation.

The energy function $E(K, V)$ depends on two parameters c_{rep} and κ. The parameter c_{rep} controls the trade-off between conciseness and fidelity to the data and should be set by the user. The parameter κ, on the other hand, is a regularizing parameter that is chosen automatically. The method for setting κ is described in Section 3.3.5.

3.3 Minimization of the energy function

Our goal is to minimize the energy function

$$E(K, V) = E_{dist}(K, V) + E_{rep}(K) + E_{spring}(K, V)$$

over the set $\mathcal{K}$ of simplicial complexes K homeomorphic to the initial simplicial complex K_0, and the vertex positions V defining the embedding. Here is an outline of the optimization algorithm, a pseudo-code version of which appears in Figure 3.5. The details are deferred to the next two subsections.

To minimize $E(K, V)$ over both K and V, the problem is partitioned into two nested subproblems: an inner, continuous minimization over V for fixed simplicial complex K, and an outer, discrete minimization over K.

Section 3.3.1 describes an algorithm that solves the inner minimization problem. Its goal is to find $E(K) = \min_V E(K, V)$, the energy of the best possible embedding of the fixed simplicial complex K, and the corresponding vertex positions V, given an initial guess for V. This corresponds to the procedure OptimizeVertexPositions in Figure 3.5.

Whereas the inner minimization is a continuous optimization problem, the outer minimization of $E(K)$ over the simplicial complexes $K \in \mathcal{K}$ (procedure OptimizeMesh) is a discrete optimization problem. An algorithm for its solution is presented in Section 3.3.2.

```
OptimizeMesh(K_0,V_0) {
    K := K_0
    V := OptimizeVertexPositions(K_0,V_0)
    – Solve the outer minimization problem.
    repeat {
        (K',V') := GenerateLegalMove(K,V)
        V' = OptimizeVertexPositions(K',V')
        if E(K', V') < E(K, V) then
            (K,V) := (K',V')
        endif
    } until convergence
    return (K,V)
}
– Solve the inner optimization problem
–    E(K) = min_V E(K, V)
– for fixed simplicial complex K.
OptimizeVertexPositions(K,V) {
    repeat {
        – Compute barycentric coordinates by projection.
        B := ProjectPoints(K,V)
        – Minimize E(K, V, B) over V using conjugate gradients.
        V := ImproveVertexPositions(K,B)
    } until convergence
    return V
}
GenerateLegalMove(K,V) {
    Select a legal move K ⇒ K'.
    Locally modify V to obtain V' appropriate for K'.
    return (K',V')
}
```

Figure 3.5: An idealized pseudo-code version of the mesh optimization algorithm.

3.3.1 Optimization over V for fixed K (Procedure OptimizeVertexPositions)

In this section, we consider the problem of finding a set of vertex positions V that minimizes the energy function $E(K, V)$ for a given simplicial complex K. As $E_{rep}(K)$ does not depend on V, this amounts to minimizing $E_{dist}(K, V) + E_{spring}(K, V)$.

To evaluate the distance energy $E_{dist}(K, V)$, it is necessary to compute the distance of each data point $\mathbf{x}_i$ to the surface $\pi_V(|K|)$. Each of these distances is itself the solution to the minimization problem

$$d^2(\mathbf{x}_i, \pi_V(|K|)) = \min_{\mathbf{b}_i \in |K|} \|\mathbf{x}_i - \pi_V(\mathbf{b}_i)\|^2 ,$$

in which the unknown is the barycentric coordinate vector $\mathbf{b}_i \in |K| \subset \mathbf{R}^m$ of the projection of $\mathbf{x}_i$ onto M (Figure 3.6). Thus, minimizing $E(K, V)$ for fixed K is equivalent to minimizing the new objective function

$$\begin{aligned} E(K, V, B) &= \sum_{i=1}^{n} \|\mathbf{x}_i - \pi_V(\mathbf{b}_i)\|^2 + E_{spring}(K, V) \\ &= \sum_{i=1}^{n} \|\mathbf{x}_i - \pi_V(\mathbf{b}_i)\|^2 + \sum_{\{j,k\} \in K} \kappa \|\mathbf{v}_j - \mathbf{v}_k\|^2 \end{aligned}$$

over the vertex positions $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}, \mathbf{v}_i \in \mathbf{R}^3$ and the barycentric coordinates $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}, \mathbf{b}_i \in |K| \subset \mathbf{R}^m$.

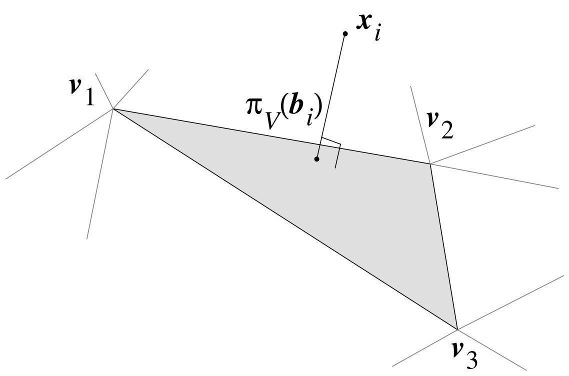


Figure 3.6: Distance of a point $\mathbf{x}_i$ from the mesh.

To solve this optimization problem (procedure OptimizeVertexPositions), the method alternates between two subproblems:

1. For fixed vertex positions V, find optimal barycentric coordinate vectors B by *projection* (procedure ProjectPoints).

2. For fixed barycentric coordinate vectors B, find optimal vertex positions V by solving a *linear* least squares problem (procedure ImproveVertexPositions).

Because we find optimal solutions to both of these subproblems, $E(K, V, B)$ can never increase, and since it is bounded from below, it must converge.[1] In principle, one could iterate until some formal convergence criterion is met. Instead, as is common, we perform a fixed number of iterations. As an example, Figure 3.4b shows the result of optimizing over the vertex positions while holding the simplicial complex fixed.

It is conceivable that procedure OptimizeVertexPositions returns a set V of vertices for which the mesh is self-intersecting, i.e. π_V is not an embedding. While it is possible to check *a posteriori* whether π_V is an embedding, constraining the optimization to always produce an embedding appears to be difficult. This has not presented a problem in the examples we have run.

3.3.1.1 Projection subproblem (Procedure ProjectPoints)

The problem of optimizing $E(K, V, B)$ over the barycentric coordinate vectors $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ while holding the vertex positions $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$ and the simplicial complex K constant, decomposes into n separate optimization problems:

$$\mathbf{b}_i = \underset{\mathbf{b} \in |K|}{\operatorname{argmin}} \left\| \mathbf{x}_i - \pi_V(\mathbf{b}) \right\| .$$

In other words, $\mathbf{b}_i$ is the barycentric coordinate vector corresponding to the point $\mathbf{p} \in \pi_V(|K|)$ closest to $\mathbf{x}_i$.

A naive approach to computing $\mathbf{b}_i$ is to project $\mathbf{x}_i$ onto all faces of M, and then find the projection with minimal distance. To speed up the projection, the faces of the mesh are first entered into a spatial partitioning data structure (similar to the one used in Wyvill *et al.* [76]).

[1] Although the energy E must converge, that is not necessarily true of the vertex positions V. Theoretically, it is possible that the iterative procedure follow a cyclic path in the space of vertex positions, converging over E while not converging over V. However, we have not encountered such behavior in practice.

Then for each point $\mathbf{x}_i$ only a nearby subset of the faces needs to be considered, so that the overall projection step takes expected time $O(n)$. For additional speedup we exploit coherence between iterations. A point's projection is assumed to lie in a neighborhood of its projection in the previous iteration. Specifically, the point is projected onto the simplicial neighborhood nbhd(f; K) of the face f onto which it previously projected. Although this is a heuristic that can fail, it has performed well in practice.

3.3.1.2 Linear least squares subproblem (Procedure ImproveVertexPositions)

Minimizing $E(K, V, B)$ over the vertex positions V while holding B and K fixed is a linear least squares problem. It decomposes into three independent subproblems, one for each of the three coordinates of the vertex positions. We will write down the problem for the first coordinate.

Let e be the number of edges (1-simplices) in K; note that e is $O(m)$. We can express the least squares problem for the first coordinate as minimizing a linear system $\|A\mathbf{v}^1 - \mathbf{d}^1\|^2$ over $\mathbf{v}^1$, where the design matrix A is an $(n + e) \times m$ matrix and $\mathbf{d}^1$ is an $(n + e)$-vector. The first n rows of the least squares problem correspond to $E_{dist}(K, V)$, so that row i of A is $\mathbf{b}_i$ and row i of $\mathbf{d}^1$ is $\mathbf{x}_{i,1}$. The last e rows correspond to the springs of $E_{spring}(K, V)$, so that each of these rows of A contains two non-zero entries with values $\sqrt{\kappa}$ and $-\sqrt{\kappa}$ in the columns corresponding to the indices of the edge's vertices, and these rows of $\mathbf{d}^1$ contain zero. An important feature of the matrix A is that it contains at most 3 non-zero entries in each row, for a total of $O(n + m)$ non-zero entries.

The least squares problem is solved using the conjugate gradient method (cf. Golub and Van Loan [24]). This is an iterative method guaranteed to find the exact solution in as many iterations as there are distinct singular values of A, i.e. in at most m iterations. Usually far fewer iterations are required to get a result with acceptable precision. For example, we find that for m as large as 10^4, as few as 200 iterations are sufficient.

The two time-consuming operations in each iteration of the conjugate gradient algorithm are the multiplication of A by an $(n + e)$-vector and the multiplication of A^T by an m-vector. Because A is sparse, these two operations can be executed in $O(n + m)$ time. Thus, an acceptable solution to the least squares problem is obtained in $O(n + m)$ time (with a constant number of conjugate gradient iterations). In contrast, a typical noniterative method for solving dense least squares problems, such as QR decomposition, would require $O((n + m)m^2)$ time to find an exact solution.

3.3.2 Optimization over K (Procedure OptimizeMesh)

To solve the outer minimization problem, minimizing $E(K)$ over K, we define a set of three elementary mesh transformations, *edge collapse*, *edge split*, and *edge swap*, that change a simplicial complex K into another simplicial complex K' (see Figure 3.7).

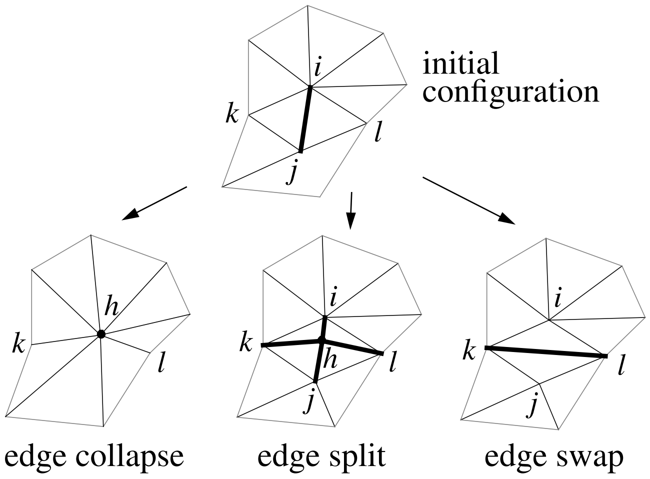


Figure 3.7: The three elementary mesh transformations defined in phase 2.

We define a *legal move* to be the application of one of these elementary transformations to an edge of K that leaves the topological type of K unchanged. The set of elementary transformations is complete in the sense that *any* simplicial complex in $\mathcal{K}$ can be obtained from K_0 through a sequence of legal moves.[2]

Our goal then is to find such a sequence taking us from K_0 to a minimum of $E(K)$. We do this using a variant of random descent: we randomly select a legal move, $K \Rightarrow K'$. If $E(K') < E(K)$, we accept the move, otherwise we select another move and repeat. We use a slightly more sophisticated strategy for randomly selecting legal moves, described in Section 3.3.3. More elaborate optimization schemes, such as steepest descent or simulated annealing, are possible. We have obtained good results with the simple method of random descent, so we have not yet implemented the other schemes.

[2] In fact, Duchamp [32] shows that edge collapse and edge split are sufficient; we include edge swap to allow the optimization procedure to "tunnel" through small hills in the energy function.

Identifying legal moves An edge split transformation is always a legal move, as it can never change the topological type of K. The other two transformations, on the other hand, can cause a change of topological type, so tests must be performed to determine if they are legal moves.

We define an edge $\{i, j\} \in K$ to be a *boundary edge* if it is a subset of only one face $\{i, j, k\} \in K$, and a vertex $\{i\}$ to be a *boundary vertex* if there exists a boundary edge $\{i, j\} \in K$.

An edge collapse transformation $K \Rightarrow K'$ that collapses the edge $\{i, j\} \in K$ is a legal move if and only if the following conditions are satisfied (proven in joint work with Duchamp [32]):

- For all vertices $\{k\}$ adjacent to both $\{i\}$ and $\{j\}$ ($\{i, k\} \in K$ and $\{j, k\} \in K$), $\{i, j, k\}$ is a face of K.
- If $\{i\}$ and $\{j\}$ are both boundary vertices, $\{i, j\}$ is a boundary edge.
- K has more than 4 vertices if neither $\{i\}$ nor $\{j\}$ are boundary vertices, or K has more than 3 vertices if either $\{i\}$ or $\{j\}$ are boundary vertices.

An edge swap transformation $K \Rightarrow K'$ that replaces the edge $\{i, j\} \in K$ with $\{k, l\} \in K'$ is a legal move if and only if $\{k, l\} \notin K$.

3.3.3 Strategy for selecting legal moves (Procedure GenerateLegalMove)

The simple strategy of randomly selecting legal moves described in Section 3.3.2 can be improved by exploiting locality. Instead of selecting edges completely at random, edges are selected at random from a candidate set. This candidate set consists of all edges that may lead to beneficial moves, and initially contains all edges.

To generate a legal move, we randomly remove an edge from the candidate set. We first consider collapsing the edge, accepting the move if it is legal and reduces the total energy. If the edge collapse is not accepted, we then consider edge swap and edge split in that order. If one of the transformations is accepted, we update the candidate set by adding all neighboring edges. The candidate set becomes very useful toward the end of optimization, when the fraction of beneficial moves diminishes. The candidate set also provides a simple termination criterion: terminate when the candidate set becomes empty.

3.3.4 Exploiting locality

The idealized algorithm described so far is too inefficient to be of practical use. In this section, we describe some heuristics that dramatically reduce the running time. These heuristics capitalize on the observation that a local change in the structure of the mesh leaves the optimal positions of distant vertices essentially unchanged.

Our discrete optimization procedure requires evaluation of $E(K') = \min_{V'} E(K', V')$ for a simplicial complex K' obtained from K through a legal move. Ideally, we would use procedure OptimizeVertexPositions of Section 3.3.1 for this purpose, as indicated in Figure 3.5. In practice, however, this is too slow. Instead, we use fast local heuristics to estimate the effect of a legal move on the energy function.

Each of the heuristics is based on extracting a submesh in the neighborhood of the transformation, along with the subset of the data points projecting onto the submesh. The change in overall energy is estimated by considering only the contribution of the submesh and the corresponding point set. This estimate is always pessimistic, as full optimization would only further reduce the energy. Therefore, the heuristics never suggest changes that would increase the true energy of the mesh.

Evaluation of edge collapse To evaluate a transformation $K \Rightarrow K'$ collapsing an edge $\{i, j\}$ into a single vertex $\{h\}$ (Figure 3.7), we take the submesh to be the simplicial neighborhood $\mathrm{nbhd}(\{h\}; K')$ (as defined in Section 1.7.2), and optimize over the single vertex position $\mathbf{v}_h$ while holding all other vertex positions constant.

Because we perform only a small number of iterations (for reasons of efficiency), the initial choice of $\mathbf{v}_h$ greatly influences the accuracy of the result. Therefore, we perform three different optimizations, with $\mathbf{v}_h$ starting at $\mathbf{v}_i$, $\mathbf{v}_j$, and $\frac{1}{2}(\mathbf{v}_i + \mathbf{v}_j)$, and use the one that obtains the lowest energy.

The edge collapse should be allowed only if the new mesh does not intersect itself. Checking for this would be costly; instead we settle for a less expensive heuristic check. If, after the local optimization, the maximum dihedral angle of the edges in $\mathrm{nbhd}(\{h\}; K')$ is greater than some threshold, the edge collapse is rejected.

Evaluation of edge split The procedure is similar to that for edge collapse, with the submesh defined to be the neighborhood $\mathrm{nbhd}(\{h\}; K')$ about the new vertex $\{h\}$, and the initial position of $\mathbf{v}_h$ chosen to be $\frac{1}{2}(\mathbf{v}_i + \mathbf{v}_j)$.

Evaluation of edge swap To evaluate an edge swap transformation $K \Rightarrow K'$ that replaces an edge $\{i, j\} \in K$ with $\{k, l\} \in K'$, we consider two local optimizations, one with submesh nbhd($\{k\}; K'$), varying vertex $\mathbf{v}_k$, and one with submesh nbhd($\{l\}; K'$), varying vertex $\mathbf{v}_l$ (Figure 3.8). The change in energy is taken to best of these.[3] As is the case in evaluating an edge collapse, we reject the transformation if the maximum dihedral angle after the local optimization exceeds a threshold.

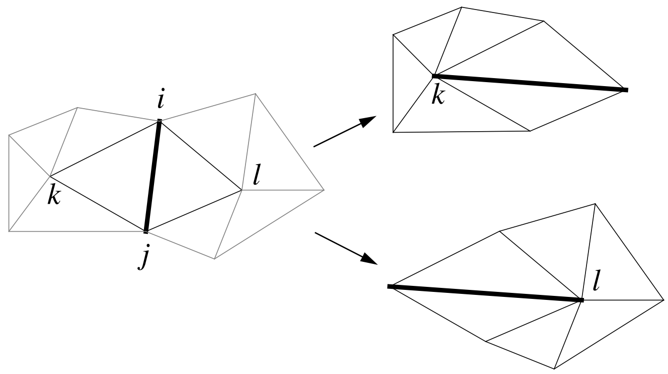


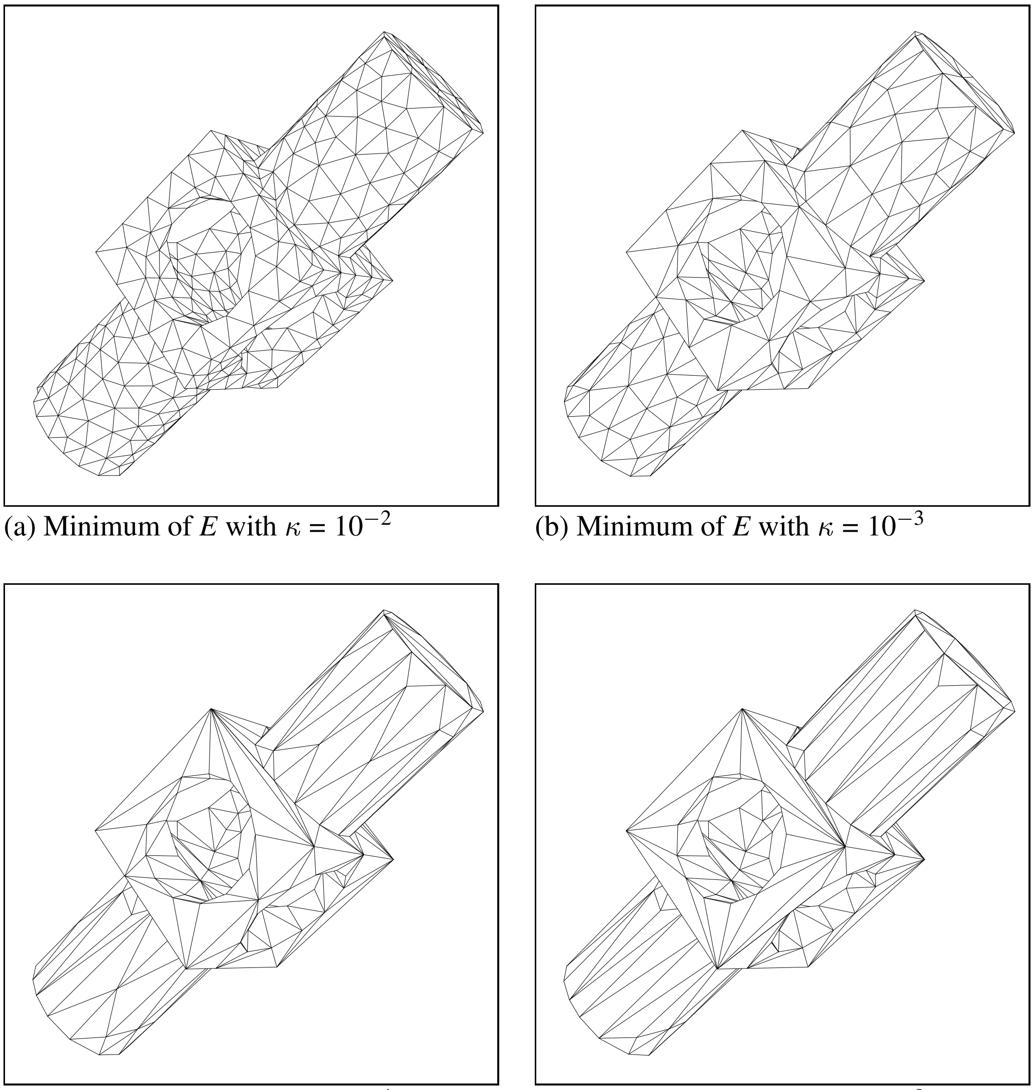
Figure 3.8: Two local optimizations to evaluate an edge swap mesh transformation.

3.3.5 Setting of the spring constant

We view the spring energy E_{spring} as a regularizing term that helps guide the optimization process to a good local minimum. The spring constant κ determines the contribution of this term to the total energy. We have obtained good results by making successive calls to procedure OptimizeMesh, each with a different value of κ, according to a schedule that gradually decreases κ.

As an example, to obtain the final mesh in Figure 3.9d starting from the phase 1 mesh in Figure 3.1, we successively set κ to 10^{-2}, 10^{-3}, 10^{-4}, and 10^{-8} (see Figures 3.9a–3.9d). This same schedule was used in all the examples.

[3] An obvious alternative is to optimize simultaneously over both $\mathbf{v}_k$ and $\mathbf{v}_l$, but this would slightly complicate the implementation.



(a) Minimum of E with $\kappa = 10^{-2}$

(b) Minimum of E with $\kappa = 10^{-3}$

(c) Minimum of E with $\kappa = 10^{-4}$

(d) Minimum of E with $\kappa = 10^{-8}$

Figure 3.9: Successive minimizations of E with decreasing spring constant κ schedule.

3.4 Results

From the point sets shown in Figure 2.11 and the initial meshes produced by phase 1 shown in the left column of Figure 3.10, phase 2 produces the optimized meshes shown in the right column of Figure 3.10.

As these examples reveal, by simply minimizing the energy function E, the density and shapes of elements in the meshes adapts to the curvature of the underlying surface:

- Vertices are dense in regions of high Gaussian curvature, whereas a few large faces span the flat regions.

- Long edges are aligned in directions of low curvature, and the aspect ratios of the triangles adjust to local curvature.

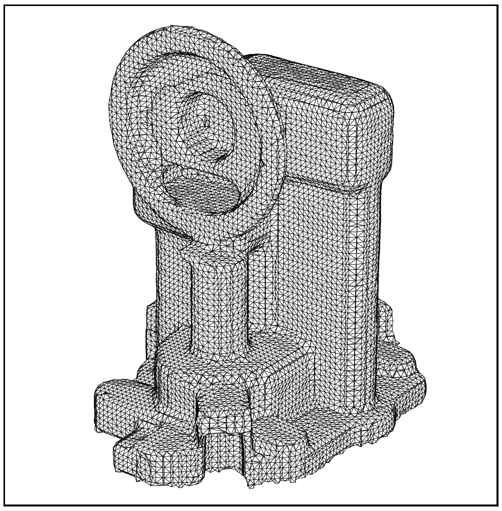
- Edges and vertices are placed near sharp features of the underlying surface.

These effects are rather surprising, since the underlying surface is unknown and can only be estimated from the discrete set of sample points. To re-emphasize, no heuristics are introduced to reap these benefits; they are simply by-products of the energy minimization process.

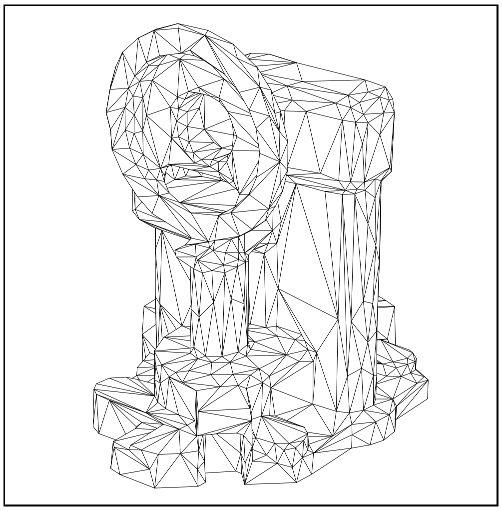
Parameter settings and quantitative results All examples use the spring constant schedule of $\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-8}\}$ as described in Section 3.3.5. Table 3.1 shows the improvements in conciseness and accuracy obtained from mesh optimization, and lists execution times (on an SGI Indigo workstation).

Surface segmentation Mesh optimization allows the detection of sharp features in the underlying surface. Using a simple thresholding method, the optimized mesh can be segmented into smooth components. To this end, we build a graph in which the nodes are the faces of mesh. Two nodes of this graph are connected if the two corresponding faces are adjacent and their dihedral angle is smaller than a given threshold. The connected components of this graph identify the desired smooth segments.

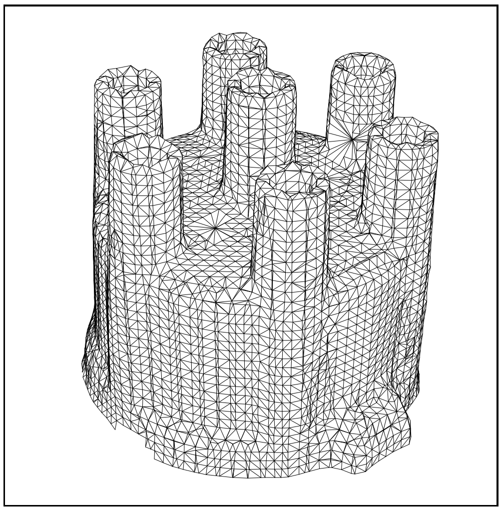
As an example, Figure 3.11a shows the segmentation of the optimized mesh into 11 components. After segmentation, a smoothly shaded surface can be created by estimating vertex normals from neighboring faces within each component, as in Figure 3.11b. Note



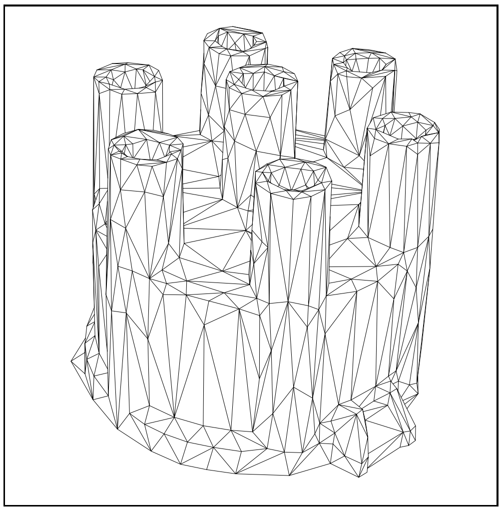
(a) Phase 1 mesh



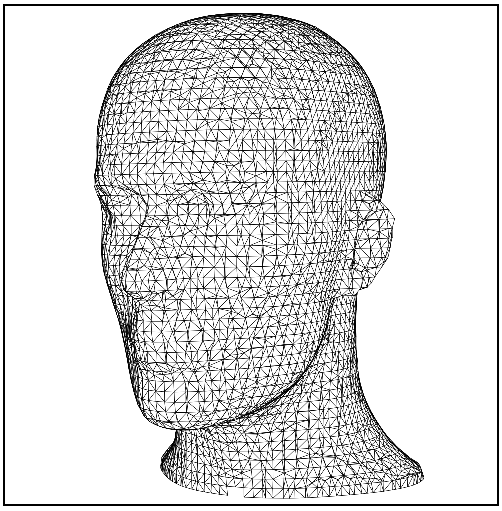
(b) Phase 2 optimized mesh



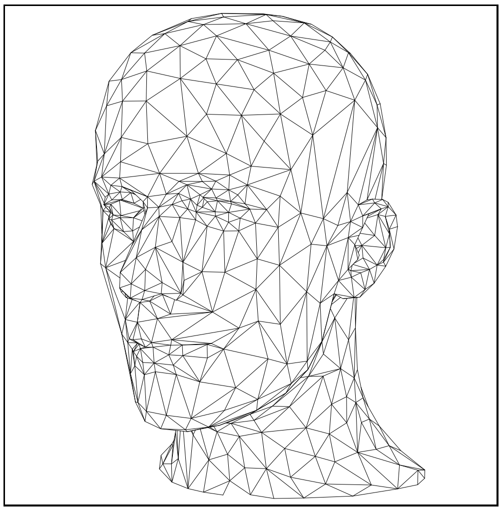
(c) Phase 1 mesh



(d) Phase 2 optimized mesh

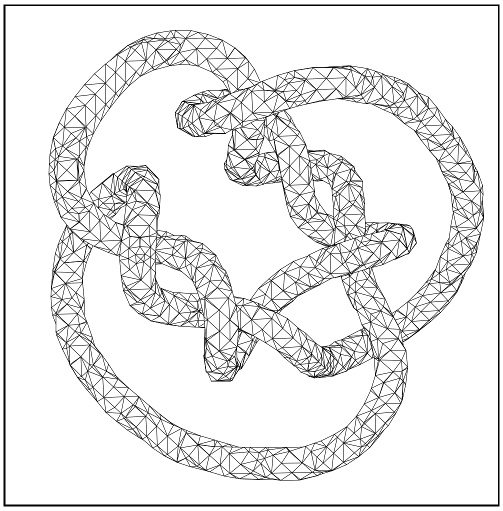


(e) Phase 1 mesh

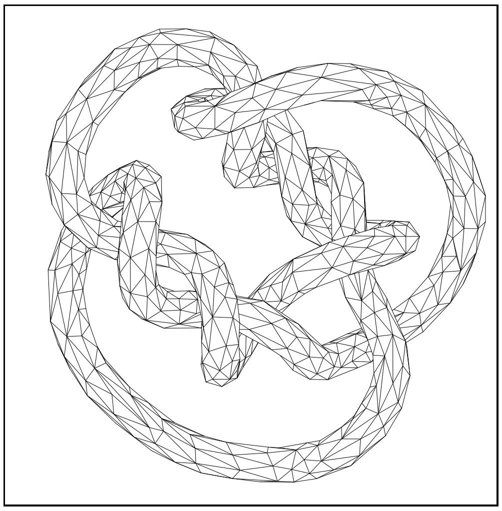


(f) Phase 2 optimized mesh

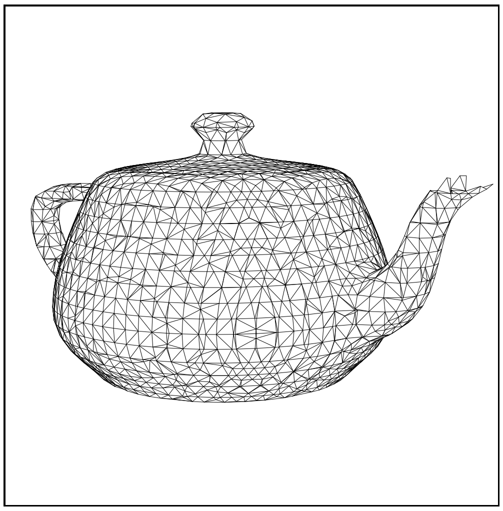
Figure 3.10: Results of phase 2 (mesh optimization). The same point sets were used as in phase 1 (Figure 2.11).



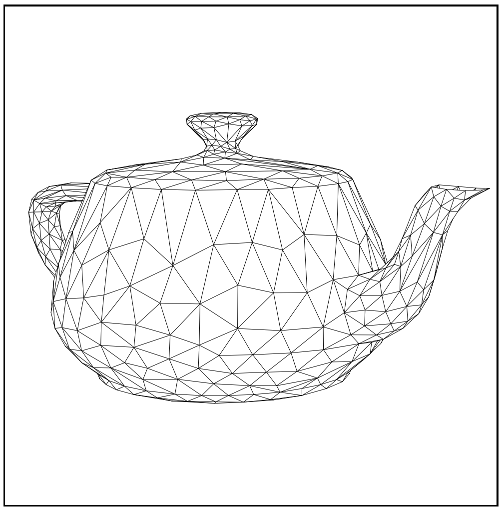
(g) Phase 1 mesh



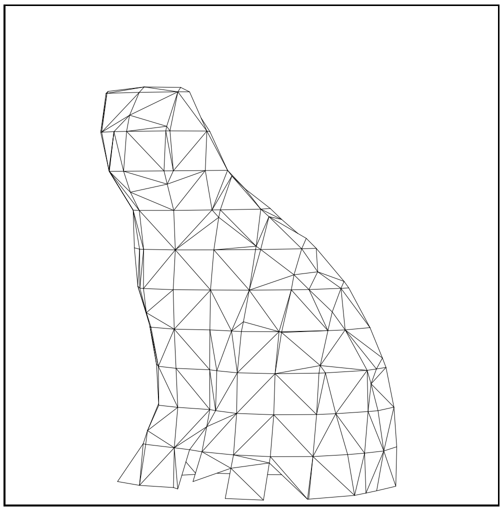
(h) Phase 2 optimized mesh



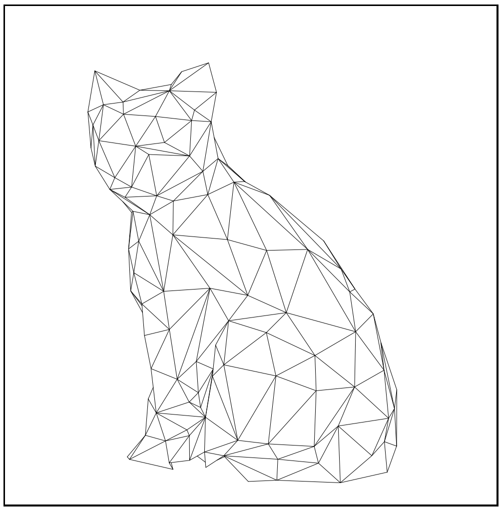
(i) Phase 1 mesh



(j) Phase 2 optimized mesh



(k) Phase 1 mesh

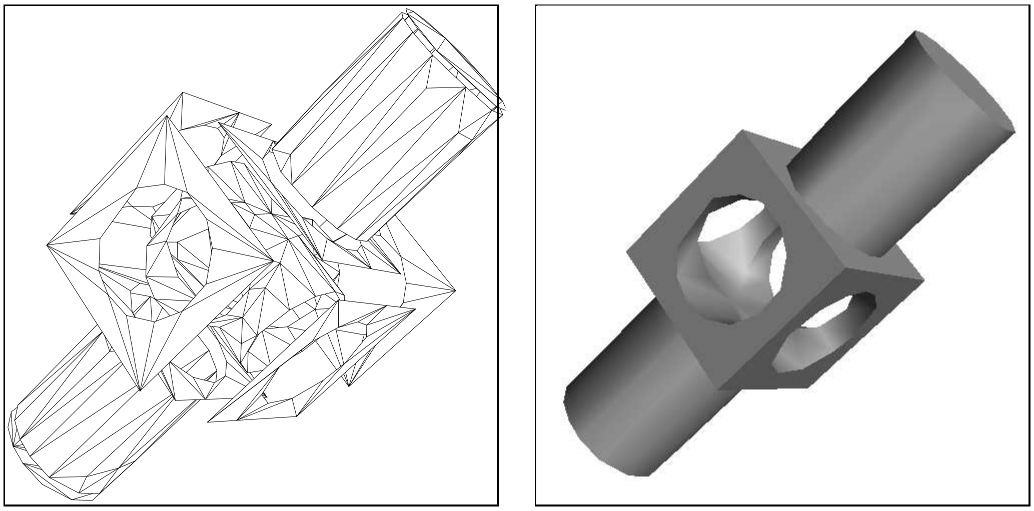


(l) Phase 2 optimized mesh

Figure 3.10: (continued)

Table 3.1: Phase 2 parameter settings and optimization results.

Fig.	Object	# pts n	c_{rep}	Conciseness m (# vertices)			Accuracy E_{dist}			Time min.
				phase 1	phase 2	$\frac{ph1}{ph2}$	phase 1	phase 2	$\frac{ph1}{ph2}$	
3.1	mechpart	4,102	10^{-5}	886	163	5	$1.72\ 10^{-1}$	$4.86\ 10^{-4}$	354	12
3.10ab	oilpmp	30,937	10^{-5}	19,002	891	21	$5.83\ 10^{-2}$	$4.93\ 10^{-3}$	12	107
3.10cd	distcap	12,745	10^{-5}	6,253	685	9	$1.30\ 10^{-1}$	$4.05\ 10^{-3}$	32	40
3.10ef	mannequin	12,772	10^{-5}	7,834	689	11	$4.74\ 10^{-2}$	$3.39\ 10^{-3}$	14	36
3.10gh	knot	10,000	10^{-5}	2,689	975	3	$1.52\ 10^{-1}$	$3.08\ 10^{-3}$	49	25
3.10ij	teapot	26,103	10^{-5}	2,986	623	5	$1.91\ 10^{-1}$	$3.17\ 10^{-3}$	60	63
3.10kl	cat	1,000	10^{-5}	232	181	1.3	$1.93\ 10^{-1}$	$1.09\ 10^{-3}$	177	2
6.2ac	curve1	200	10^{-3}	75	13	6	$8.61\ 10^{-3}$	$4.79\ 10^{-3}$	1.8	.1
6.2bd	curve2	200	10^{-2}	62	9	7	$1.12\ 10^{-1}$	$8.33\ 10^{-2}$	1.3	.1



(a) Segmented mesh (11 components) (b) Shaded rendering

Figure 3.11: Segmentation of the optimized mesh into smooth components and shaded rendering of the segmented surface.

how the silhouette of the rendered surface betrays that the surface is not smooth. The next chapter presents a method for reconstructing a piecewise smooth surface from the points.

3.5 Discussion

Alternate solution to non-linear least squares problem In Section 3.3.1 we described how to solve the continuous non-linear least squares problem using an algorithm that alternates between projection of the points and linear least squares (LLS). Another common approach to solving such a problem is to use conjugate gradients, such as the Polak-Ribiere algorithm [51]. Comparing the two, we observed that the conjugate gradient approach has a slightly better convergence rate, but that our alternating projection/LLS method reduces E much more quickly in the first few iterations.

In our procedure, the non-linear least squares problem appears twice:

1. In the initial global fitting of mesh mesh (the call to OptimizeVertexPositions on (K_0,V_0) in the pseudo-code of Figure 3.5).

2. In the evaluation of the local heuristics described in Section 3.3.4.

Since evaluating the local heuristics requires a quick estimate for $E(K')$, we have found the alternating projection/LLS method to be better suited.

On the other hand, the initial global fitting of the mesh is only done once, so for it we can afford to perform a greater number of optimization iterations. We therefore favor the faster convergence rate of the conjugate gradient method, which we will now discuss.

The conjugate gradient method is straightforward once we derive evaluation functions for the energy function $E(V)$ and its gradient,

$$\nabla E(V) = \left(\frac{\partial E}{\partial \mathbf{v}_1}, \ldots, \frac{\partial E}{\partial \mathbf{v}_m} \right) .$$

To compute $\nabla E(V)$ at a given set of vertex positions V, we project the data points $\mathbf{x}_i$ onto the mesh (Section 3.3.1.1) to obtain the barycentric coordinates $\mathbf{b}_i$ of their closest points on the mesh. By symbolic differentiation, we then obtain:

$$\begin{aligned}\left.\frac{\partial E}{\partial \mathbf{v}_j}\right|_V &= \left.\frac{\partial E_{dist}}{\partial \mathbf{v}_j}\right|_V + \left.\frac{\partial E_{spring}}{\partial \mathbf{v}_j}\right|_V \\ &= \sum_{i=1}^{n} -2b_{i,j}(\mathbf{x}_i - \pi_V(\mathbf{b}_i)) + \sum_{\{j,k\}\in K} -2\kappa(\mathbf{v}_k - \mathbf{v}_j)\,.\end{aligned}$$

Since the vectors $\mathbf{b}_1, \ldots, \mathbf{b}_m$ are sparse, and since the number of edges in K is $O(m)$, both $E(V)$ and $\nabla E(V)$ can be computed in $O(n+m)$ time given a fast projection method.

It is interesting to note that $E(V)$ is continuous but not differentiable. Indeed, it may happen that some points do not project uniquely onto the mesh, in which case $\nabla E(V)$ will be undefined. However, this seldom occurs, and if it does, our projection procedure makes an arbitrary choice which leads to one of several "possible" values for $\nabla E(V)$.

Chapter 4

PHASE 3:
PIECEWISE SMOOTH
SUBDIVISION SURFACE OPTIMIZATION

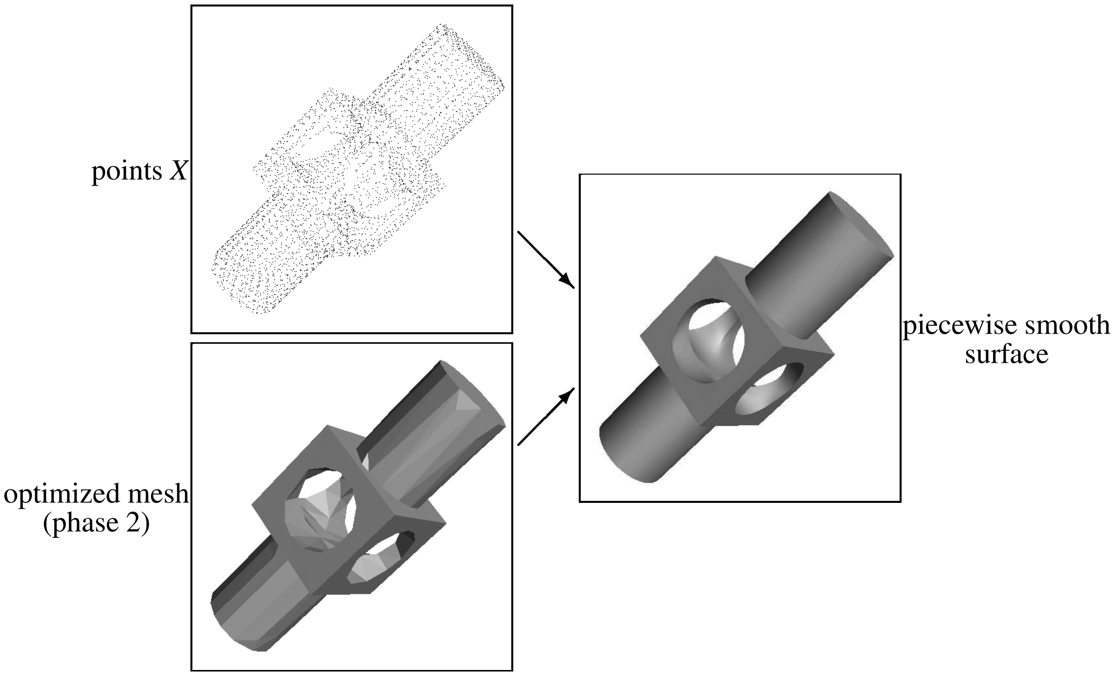


Figure 4.1: Phase 3: from a piecewise linear to a piecewise smooth representation.

4.1 Introduction

As detailed in the previous two chapters, phases 1 and 2 of the surface reconstruction procedure create an accurate, concise piecewise linear surface approximation to a set of points. Phase 3, described in this chapter, finds an even more accurate and more concise piecewise smooth surface (Figure 4.1). A key ingredient in phase 3, and a principal contribution of this thesis, is the introduction of a new class of piecewise smooth surfaces based on subdivision.

The generalization to smooth surfaces in phase 3 is a natural and necessary extension of phase 2. Many objects of interest are piecewise smooth; that is, their surfaces consist of smoothly curved regions that meet along sharp curves and at sharp corners. Modeling such objects as piecewise linear surfaces typically requires a large number of triangles, whereas curved surface models can provide both a more accurate and a more concise representation of the true surface. It is critical, however, to use a surface representation capable of explicitly modeling sharp features. Using an everywhere smooth surface representation to model sharp features typically results in a greater number of surface elements, poor geometric fit, and unwanted surface artifacts, as illustrated in Figure 4.2. Additionally, the surface representation should be capable of modeling surfaces of arbitrary topological type.

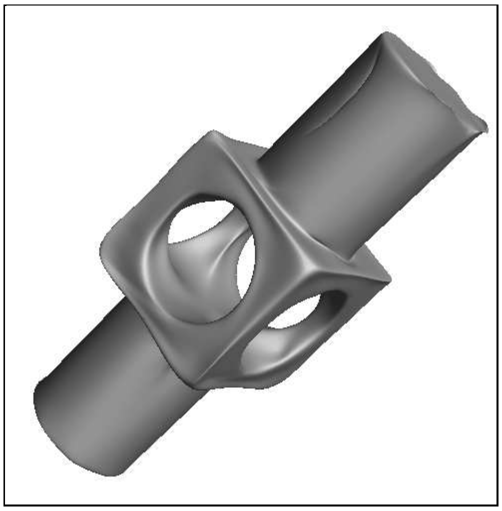


Figure 4.2: Poor geometric fit when using an everywhere smooth surface. (E_{dist} is 6 times larger than in Figure 4.3b.)

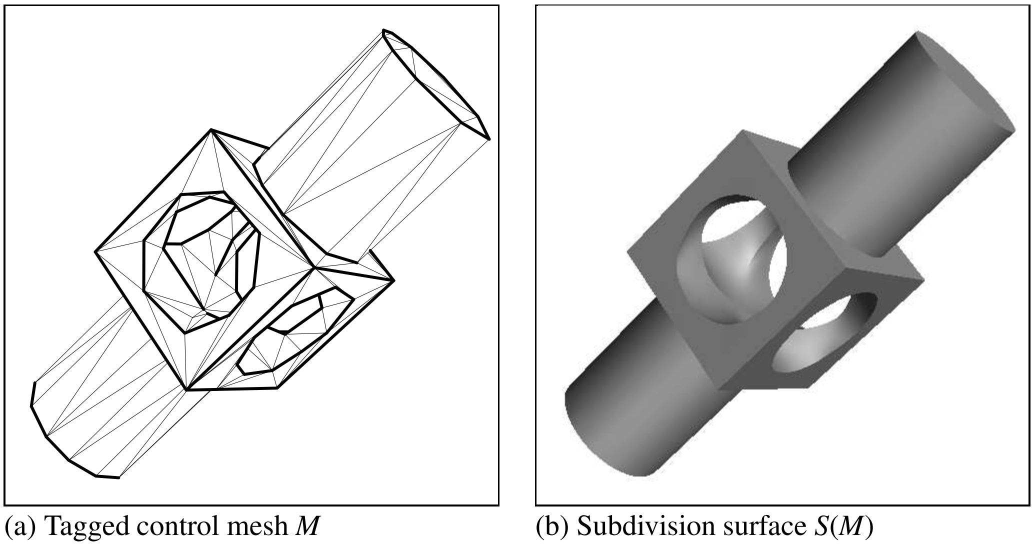
The most popular smooth surface representations are tensor product NURBS. However, NURBS can only represent surfaces of arbitrary topological type by partitioning the model into a collection of individual NURBS patches. Adjacent patches must then be explicitly stitched together using geometric continuity conditions [20]. A large number of parameters (the B-spline coefficients) are therefore introduced, most of which are constrained by the continuity conditions. As a consequence, fitting NURBS surfaces in general requires high-dimensional constrained optimization.

Subdivision surfaces, first introduced by Doo/Sabin [14] and Catmull/Clark [11], offer a promising alternative. As will be detailed in Section 4.2, a subdivision surface $S(M)$

is defined as the limit of a subdivision process applied to a control mesh M, as indicated in Figures 4.5a–4.5d. Subdivision surfaces are capable of modeling everywhere smooth surfaces of arbitrary topological type using a small number of unconstrained parameters.

Our new surface representation is a generalization of the subdivision surface scheme introduced by Loop [33]. Loop's scheme, like all subdivision schemes to date, produces tangent plane continuous surfaces of arbitrary topological type. A principal contribution of our work is to show that it is possible to locally modify Loop's subdivision rules to model sharp features such as creases and corners. Our *piecewise smooth subdivision scheme* also models boundary curves, as shown for instance in the spout of the Utah teapot (Figure 4.15j).

We can now rephrase the goal of phase 3 as: Starting with the optimized mesh (piecewise linear surface) produced by phase 2, find a concise control mesh M (Figure 4.3a), of the same topological type as the phase 2 mesh, defining a piecewise smooth subdivision surface $S(M)$ (Figure 4.3b) that accurately fits the points.



(a) Tagged control mesh M (b) Subdivision surface $S(M)$

Figure 4.3: Example of subdivision surface optimization.

As in phase 2, we set up an energy minimization problem that trades off conciseness and fit to the data. Both the energy function and the optimization algorithm of phase 3 are similar to those of phase 2, but with two major differences:

- The distance energy E_{dist} measures distance of the points not to the piecewise linear control mesh M, but to the piecewise smooth subdivision surface $S(M)$ defined by M.

- In addition to varying the number of vertices in the control mesh, their connectivity, and their positions, the phase 3 optimization algorithm also varies the number and locations of sharp features. The automatic detection and recovery of sharp features in the surface is an essential part of phase 3.

The search space of the subdivision surface optimization consists of all piecewise smooth subdivision surfaces of a given topological type. As illustrated in Figure 4.4, this search space is a superset of the space of meshes considered in phase 2.

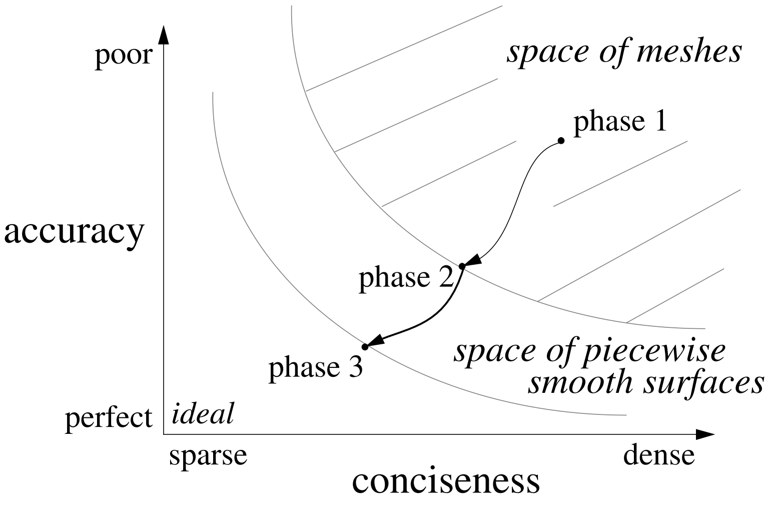


Figure 4.4: Trade-off between accuracy and conciseness in phase 3.

We introduce subdivision surfaces and review some of their properties in Section 4.2. Our piecewise smooth subdivision surface scheme is presented in Section 4.3. The phase 3 optimization problem and algorithm are described in Sections 4.4 and 4.5.

4.2 Background on subdivision surfaces

A *subdivision surface* is defined by repeatedly refining a control mesh as indicated in Figures 4.5a–4.5d. The first and most popular subdivision surface schemes, introduced by Doo/Sabin [14] and Catmull/Clark [11], are based on quadrilateral meshes, and generalize biquadratic and bicubic tensor product B-splines, respectively. A subdivision scheme based on triangles is most convenient for our purposes. We use a generalization of the triangular scheme introduced by Loop [33], as it is the simplest known scheme leading to tangent plane smooth surfaces.

4.2.1 Loop's subdivision surface scheme

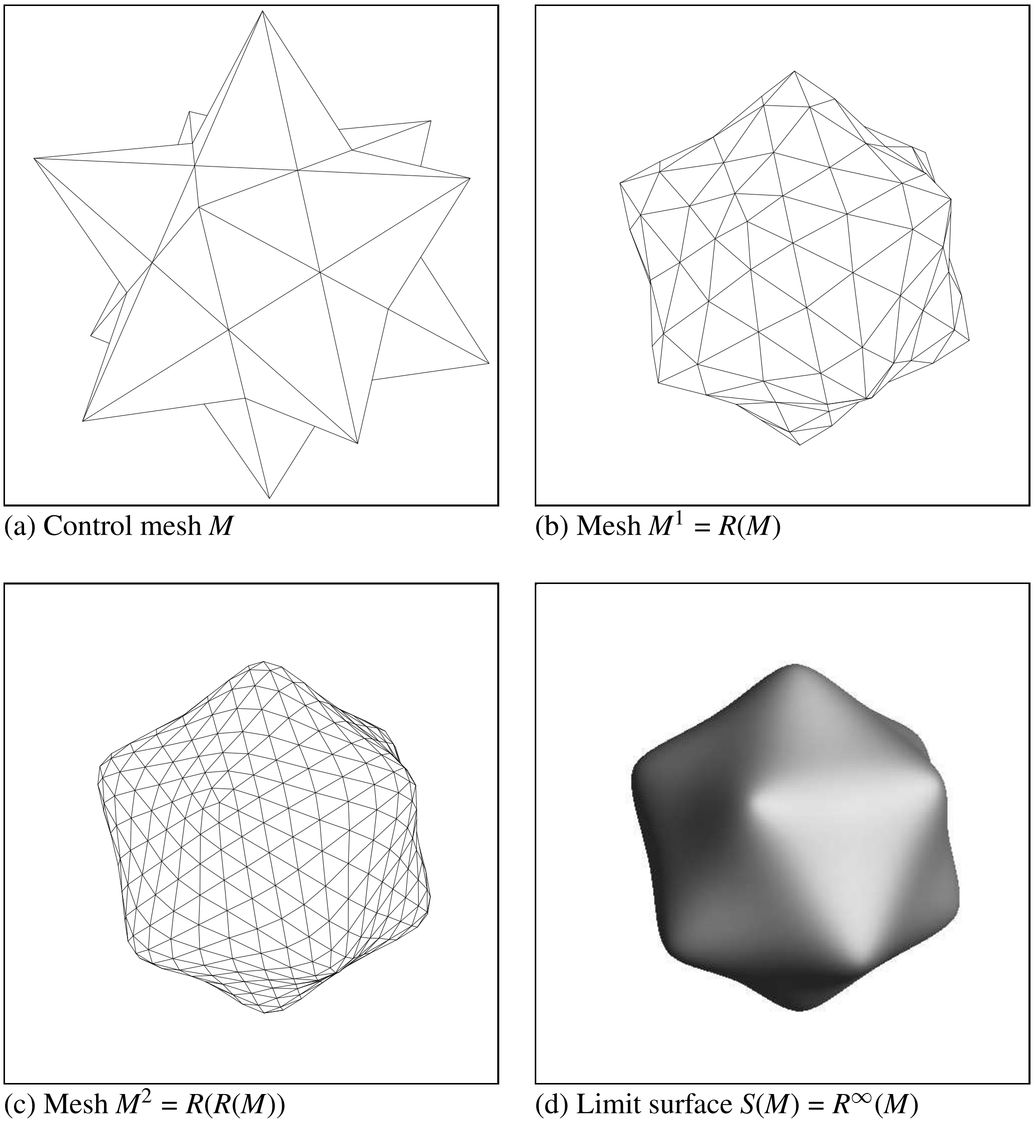
Loop's subdivision scheme is a generalization of C^2 quartic triangular B-splines. As illustrated in Figure 4.5, the subdivision surface $S(M)$ associated with a control mesh $M = (K, V)$ is defined as the limit of a refinement process applied to M:

$$M, \quad M^1 = R(M), \quad M^2 = R(R(M)), \quad \ldots .$$

The refinement procedure R proceeds by splitting each triangular face into four subfaces. The vertices of the refined mesh are then positioned using weighted averages of the vertices in the unrefined mesh. Formally, starting with the initial control mesh $M = M^0$, each subdivision step carries a mesh $M^r = (K^r, V^r)$ into a refined mesh $M^{r+1} = (K^{r+1}, V^{r+1})$ where the vertices V^{r+1} are computed as affine combinations of the vertices of V^r. Some of the vertices of V^{r+1} naturally correspond to vertices of V^r—these are called *vertex points*; the remaining vertices in V^{r+1} correspond to edges of the mesh M^r—these are called *edge points*. Let $\mathbf{v}^r$ denote a vertex of V^r having neighbors $\mathbf{v}^r_1, \ldots, \mathbf{v}^r_n$ as shown in Figure 4.6. Such a vertex is said to have valence n. Let $\mathbf{v}^{r+1}_i$ denote the edge point of V^{r+1} corresponding to the edge $\mathbf{v}^r\mathbf{v}^r_i$, and let $\mathbf{v}^{r+1}$ be the vertex point of V^{r+1} associated with $\mathbf{v}^r$. The positions of $\mathbf{v}^{r+1}$ and $\mathbf{v}^{r+1}_i$ are computed according to the subdivision rules

$$\begin{aligned} \mathbf{v}^{r+1} &= \frac{\alpha(n)\mathbf{v}^r + \mathbf{v}^r_1 + \cdots + \mathbf{v}^r_n}{\alpha(n) + n} \\ \mathbf{v}^{r+1}_i &= \frac{3\mathbf{v}^r + 3\mathbf{v}^r_i + \mathbf{v}^r_{i-1} + \mathbf{v}^r_{i+1}}{8}, \qquad i = 1, \ldots, n \end{aligned} \tag{4.1}$$

where subscripts are taken modulo n, and where $\alpha(n) = \frac{n(1-a(n))}{a(n)}$ with $a(n) = \frac{5}{8} - \frac{(3+2\cos(2\pi/n))^2}{64}$. Affine combinations such as those in Equation 4.1 can be nicely visualized by diagrams



(a) Control mesh M

(b) Mesh $M^1 = R(M)$

(c) Mesh $M^2 = R(R(M))$

(d) Limit surface $S(M) = R^\infty(M)$

Figure 4.5: Example of Loop's subdivision surface scheme.

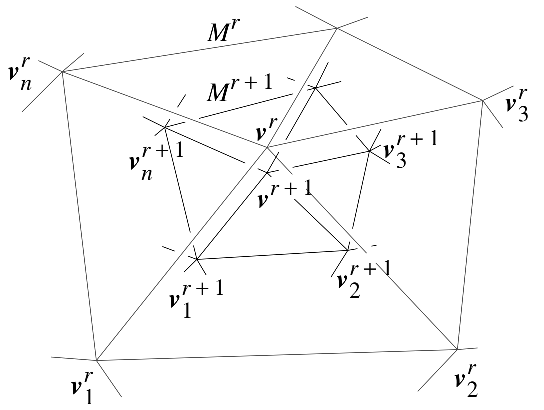


Figure 4.6: The neighborhood around a vertex $\mathbf{v}^r$ of valence n.

called *masks*, as shown in Figure 4.7.

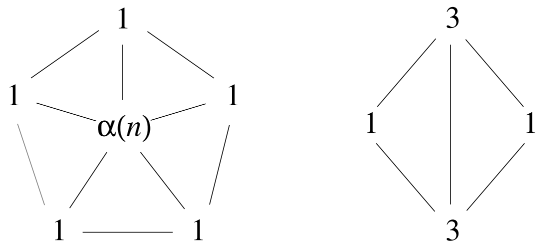


Figure 4.7: Vertex and edge subdivision masks for Loop's subdivision surface scheme.

4.2.2 Computing surface points and tangent vectors

Loop's surfaces in particular, and subdivision surfaces in general, are defined only as the limit of an infinite refinement process. In most cases closed form expressions for the limit surfaces are not known, but somewhat surprisingly, various properties of subdivision surfaces, such as exact points on the surface and exact tangent planes, can nonetheless be computed [26].

To study the properties of subdivision surfaces, it is convenient to write Equation 4.1 in matrix form as

$$\begin{aligned}(\mathbf{v}^{r+1}, \mathbf{v}_1^{r+1}, ..., \mathbf{v}_n^{r+1})^T &= S_n(\mathbf{v}^r, \mathbf{v}_1^r, ..., \mathbf{v}_n^r)^T \\ &= S_n^{r+1}(\mathbf{v}^0, \mathbf{v}_1^0, ..., \mathbf{v}_n^0)^T \qquad (4.2)\end{aligned}$$

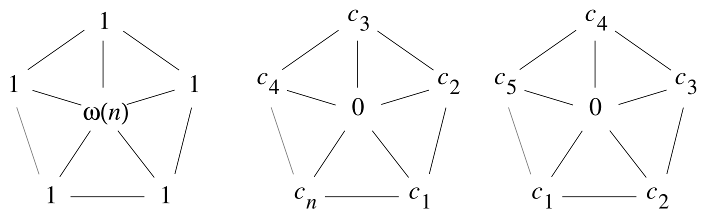
where superscript T denotes matrix transpose [14]. For Loop's subdivision scheme, the matrix S_n, called the *local subdivision matrix*, has the form

$$S_n^{\text{Loop}} = \frac{1}{8}\begin{pmatrix} 8-8a(n) & \frac{8a(n)}{n} & \frac{8a(n)}{n} & \frac{8a(n)}{n} & \frac{8a(n)}{n} & \cdots & \frac{8a(n)}{n} \\ 3 & 3 & 1 & 0 & 0 & \cdots & 1 \\ 3 & 1 & 3 & 1 & 0 & \cdots & 0 \\ \vdots & & & & & \ddots & \vdots \\ 3 & 1 & 0 & 0 & \cdots & 1 & 3 \end{pmatrix}.$$

As $r \to \infty$, each point $\mathbf{v}^r$ approaches a point on the limit surface. Equation 4.2 suggests that the limit point can be obtained by analyzing the eigenstructure of the local subdivision matrix. Indeed, the limit point can be expressed as an affine combination of the initial vertex positions [26]:

$$\mathbf{v}^\infty = \frac{\ell_0\mathbf{v}^0 + \ell_1\mathbf{v}_1^0 + \cdots \ell_n\mathbf{v}_n^0}{\ell_0 + \ell_1 + \cdots \ell_n}$$

where $(\ell_0, ..., \ell_n)$ is the dominant left eigenvector of S_n. For Loop's surfaces this affine combination can be expressed as the *position mask* shown in Figure 4.8a [33].



(a) position mask (b) tangent masks

Figure 4.8: Position and tangent masks for Loop's subdivision scheme, where $\omega(n) = \frac{3n}{8a(n)}$, and where $c_i = \cos(2\pi i/n)$.

Eigenanalysis of the local subdivision matrix can also be used to establish smoothness. It can be shown, for instance, that Loop's surfaces are indeed tangent plane continuous [33, 52]. Moreover, Halstead *et al.* [26] show that the tangent vectors to the limit surface at $\mathbf{v}^\infty$ can be computed using the two left eigenvectors of S_n corresponding to the second largest eigenvalue (this eigenvalue has multiplicity 2). For Loop's surfaces the vectors

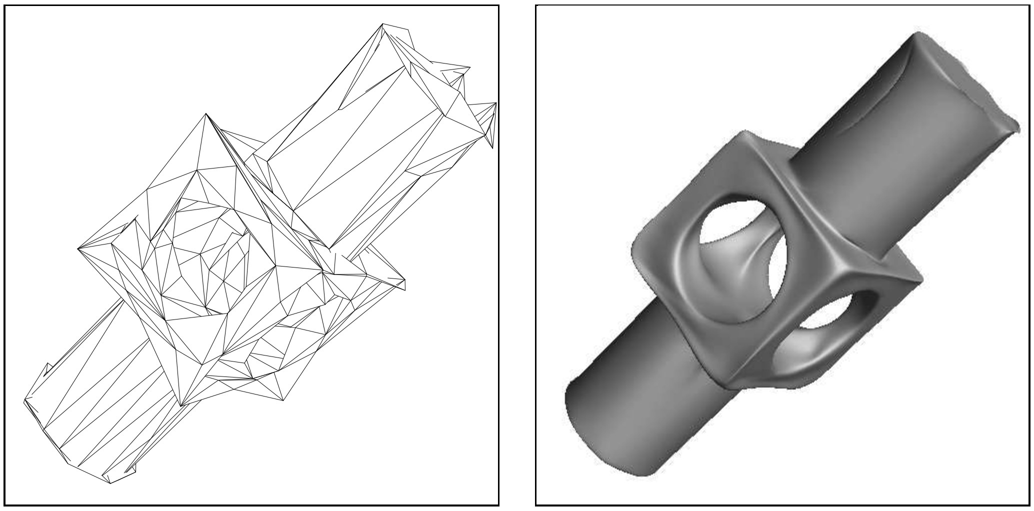
$$\begin{aligned} \mathbf{u}_1 &= c_1\mathbf{v}_1^0 + c_2\mathbf{v}_2^0 + \cdots + c_n\mathbf{v}_n^0 \\ \mathbf{u}_2 &= c_2\mathbf{v}_1^0 + c_3\mathbf{v}_2^0 + \cdots + c_1\mathbf{v}_n^0, \end{aligned} \tag{4.3}$$

with $c_i = \cos(2\pi i/n)$ span the tangent plane of the limit surface. Their cross product therefore gives an exact normal vector to the surface which is useful, for example, to create Phong-shaded renderings such as those shown in Figure 4.15. The formulas given in Equation 4.3 can be visualized as the *tangent masks* shown in Figures 4.8b.

Eigenanalysis will again be used in Section 4.3.2 to study the properties of piecewise smooth subdivision surfaces.

4.3 Piecewise smooth subdivision surfaces

Attempting to fit smooth surfaces to non-smooth objects often produces unacceptable results. As an example, fitting an everywhere smooth subdivision surface to the points of the mechanical part produces the surface shown in Figure 4.9b. The control mesh for this surface, shown in Figure 4.9a, is rather unwieldy.



(a) Control mesh (b) Subdivision surface

Figure 4.9: Result of optimizing an everywhere smooth subdivision surface.

To accurately model objects with tangent discontinuities, we develop new subdivision rules that introduce a set of commonly occurring sharp features that we call *creases*, *corners*, and *darts*, as illustrated in Figure 4.10. A crease is a curve along which the surface is C^0 but not C^1; a corner is a point where three or more creases meet; finally, a dart is a point on the interior of a surface where a crease terminates. Although this list of sharp features is not exhaustive (for instance, we cannot model a cone or two coincident darts), it has proven sufficient for the examples we have encountered.

Subdivision surfaces produced by the new rules are tangent plane smooth everywhere except along creases and at corners. A detailed theoretical analysis of the behavior along creases and at corners is beyond the scope of this thesis and will be presented by Schweitzer and Duchamp [64]. In Section 4.3.2 we summarize the relevant results of the analysis.

4.3.1 Subdivision rules

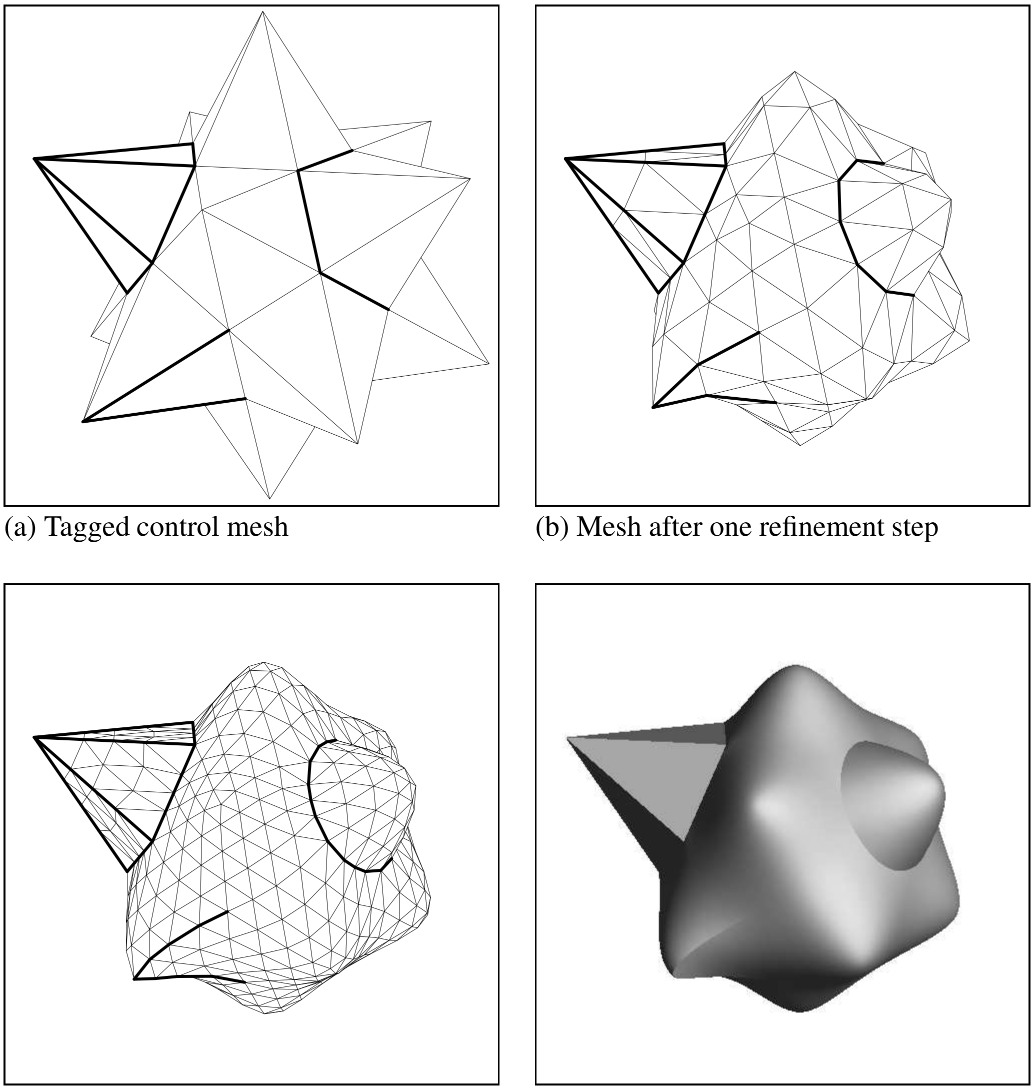
To model creases, corners, and darts using subdivision surfaces, a subset L of edges in the simplicial complex K is tagged as *sharp*. We refer to the pair (K, L) as a *tagged simplicial complex*. The subdivision masks are modified so that tangent plane continuity across sharp edges is relaxed. Boundary curves are produced by tagging all boundary edges of the mesh as sharp.[1] In the subdivision process, edges created through refinement of a sharp edge are tagged as sharp.

Subdivision rules at crease vertices must be chosen carefully in order for the surface to have a well-defined tangent plane on each side of the crease. Similar considerations apply to corners and darts. It should be noted that the specific subdivision masks we use are by no means unique. Indeed, there is considerable flexibility in selecting them. The masks we present here are simple and have worked well in practice, but further research should be done to explore other alternatives.

We classify vertices into five different types based on the number and arrangement of incident edges. A *smooth vertex* is one where the number of incident sharp edges s is zero; a *dart vertex* has $s = 1$; a *crease vertex* has $s = 2$; and a *corner vertex* has $s > 2$. Crease vertices are further classified as *regular* and *non-regular* depending on the arrangement of smooth edges. A crease vertex is regular if and only if there are exactly 2 smooth edges on each side of the crease (or on the one side if on a boundary).

Figure 4.11 shows our vertex and edge subdivision masks. As indicated in the figure, vertex subdivision masks are chosen based on the type of the vertex. We use three different types of edge subdivision masks. A smooth edge (one not tagged as sharp) is subdivided using the smooth edge subdivision mask. The mask used to subdivide a sharp edge depends on the types of the incident vertices as shown in Table 4.1. When applying the non-

[1] In related work, Nasri [44, 45] developed a method to model boundary curves in a Doo-Sabin subdivision procedure by augmenting the control mesh rather than by modifying the subdivision masks.



(a) Tagged control mesh

(b) Mesh after one refinement step

(c) Mesh after two refinement steps

(d) Piecewise smooth limit surface

Figure 4.10: Example of our piecewise smooth subdivision surface scheme. Sharp edges $L \subset K$ are drawn as bold line segments.

symmetric edge subdivision mask 3, the regular crease vertex incident to the edge receives the weight 5.

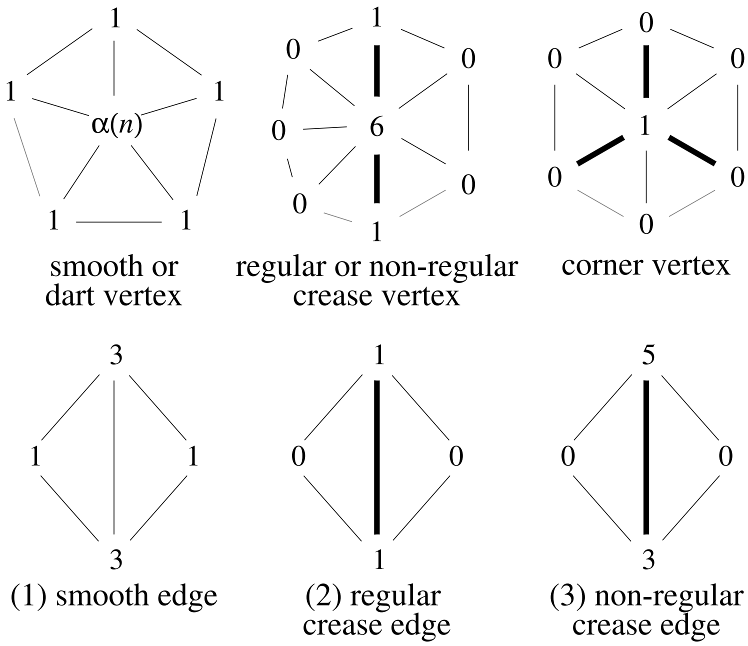


Figure 4.11: Vertex and edge subdivision masks for our piecewise smooth scheme. Bold lines denote sharp edges.

Table 4.1: Assignment of sharp edge subdivision masks as a function of the types of the two incident vertices. Masks are numbered as shown in Figure 4.11.

	dart	reg. crease	non-reg. crease	corner
dart	1	1	1	1
regular crease	1	2	3	3
non-regular crease	1	3	2	2
corner	1	3	2	2

Those familiar with B-spline curve subdivision may recognize that the crease subdivision masks have been designed so that the sharp edges converge to uniform cubic B-splines except near non-regular crease and corner vertices. The zeros in these crease subdivision masks completely decouple the behavior of the surface on one side of the crease from the behavior on the other side.

Since all subdivision masks are convex combinations (i.e. their entries are non-negative), the piecewise smooth subdivision surface retains the convex hull property—meaning that the surface is contained within the convex hull of its control vertices. This property, shared by other surface representations like tensor product B-splines, is useful for algorithms that use a divide-and-conquer strategy.

4.3.2 Computing surface points and tangent vectors

As explained in Section 4.2.2, limiting points and tangent planes can be computed using masks. These masks are determined by the eigenstructure of local subdivision matrices, which depend on the type of the vertex (smooth, dart, regular and non-regular crease, and corner).

Smooth and dart vertices: At smooth and dart vertices, our local subdivision matrix is identical to Loop's subdivision matrix. The position and tangent masks are therefore as in Figure 4.8.

Crease vertices: Since the zeros in the crease subdivision masks (Figure 4.11) decouple the behavior of the surface on one side of the crease from the behavior on the other side, we can decouple the analysis, focusing on a local subdivision matrix that describes the behavior on *one side* of the crease. As indicated earlier, boundary curves are modeled as one-sided creases.

In the following, we assume that the vertices $\mathbf{v}_1^0, \ldots, \mathbf{v}_n^0$ surrounding one side of a crease vertex $\mathbf{v}^0$ are indexed as shown in Figure 4.12d. We also assume that no two non-regular crease vertices are adjacent to each other; this assumption always holds after one iteration of subdivision, since all newly introduced vertices are either smooth or regular crease vertices.

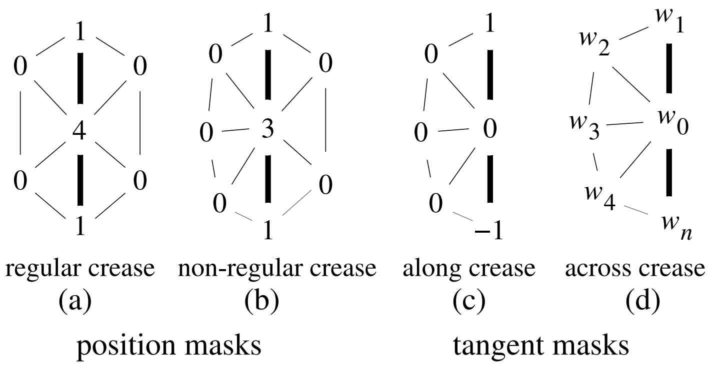


Figure 4.12: Position and tangent masks for crease vertices.

At a regular crease vertex, the valence is 4 and the local subdivision matrix is

$$S^{\text{reg crease}} = \frac{1}{8}\begin{pmatrix} 6 & 1 & 0 & 0 & 1 \\ 4 & 4 & 0 & 0 & 0 \\ 3 & 1 & 3 & 1 & 0 \\ 3 & 0 & 1 & 3 & 1 \\ 4 & 0 & 0 & 0 & 4 \end{pmatrix}.$$

The dominant left eigenvector of this matrix yields the position mask $(4, 1, 0, 0, 1)$ shown in Figure 4.12a, meaning that

$$\mathbf{v}^{\infty} = \frac{1}{6}(4\mathbf{v}^0 + \mathbf{v}_1^0 + \mathbf{v}_n^0)$$

is a point on the limit crease.[2]

Similarly, when the crease vertex is non-regular, the local subdivision matrix has the form

$$S_n^{\text{non-reg crease}} = \frac{1}{8}\begin{pmatrix} 6 & 1 & 0 & 0 & 0 & 0 & \cdots & 1 \\ 3 & 5 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 3 & 1 & 3 & 1 & 0 & 0 & \cdots & 0 \\ 3 & 0 & 1 & 3 & 1 & 0 & \cdots & 0 \\ \vdots & & & & & & \ddots & \vdots \\ 3 & 0 & 0 & 0 & 0 & 0 & \cdots & 5 \end{pmatrix},$$

[2] This equation is consistent with the observation that, away from non-regular crease vertices, the crease is a uniform cubic B-spline curve.

and we obtain the position mask shown in Figure 4.12b.

For crease vertices of valence 4 or higher, the subdivision rules described in the previous section give rise to well-defined tangent planes on both sides of the crease [64].[3] As for smooth vertices, tangent masks are again determined by the two left eigenvectors corresponding to the 2nd and 3rd largest eigenvalues. For both regular and non-regular crease vertices, a tangent along the crease is obtained by the tangent mask shown in Figure 4.12c. To compute a tangent vector transverse to the crease, we use the tangent mask shown in Figure 4.12d, where the weights are defined as follows [64]. At a regular crease vertex, the valence is 4 and the mask is given by $(w_0, ..., w_4) = (-2, -1, 2, 2, -1)$. At a non-regular crease vertex, for $n \geq 4$, $w_0 = 0$, $w_1 = w_n = \sin\theta$, and $w_i = (2\cos\theta - 2)(\sin(i-1)\theta)$ for $i = 2, \ldots, (n-1)$ where $\theta = \pi/(n-1)$; for $n = 3$, $(w_0, ..., w_3) = (-1, 0, 1, 0)$; finally, for $n = 2$, $(w_0, w_1, w_2) = (-2, 1, 1)$.

Corner vertices: The subdivision masks at a corner vertex are much like those at a crease vertex. If the corner vertex has s sharp edges, the local subdivision matrix decouples into s separate matrices (or $s-1$ matrices if the corner vertex lies on a boundary), each describing a smooth region of the surface. After the first subdivision step, since the crease vertices adjacent to a corner vertex are always regular crease vertices, each such matrix has the form

$$S_n^{\text{corner}} = \frac{1}{8}\begin{pmatrix} 8 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 3 & 5 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 3 & 1 & 3 & 1 & 0 & 0 & \cdots & 0 \\ 3 & 0 & 1 & 3 & 1 & 0 & \cdots & 0 \\ \vdots & & & & & & \ddots & \vdots \\ 3 & 0 & 0 & 0 & 0 & 0 & \cdots & 5 \end{pmatrix}.$$

Since the corner vertex is stationary during subdivision, it is itself a point on the surface; equivalently, $(1, 0, ..., 0)$ is the dominant left eigenvector of S_n^{corner}. The second largest eigenvalue has multiplicity 2 and the two corresponding left eigenvectors define the tangent masks $(1, -1, 0, ..., 0)$ and $(1, 0, 0, ..., -1)$.

[3] The techniques used to prove smoothness do not apply to vertices of valence 2 and 3, although numerical experiments suggest that tangent planes are well-defined in these cases too.

4.4 Definition of the energy function

As outlined in Section 4.1, the input to phase 3 is an unstructured collection $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ of data points together with the mesh obtained from phase 2. Phase 3 seeks to find a concise *tagged mesh* $M = (K, L, V)$ defining a piecewise smooth subdivision surface $S(M)$ that accurately fits X. We use the mesh produced by phase 2 as the initial estimate (K_0, V_0), and let the initial set L_0 of sharp edges contain the edges whose dihedral angles are above a threshold (e.g. 40 degrees).

Note that the goals of phase 3 are the same as those of phase 2: to find a surface that both provides a good fit to a set of points and has concise representation. As in phase 2 (Section 3.2), we cast the problem as one of minimizing an energy function that captures the competing goals of conciseness and accuracy.

The energy function is given by

$$E(K, L, V) = E_{dist}(K, L, V) + c_{rep}m + c_{sharp}e$$

where

- E_{dist} is the total squared distance from the points to the subdivision surface;
- $c_{rep}m$ is a penalty on the number m of vertices;
- $c_{sharp}e$ is a penalty on the number e of sharp edges.

As in phase 2, the parameter c_{rep} controls the trade-off between conciseness and fidelity to the data and should be set by the user. The parameter c_{sharp} controls the trade-off between smoothness of the surface and fidelity to the data. Setting $c_{sharp} = c_{rep}/5$ has worked well in all our examples.

We minimize the energy function over the space $\mathcal{M}$ of tagged meshes $M = (K, L, V)$ where K is of the same topological type as the phase 2 mesh, and L is the subset of sharp edges of K. The goal is to find the tagged mesh in $\mathcal{M}$ that minimizes E.

Note the absence of a "spring energy" term, which was introduced in phase 2 to guide the mesh optimization algorithm into a good local energy well. For the type of data we have used, such an energy term has been unnecessary in phase 3.

4.5 Minimization of the energy function

The energy minimization algorithm closely parallels the one used in phase 2 (Section 3.3). We decompose the problem into two nested subproblems: an inner, continuous optimization over the control vertex positions V for fixed (K, L), and an outer, discrete optimization over (K, L).

4.5.1 Optimization over V for fixed (K, L)

In the inner minimization, we hold the tagged simplicial complex (K, L) fixed and consider the continuous non-linear optimization over V. We want to determine

$$E(K, L) = \min_V E_{dist}(K, L, V) + c_{rep}m + c_{sharp}e \,,$$

the minimum energy for fixed (K, L). Since m and e are fixed, this is equivalent to minimizing the distance energy over the vertex positions V. In the following, V is treated as an $m \times 3$ matrix whose rows contain the (x, y, z) coordinates of the vertices.

Computing the distance energy E_{dist} involves projecting the data points $\mathbf{x}_i$ onto the subdivision surface $S(M)$. This is not feasible in practice as the surface is defined only as the limit of an infinite process. Instead, we project onto a piecewise linear approximation $\tilde{M}^r$ to $S(M)$ obtained by subdividing the original mesh r times to produce a refined mesh $M^r = R^r(M)$, and then pushing all the vertices of M^r to their limit positions using the position masks. (Typically we use $r = 2$.) Since each of the vertices of M^r can be written as an affine combination of the vertices V of M (using the subdivision rules), and since the position masks are affine, by composition each of the vertices of $\tilde{M}^r$ can be written as an affine combination of the vertices V. That is, each vertex $\tilde{\mathbf{v}}^r$ of $\tilde{M}^r$ can be written as $\tilde{\mathbf{v}}^r = \mathbf{y}V$, where the entries of the row vector $\mathbf{y}$ can be computed by composing the effects of r-fold subdivision followed by application of a position mask. Moreover, since $\tilde{M}^r$ is piecewise linear, every point on $\tilde{M}^r$—not just the vertices—can be written as an affine combination of the vertices V.

For each data point $\mathbf{x}_i$, let $\mathbf{z}_i$ be the closest point on $\tilde{M}^r$. As argued above, $\mathbf{z}_i$ can be written as $\mathbf{y}_i V$, meaning that E_{dist} can be expressed as

$$E_{dist} = \sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{y}_i\, V\|^2 \,.$$

This expression for E_{dist} is quadratic in V. Hence, for fixed $\mathbf{y}_i$, optimizing over V is a linear least squares problem. Moreover, the vectors $\mathbf{y}_i$ are sparse since the subdivision rules are local.

This suggests an iterative minimization scheme alternating between the following steps:

1. For fixed V, compute the projections $\mathbf{y}_i V$ of the data points $\mathbf{x}_i$ onto $\tilde{M}^r$.

2. For fixed $\mathbf{y}_1, \cdots, \mathbf{y}_n$, optimize E_{dist} over V.

Step 2, which is a sparse linear least squares problem, can be solved using a sparse, iterative conjugate gradient method, as described in Section 3.3.1. Since the rows of the design matrix (the $\mathbf{y}_i$'s) have approximately 12 non-zero entries on average (vs. 3 in phase 2), the sparse conjugate gradient solution is more expensive than that in phase 2, but only by a constant factor.

4.5.2 Optimization over (K, L)

Our algorithm for solving the outer minimization problem, minimizing $E(K, L)$, again closely parallels the phase 2 algorithm of Section 3.3.2.

We define a set of four elementary mesh transformations, *edge collapse*, *edge swap*, *edge split*, and *edge tag*, taking a tagged simplicial complex (K, L) to another tagged simplicial complex (K', L'), as shown in Figure 4.13. The first three transformations were discussed in Section 3.3.2. The fourth transformation, edge tag, is a toggle that either adds an edge to the set L of sharp edges, or removes one from it. As in phase 2, these four transformations are complete in the sense that they form a transitive set of transformations on the set of tagged simplicial complexes (of a given topological type).

A *legal move* is the application of one of these elementary transformations to an edge of K that leaves the topological type of K unchanged. The criterion for determining whether a move is legal was given in Section 3.3.2. Our goal is to find a sequence of legal moves taking us from an initial tagged simplicial complex (K_0, L_0) to one for which a minimum of E is achieved.

As in phase 2, this is accomplished via a variant of random descent: We form a candidate set, initially consisting of all edges of K_0. We randomly select an edge from the candidate set and try the four elementary transformations in turn until we find a legal move $(K, L) \Rightarrow (K', L')$ with $E(K', L') < E(K, L)$. If none is found, we remove the edge from

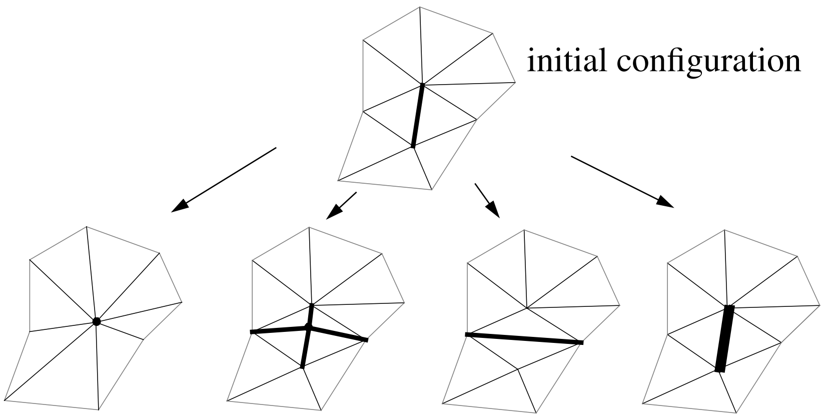


Figure 4.13: The four elementary mesh transformations defined in phase 3.

the candidate set; otherwise, we accept the move and expand the candidate set to include edges whose vertices were affected by the transformation. The process is repeated until the candidate set is empty.

Due to the expense of computing $E(K', L')$ for each speculative move, the idealized algorithm just described is too inefficient to be of practical use. We therefore replace the exact computation of $E(K', L')$ by an approximate one.

4.5.2.1 Approximate evaluation of $E(K', L')$

Our approximate computation of $E(K', L')$ is similar to that used in phase 2 (Section 3.3.4). It is based on the observation that applying a local change to the control mesh (K, L, V) leaves the optimal positions of distant control vertices essentially unchanged. Thus, when speculating upon an elementary transformation, we only optimize over the positions of control vertices in a neighborhood of the affected edge, and recompute projections of data points originally projecting onto the neighborhood of $\tilde{M}^r$ supported by these control vertices.

More precisely, when speculating upon an elementary transformation $T : (K, L) \to (K', L')$, we optimize over the set of local vertices V'_T indicated in Figure 4.14. Our choice of V'_T is based on experience; other choices are certainly possible.

Varying V'_T changes $\tilde{M}^r$ only in the simplicial neighborhood $N'_T = \mathrm{nbhd}^2(V'_T; K')$ (as defined in Section 1.7.2), with diminishing effect near the boundary of N'_T. Consequently,

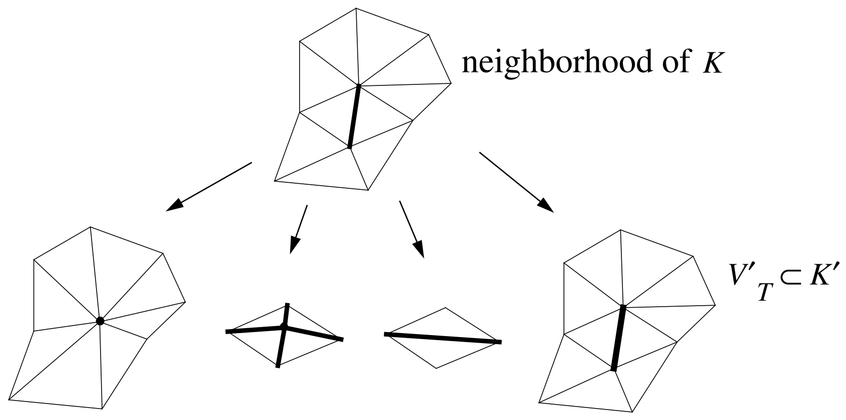


Figure 4.14: Set of control vertices $V'_T \subset K'$ over which to optimize for each elementary mesh transformation T.

to evaluate the change to E_{dist} when varying V'_T, we need only consider the data points originally projecting onto N_T. Let $X_T \subset X$ denote this set.

We approximate $E(K', L')$ by iterating over the two steps of (1) recomputing projections for points X_T onto N'_T, and (2) optimizing over the control vertex positions V'_T.

4.5.3 Implementation issues

As mentioned in Section 4.5.1, to project onto the subdivision surface $S(M)$, we use a piecewise linear approximation, namely the mesh $\tilde{M}^r$ obtained after r subdivisions with its vertices pushed to their limit positions.

If the tagged simplicial complex (K, L) is held fixed, each vertex $\tilde{\mathbf{v}}_j^r$ of $\tilde{M}^r$ is an affine combination $\mathbf{y}_j V$ of the vertices V of M, where the combination $\mathbf{y}_j$ is obtained by composing r applications of the subdivision masks and the position mask. Since each $\mathbf{y}_j$ is sparse, it can be precomputed and stored at the vertex $\tilde{\mathbf{v}}_j^r$ as a hash table indexed by vertices of V.

Thus, each iteration of the alternating procedure summarized in Section 4.5.1 involves projecting the data points onto $\tilde{M}^r$, computing new vertex positions V by solving a linear system, and updating the vertex positions $\tilde{\mathbf{v}}_j^r$ by re-evaluating the affine combinations $\mathbf{y}_j V$.

On the other hand, the approximate evaluation of $E(K', L')$ used when speculating a mesh transformation T is more involved since, as (K, L) is modified, any precomputed affine combinations in the neighborhood of T become invalid. Moreover, those affine combinations are precisely the ones needed to carry out the local optimization over control vertices V'_T. Therefore, a new piecewise linear approximation ($\tilde{M}^{r'}$) to the subdivision surface must be computed for the neighborhood of T. Our approach is as follows:

1. Make a copy of the neighborhood $\mathrm{nbhd}^3(V_T; K)$ of M;

2. Apply the transformation T to that neighborhood;

3. Subdivide that neighborhood r times and push its vertices to their limit positions. The position of each vertex in the subdivided mesh can be represented as an affine combination of vertices in $\mathrm{nbhd}^3(V'_T; K')$. Note that many vertices in these affine combinations are constant, since we will only vary vertices V'_T. We can therefore apply constant-folding to simplify the combinations.

4. Trim the resulting subdivided mesh to $\mathrm{nbhd}^2(V'_T; K')$. (In Step 1 we extracted the larger neighborhood to correctly compute the boundary of $\mathrm{nbhd}^2(V'_T; K')$.)

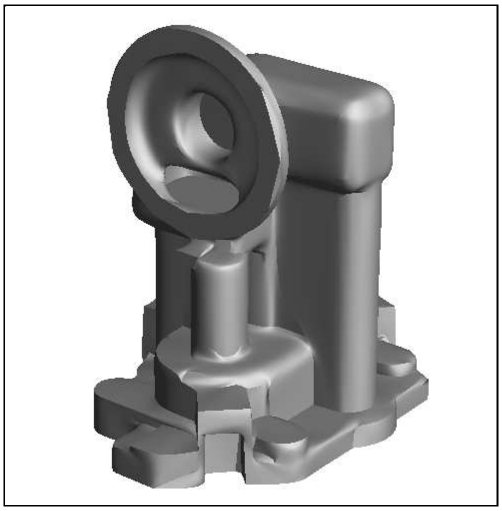
4.6 Results

From the point sets in Figure 2.11 and the phase 2 meshes shown in Figure 3.10, phase 3 produces the surfaces shown in Figure 4.15. The left columns show the optimized tagged control meshes, where the sharp edges L are drawn as bold line segments; the right columns show the piecewise smooth subdivision surfaces associated with these control meshes.

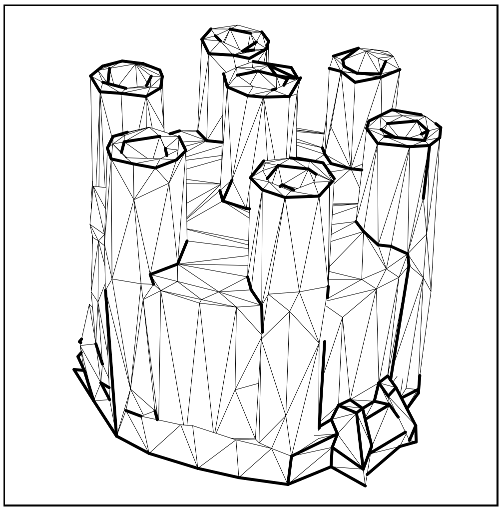
Modeling surfaces such as the one shown in Figure 4.15b using NURBS would be cumbersome and would likely require significant user intervention. In contrast, our subdivision surface approach is both simple and automatic. Also note how the teapot (Figure 4.15j) is modeled as a single subdivision surface of genus 1 (the handle of the teapot makes it homeomorphic to a torus), without resort to explicit continuity constraints or trimming curves. We further develop this comparison in Section 4.7.

Another advantage of optimization using a piecewise smooth model is that the resulting surface not only fits the data more accurately than a piecewise linear model, it is also a better predictor of the true underlying surface. As a validation test, we sampled a different set of 10,000 points from the swept surface (knot) used to generate Figure 4.15h. As shown in Table 4.2, even though the subdivision control mesh (Figure 4.15g) has a fifth as many vertices as the mesh from phase 2 (Figure 3.10h), the subdivision surface fits the new set of points with about one fourth the distance energy.

(a) Phase 3 tagged control mesh

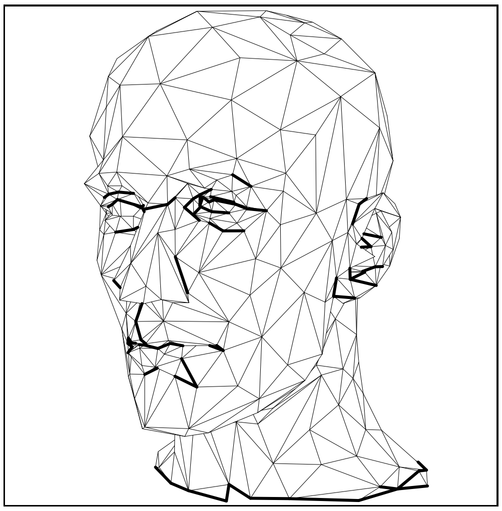


(b) Phase 3 subdivision surface

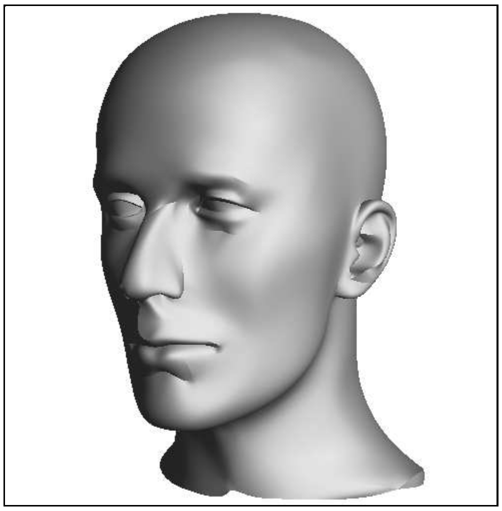


(c) Phase 3 tagged control mesh

(d) Phase 3 subdivision surface

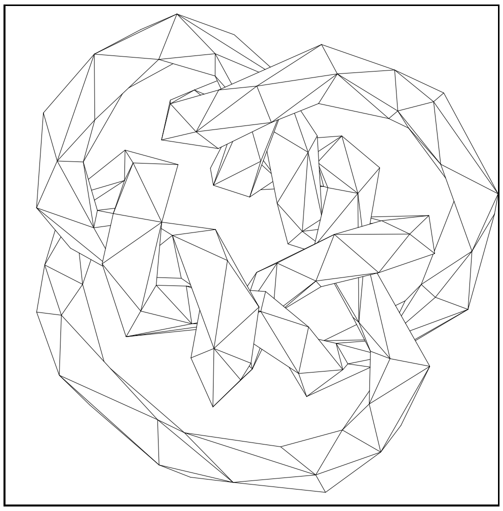


(e) Phase 3 tagged control mesh

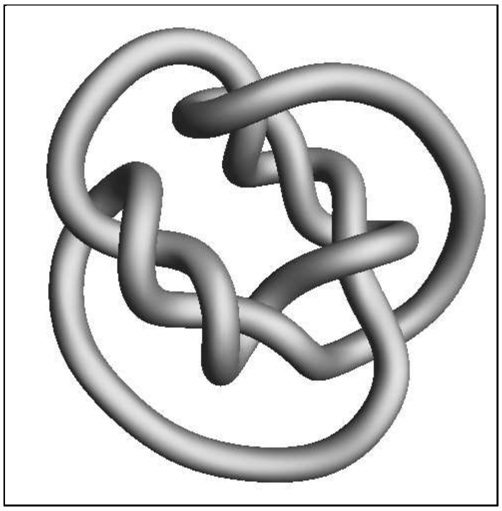


(f) Phase 3 subdivision surface

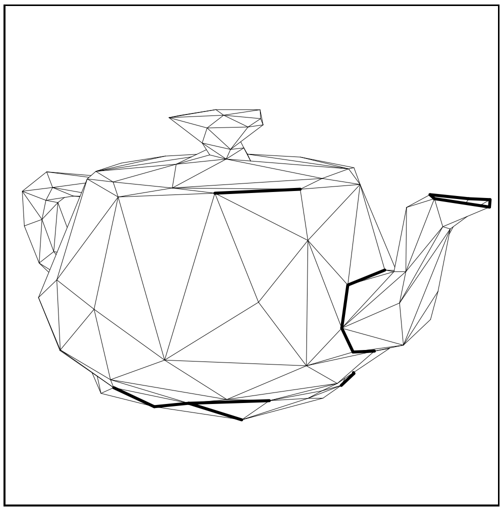
Figure 4.15: Results of phase 3 (subdivision surface optimization), using the point sets in Figure 2.11 and the phase 2 meshes in Figure 3.10.



(g) Phase 3 tagged control mesh

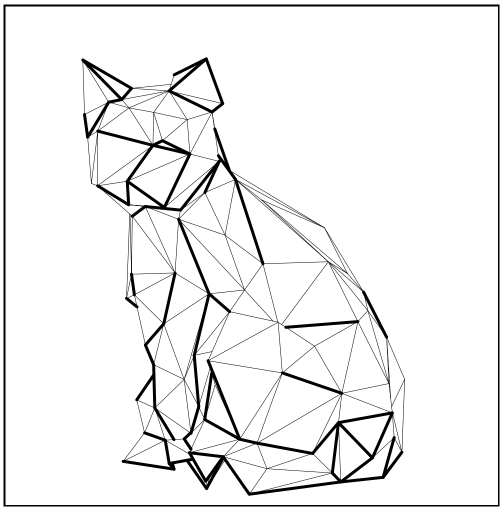


(h) Phase 3 subdivision surface

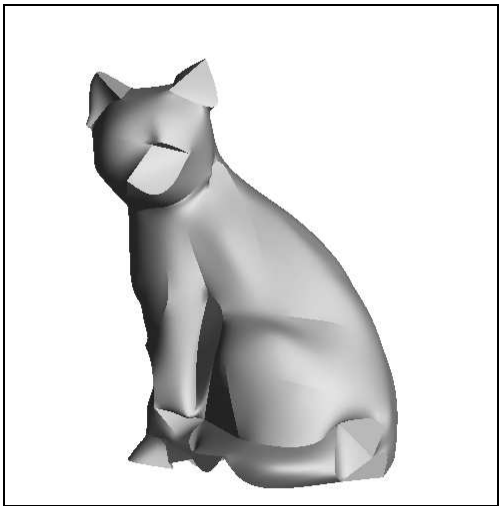


(i) Phase 3 tagged control mesh

(j) Phase 3 subdivision surface



(k) Phase 3 tagged control mesh



(l) Phase 3 subdivision surface

Figure 4.15: (continued)

Table 4.2: Validation results for phase 3.

	c_{rep}	m # vertices	E_{dist} original points	E_{dist} new points
phase 2	10^{-5}	975	.00308	.00934
phase 3	10^{-5}	363	.00042	.00054
	10^{-4}	205	.00232	.00264

Table 4.3: Phase 3 parameter settings and optimization results.

Fig.	Object	n	c_{rep}		m (#vertices)			E_{dist}			Time
			ph2	ph3	ph2	ph3	$\frac{ph2}{ph3}$	phase 2	phase 3	$\frac{ph2}{ph3}$	hrs
4.3a	mechpart	4,102	10^{-5}	10^{-5}	163	112	1.5	$4.86\ 10^{-4}$	$1.53\ 10^{-4}$	3.2	1.3
4.15ab	oilpmp	30,937	10^{-5}	10^{-5}	891	656	1.4	$4.93\ 10^{-3}$	$4.14\ 10^{-3}$	1.2	10.5
4.15cd	distcap	12,745	10^{-5}	10^{-5}	685	507	1.4	$4.05\ 10^{-3}$	$3.85\ 10^{-3}$	1.1	5.2
4.15ef	mannequin	12,772	10^{-5}	10^{-5}	689	430	1.6	$3.39\ 10^{-3}$	$1.64\ 10^{-3}$	2.1	5.6
4.15gh	knot	10,000	10^{-5}	10^{-4}	975	205	4.8	$3.08\ 10^{-3}$	$2.32\ 10^{-3}$	1.3	2.7
4.15ij	teapot	26,103	10^{-5}	10^{-4}	623	152	4.1	$3.17\ 10^{-3}$	$2.62\ 10^{-3}$	1.2	7.4
4.15kl	cat	1,000	10^{-5}	10^{-5}	181	153	1.2	$1.09\ 10^{-3}$	$8.17\ 10^{-4}$	1.3	0.8

Parameter settings and quantitative results In most examples, the representation constant c_{rep} was set to 10^{-5}, the same value that was used in phase 2. As indicated in Table 4.3, the control meshes obtained from phase 3 are more concise than those of phase 2, and at the same time, the subdivision surfaces fit the points more accurately than the meshes of phase 2. Because the point sets used for Figures 4.15gh and 4.15ij are sampled without error from piecewise smooth surfaces, we could afford to increase c_{rep} to produce very concise control meshes, while still reducing E_{dist}.

The phase 3 execution times were obtained on an SGI Indigo workstation. In all test cases we set $c_{sharp} = c_{rep}/5$ and the number of subdivision iterations (referred to in Section 4.5.1) $r = 2$.

4.7 Discussion

Comparison with NURBS fitting For the application of surface reconstruction, the phase 3 optimization method offers several advantages over traditional NURBS fitting techniques. These advantages are summarized in Table 4.4. Although one could imagine using a NURBS representation in a general optimization setting like ours, several difficulties would arise, as discussed below.

Table 4.4: Comparison of traditional NURBS fitting methods with the phase 3 approach.

	traditional NURBS fitting	piecewise smooth subdiv. surface optimization
network	set of 4-sided patches	one connected mesh
trimming curves	required	none
continuity	C^0 and G^1 constraints	continuity built-in
optimization	constrained, high-dim.	unconstrained, low-dim.
degrees of freedom	fixed patch network	local adaptation
sharp features	fixed manually	inferred automatically

As an example, let us consider how a NURBS fitting approach would reconstruct the piecewise smooth surface shown in Figure 4.3b. To fit models with sharp features such as the creases and corners of that surface, current parametric patch fitting schemes require the user to first specify a network of 4-sided faces, each corresponding to a parametric tensor product Bézier or NURBS surface patch. Often, to accommodate the intricate geometry of creases and boundaries, trimming curves are introduced that selectively remove regions of the patches. Figure 4.16 shows an example of such a patch network, where the different patches have been pulled apart for clarity. Note how trimming curves would have to be used to introduce holes in the faces of the cube.

Because the surface is defined as a network of patches, continuity constraints must be introduced to make the pieces fit together continuously (C^0) and often smoothly (G^1). Although general G^1 smoothness constraints are non-linear, sufficient conditions can often be expressed as linear constraints. Nevertheless, a NURBS fitting scheme in general requires a constrained optimization method. The presence of trimming curves further complicates

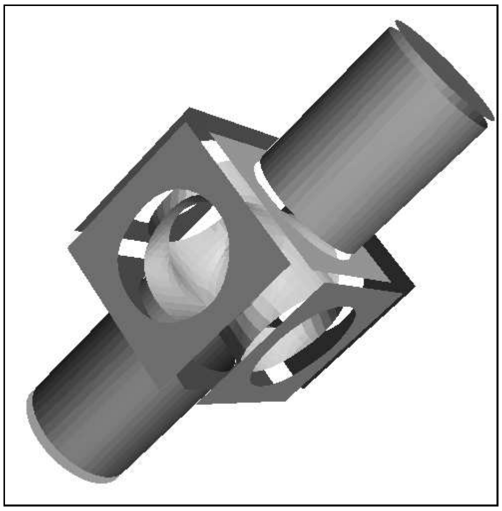


Figure 4.16: Partial segmentation of a surface into smooth patches for NURBS fitting.

the problem, since these curves are typically obtained as approximations to surface-surface intersection solutions, and are therefore difficult to incorporate in a general optimization. In current methods, the patch network is specified manually, thereby fixing the locations of sharp features and restricting the distribution of degrees of freedom. While it may be possible to develop a scheme for optimizing over the connectivity of a quadrilateral patch network, this seems to be more difficult than with meshes. One difficulty is that concave quadrilateral faces can lead to folds in the surface.

In contrast, let us see how the same surface is fit with our piecewise smooth subdivision surface representation. The surface is now associated with a single connected mesh. Since the subdivision surface scheme can represent surface boundaries and creases intrinsically, the surface need not be partitioned and trimming curves are unnecessary. Since the subdivision surface is continuous everywhere and tangent plane smooth everywhere except at intended sharp features, continuity constraints are not required. Thus, as demonstrated in Section 4.5, fitting subdivision surfaces leads to an unconstrained, low-dimensional optimization. The simplicity of the representation allowed the development of an optimization algorithm able to fine tune the distribution of degrees of freedom (control mesh vertices) in order to locally adapt to surface shape. Finally, as the set of sharp edges in the control mesh is easily made a variable of the optimization, the procedure is able to automatically infer the presence of sharp features in the surface.

Initial set of sharp edges Subdivision surface optimization requires an initial set L_0 of sharp edges. In all the examples we let L_0 consist of all edges in the phase 2 mesh whose dihedral angles exceed the threshold value of 40 degrees. Two other obvious alternatives are to let L_0 be empty or to let it contain all edges of K.

In practice letting L_0 be empty starts the optimization too far from the global minimum. The initial global optimization over V for (K_0, L_0) disrupts the control mesh as in Figure 4.9a, and the optimization algorithm cannot recover.

We also tried the alternative of letting L_0 contain all edges of K_0. This approach is elegant since the initial subdivision surface has the same geometry as the phase 2 surface.[4] However, this approach seems to unnecessarily slow down the optimization, and occasionally leads to poor energy minima.

More accurate approximation to the subdivision surface To project a point onto the subdivision surface, instead of using a global piecewise linear approximation ($\tilde{M}^r$), we could obtain a more precise projection using recursive root-finding. In such a scheme, we would first perform iterations of local subdivision. As the mesh is subdivided, we need only keep the neighborhood defining the region of the subdivision surface onto which the point projects. (The convex hull property of subdivision surfaces could be exploited here.) Local subdivision is equivalent to midpoint bisection search, and has linear convergence. After a few subdivision steps, the local neighborhood usually becomes regular, with all vertices being smooth and of valence 6. Since the subdivision surface defined by a regular neighborhood has a closed form expression (it is a quartic box-spline), numerical root-finding (e.g. Newton-Raphson) can then be used to converge superlinearly to within machine precision.

Wider variety of sharp surface features The three types of features we define (corners, creases, and darts) are derived from the simple strategy of tagging mesh edges. New subdivision rules could be developed to model a wider variety of features (e.g. cones, multiple darts meeting at a smooth vertex, darts meeting at a corner, and corners along boundaries). An obvious approach is to tag not only edges, but also vertices and faces of the mesh.

---

[4] This is true except at boundary vertices of valence 2, which are crease vertices under the current subdivision rules and are therefore not interpolated by the subdivision surface.

Chapter 5

SURFACE APPROXIMATION

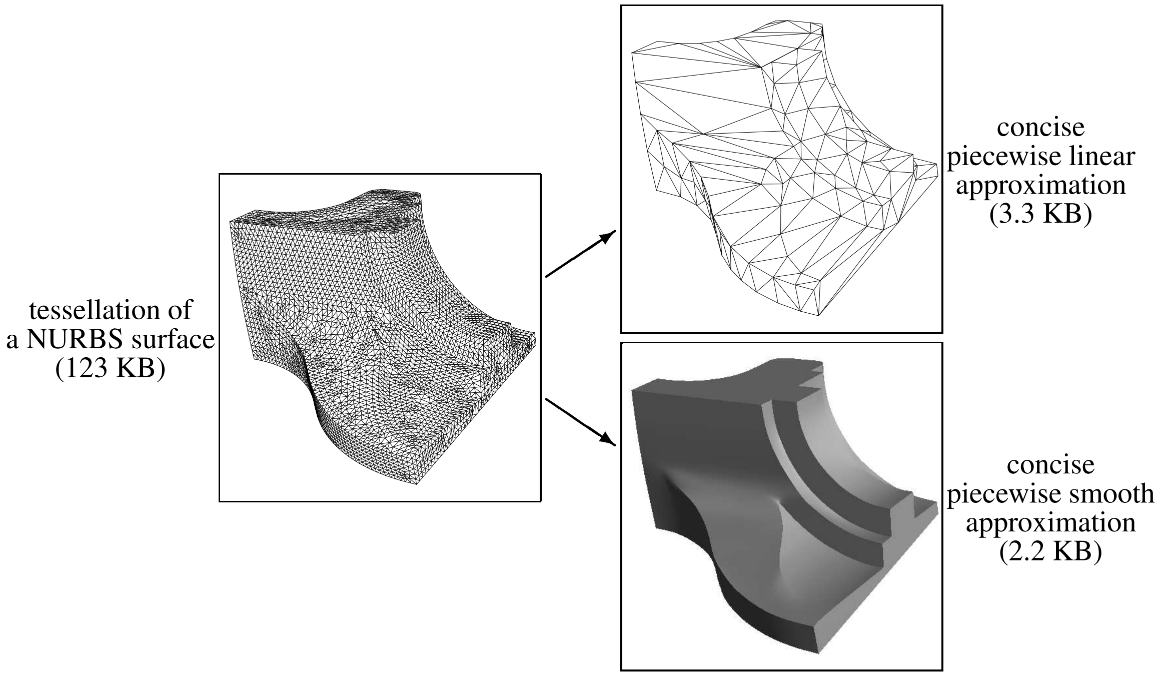


Figure 5.1: Approximation of a NURBS surface by concise piecewise linear and piecewise smooth surfaces. The surface describes a gas turbine engine component (tessellation courtesy of Pratt & Whitney). Size of representation is expressed in kilobytes.

5.1 Introduction

For practical reasons, complex surface representations must often be approximated using low-order elements. For instance, displaying a smooth surface on a graphics workstation usually involves approximating the surface by planar elements that can be fed into a graphics pipeline. Similarly, performing finite element analysis on a surface typically requires approximating the surface by polynomial elements of low degree.

A piecewise linear approximation to a parametric surface can easily be obtained by triangulating its parametric domain and evaluating the surface at these vertices, a process called *tessellation*. Similarly, the triangulation of an implicit surface is typically generated using a contour tracing algorithm in which surface points are computed exactly by root-finding.

However, common methods for tessellation and contour tracing naturally produce dense representations. Dense models are also the natural result of many other geometric modeling problems, such as approximation of offset surfaces, constructive solid geometry (CSG) operations, and surface-surface intersection. Although dense representations may be easiest to generate, they are of course inconvenient, since they require more space to store and take longer to transmit, render, and analyze. Thus there has been extensive work in various application areas to attempt to reduce the size of the output in regions where it is unnecessary. For instance, many methods adapt the density of the representation using heuristic estimates of curvature [6, 56, 71]. However, in general these heuristic methods are tedious to implement and produce results that are far from optimal.

Another approach to finding a concise surface approximation is to first generate a dense approximation, and then simplify it. One advantage of this approach is that the same algorithm can be used for all the problem areas mentioned above. The optimization algorithms from phases 2 and 3 of the surface reconstruction procedure can be used for this purpose, as demonstrated in Figure 5.1.

We first show that mesh optimization (phase 2) can be used for mesh simplification—finding piecewise linear approximations to piecewise linear surfaces (Section 5.2). More generally, mesh optimization can be used to find piecewise linear approximations to arbitrary surfaces (Section 5.3). Finally we show that subdivision surface optimization (phase 3) can be used to further reduce representation size by finding piecewise smooth approximations to arbitrary surfaces (Section 5.4).

5.2 Mesh simplification

Mesh simplification refers to the problem of reducing the number of faces in a dense mesh while minimally perturbing the shape. The problem can also be stated as that of finding a concise piecewise linear approximation to a piecewise linear surface. As an example, from the dense mesh in Figure 5.1, we want to obtain the much coarser one shown on the upper right.

5.2.1 Previous work

Some notable papers discussing the mesh simplification problem are Schroeder *et al.* [60], Turk [71], Rossignac and Borrel [53], and Lounsbery *et al.* [35].

The motivation of Schroeder *et al.* [60] is to simplify meshes generated by "marching cubes" that may consist of millions of triangles. In their iterative approach, the basic operation is removal of a vertex and re-triangulation of the hole thus created. The criterion for vertex removal in the simplest case (interior vertex not on edge or corner) is the distance from the vertex to the plane approximating its surrounding vertices. It is worthwhile noting that this criterion only considers deviation of the new mesh from the mesh created in the previous iteration; deviation from the original mesh does not figure in the strategy.

The goal of Turk [71] is to reduce the amount of detail in a mesh while remaining faithful to the original topology and geometry. His basic strategy is to distribute points on the existing mesh that are to become the new vertices, create a "mutual" triangulation containing both old and new vertices, and finally remove the old vertices. The density of the new vertices is chosen to be higher in areas of high curvature.

Rossignac and Borrel [53] describe a simple and efficient simplification method that generalizes to arbitrary simplicial complexes. Their approach is to partition space into cubical bins, enter all vertices of the model in these bins, unify vertices within each bin into a single representative vertex, and finally, appropriately collapse affected simplices. A unique aspect of their approach is that the topological type of the model may change in the process; for instance, a cylinder of small radius may be reduced to a simple line segment (1-simplex) in the simplified model. Their technique is effective at removing small detail in representations with a wide range of scale. In a detailed representation of a car, for instance, screws, rivets, and other small items would likely be removed altogether. However, the method is less successful on models of uniform scale, as it ignores geometric qualities like curvature. As a simple example, it would keep most vertices within a planar triangulation.

Lounsbery *et al.* [35] extend the notion of multiresolution analysis to surfaces of arbitrary topological type. They present a hierarchical representation for subdivision surfaces (and in particular, meshes), using locally supported basis functions that have many of the qualities of wavelets. Their method allows the fast simplification of meshes, but cannot be applied in its current form to the general problem of mesh simplification, as it requires the original mesh to have subdivision connectivity—the original mesh must be the result of uniform subdivision of a simple base mesh.

5.2.2 Mesh simplification using mesh optimization

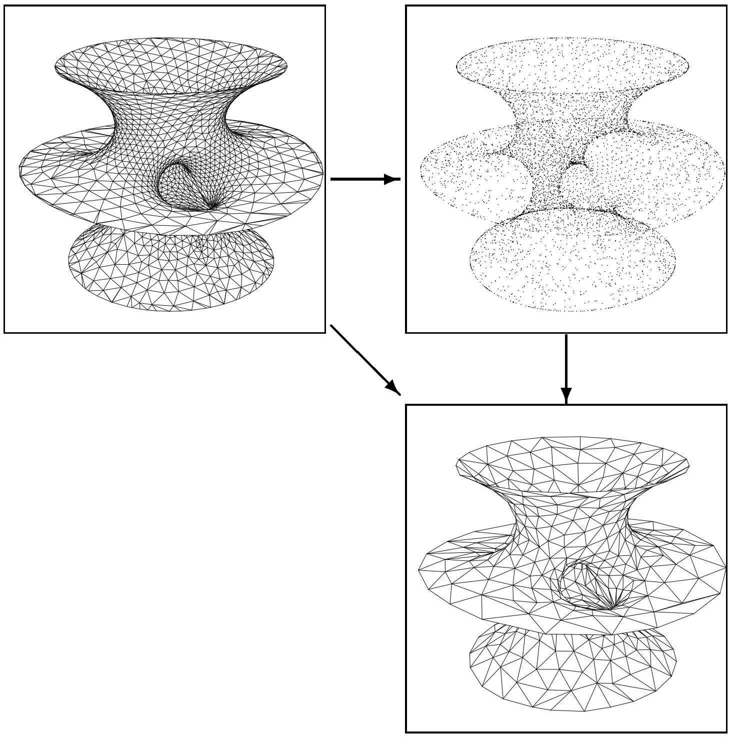


Figure 5.2: Example of mesh simplification using mesh optimization.

Our mesh simplification approach applies the mesh optimization algorithm (described in Section 3.3) as follows: Sample data points X from the initial mesh and use the initial mesh as the starting point M_0 for mesh optimization. To obtain X, we first sample a set of points randomly from the original mesh using uniform random sampling over area; that is, the average number of sample points in a triangle is proportional to the area of the triangle. Next, we add the vertices of the mesh to this point set. Finally, to more faithfully preserve the boundaries of the mesh, we sample additional points from boundary edges. Note that there is no need to run phase 1, since the initial mesh is in fact the surface U to be approximated. The distance energy E_{dist} involving the points X is in effect a metric that measures deviation of the final mesh from the original.

As an example (Figure 5.2), from a dense mesh of 2,032 vertices[1] and a set of 6,752 points sampled from this mesh, mesh optimization obtains a coarser mesh of 487 vertices.

[1] The mesh is an approximation to a minimal surface (courtesy of Celso Costa, David Hoffman, William Meeks III, and James T. Hoffman).

Although the size of the representation has been greatly reduced, the mesh is still a good geometric approximation due to the judicious placement of its vertices.

The principal advantage of our mesh simplification method compared to previous techniques is that we cast mesh simplification as an optimization problem: we find a new mesh of lower complexity that is as close as possible to the original mesh. This is recognized as a desirable property by Turk (Section 8, p. 63): "Another topic is finding measures of how closely matched a given re-tiling is to the original model. Can such a quality measure be used to guide the re-tiling process?". Optimization automatically retains more vertices in areas of high curvature, and leads to faces that are elongated along directions of low curvature, another property recognized as desirable by Turk.

5.2.3 Data dependent triangulations

In a related problem called *data dependent triangulation*, the goal is to fit meshes to data defined as a function over the plane ($z = f(x, y)$) (see Dyn and Rippa [16] for a review). This problem can be viewed as a specialized instance of mesh simplification, in which the meshes are restricted to project one-to-one onto the xy plane.

A common instance of the data dependent triangulation problem involves the modeling of elevation data. Digital elevation data typically takes the form of a dense rectangular grid of height values. Such data can be visualized as a surface by defining a dense mesh over the set of 3D data points. To obtain a more concise mesh, some simplification methods [10, 15, 63] select as vertices only a subset of the points and obtain an accurate surface by optimizing over the connectivity of these vertices. More recent work allows for movement of the mesh vertices and simplification of the mesh based on computed estimates of curvature [56].

Since this problem is an instance of the mesh simplification problem, we can apply mesh optimization. As an example, from a terrain mesh of 1200×1200 vertices[2], we obtain the much coarser mesh (6,848 vertices) shown in Figure 5.3. This represents a factor of 200 in compression. As a by-product of our energy minimization approach, the mesh vertices have clustered near features of the terrain, so that viewed from afar, the two surfaces appear similar (Figure 5.4).

---

[2] The mesh vertices correspond to height values on a grid of one degree in latitude by one degree in longitude in the Mojave desert.

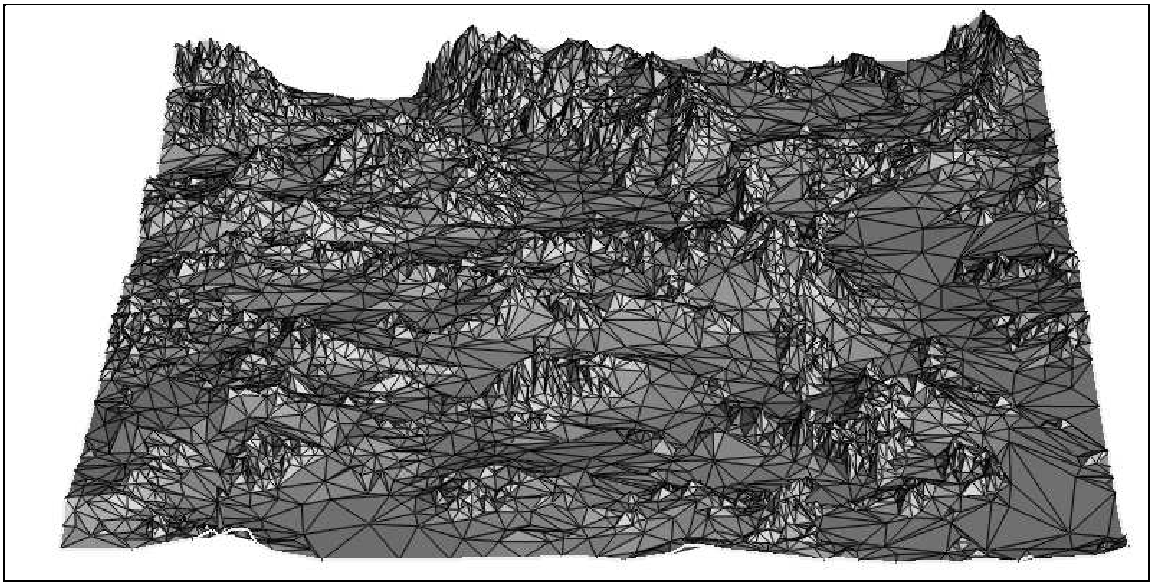
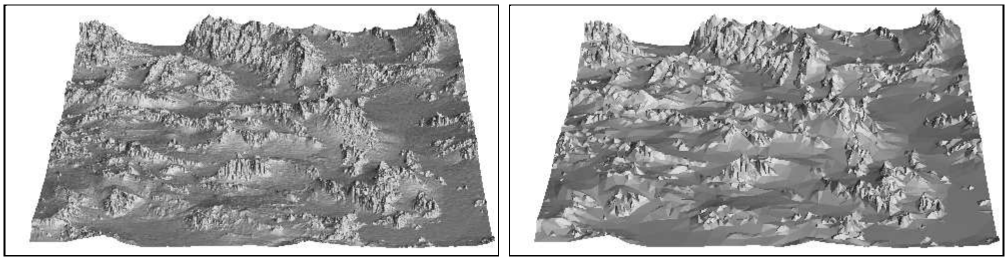


Figure 5.3: Result of mesh optimization on a dense grid of elevation data. Altitude is exaggerated by a factor of 20.

2.6in2.95in



(a) Original mesh of 1,440,000 vertices (b) Optimized mesh of 6,848 vertices

Figure 5.4: Comparison of shaded original and optimized meshes.

An important issue in the use of mesh optimization for function reconstruction is the definition of the distance metric. In mesh optimization, we measure distance of a point $\mathbf{x}_i = (x_i, y_i, z_i)$ to the surface $S = \{(x, y, f(x, y))\}$ as Euclidean distance to the closest point on the surface, or $d(\mathbf{x}_i, S)$. On the other hand, in function reconstruction, distance is measured by projecting onto the surface along the z axis, or $(z_i - f(x_i, y_i))$. The implications of this subtle difference on the optimization problem and algorithm require further investigation.

5.3 Piecewise linear approximation

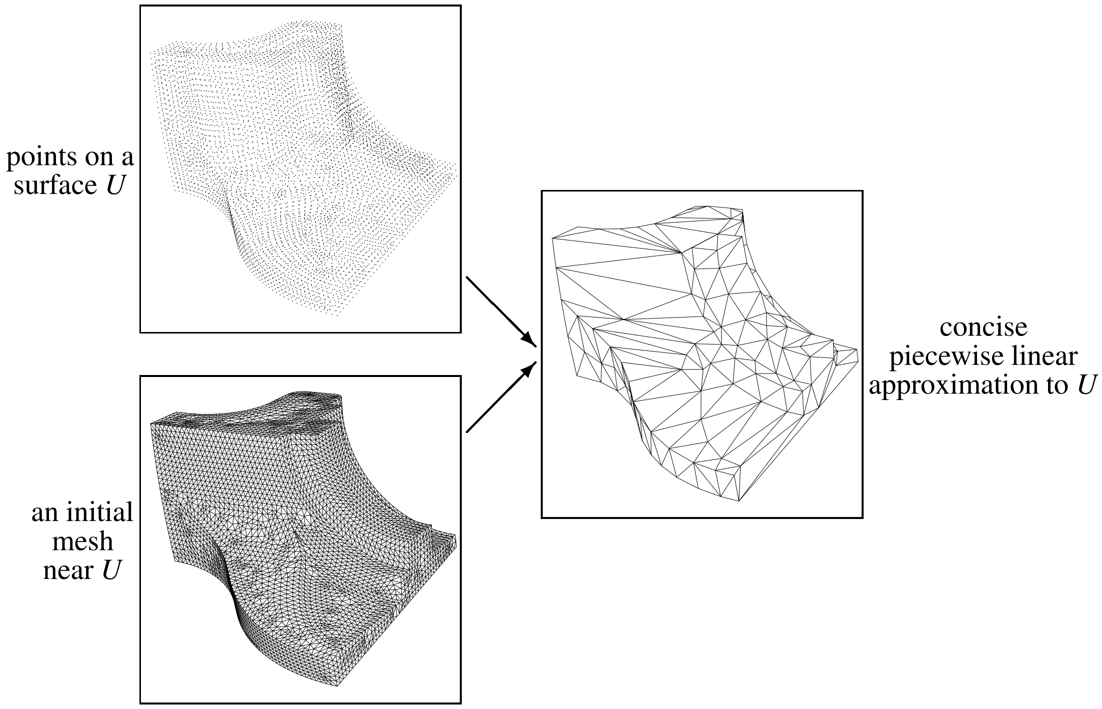


Figure 5.5: Piecewise linear approximation.

In the previous section we described how mesh optimization can be used to find a concise piecewise linear approximation to a dense mesh. More generally, mesh optimization can be used to find a concise piecewise linear approximation to an arbitrary surface U. For this purpose, we require two inputs, as shown in Figure 5.5:

1. a dense set of points that lie exactly on U,

2. an initial mesh approximating U.

Another way to supply these two inputs is to simply provide a dense tessellation of U whose vertices lie exactly on U. Such a tessellation is easily obtained from a parametric surface by evaluating the surface at grid points in the parametric domain, or from an implicit surface by executing a contour tracing algorithm in which isosurface vertices are computed exactly (using root-finding). Note that the surface need not have a closed form formula; for instance, we can find tessellations of offset surfaces, CSG surfaces, etc.

As an example, Figure 5.5 shows the piecewise linear approximation of a NURBS surface given a tessellation of 6,475 vertices. We invoke mesh optimization on the dense mesh, using its vertices as the set of points X, to obtain a concise mesh of 161 vertices.

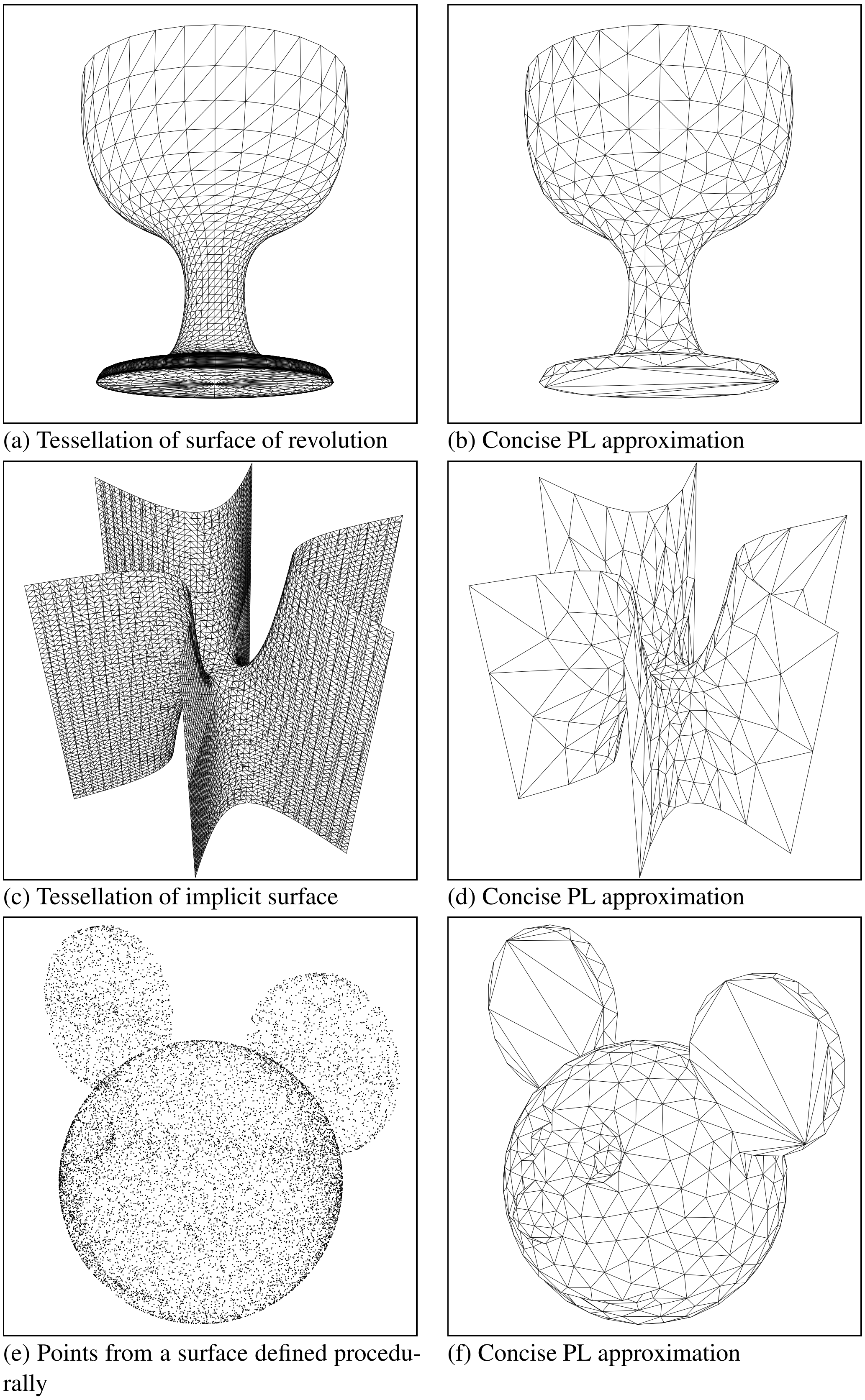
We have experimented with CSG surfaces (Figure 3.1), NURBS surfaces (Figure 3.10j), swept surfaces (Figure 3.10h), surfaces of revolution (Figure 5.6b), implicit surfaces (Figure 5.6d), and procedurally-defined models (Figure 5.6f). As these examples illustrate, using an optimization with a measure of distance between the surface U and the approximation has a number of benefits:

- Vertices are dense in regions of high Gaussian curvature, whereas a few large faces span the flat regions.

- Long edges are aligned in directions of low curvature, and the aspect ratios of the triangles adjust to local curvature.

- Edges and vertices of the simplified mesh are placed near sharp features of the original mesh.

It should be emphasized that no heuristics are necessary to obtain these benefits; they are simply by-products of the energy minimization process.



(a) Tessellation of surface of revolution
(b) Concise PL approximation
(c) Tessellation of implicit surface
(d) Concise PL approximation
(e) Points from a surface defined procedurally
(f) Concise PL approximation

Figure 5.6: Results of piecewise linear approximation. Point sets X are the vertices of the tessellations. (Mickey courtesy of Steve Mann.)

5.4 Piecewise smooth approximation

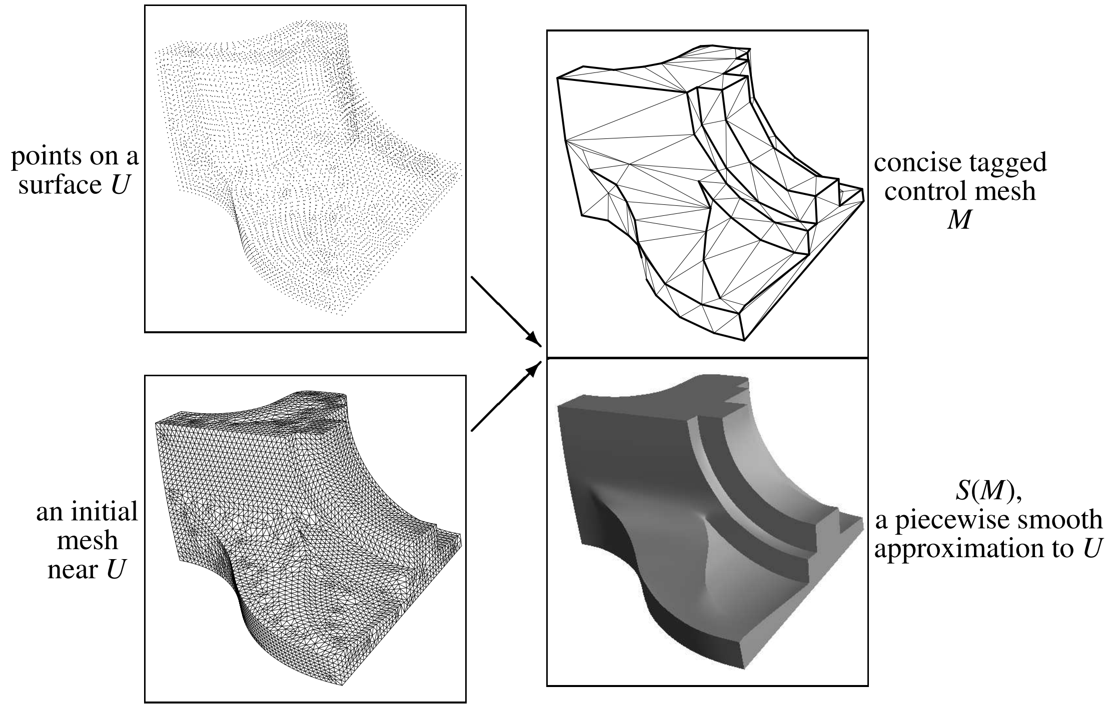


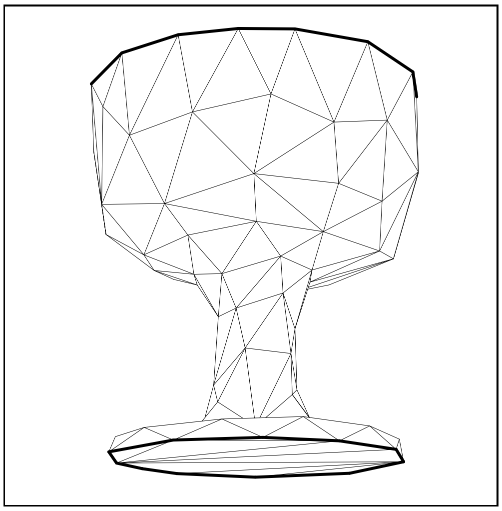
Figure 5.7: Piecewise smooth approximation.

Phase 3 (subdivision surface optimization) can be used to find a concise piecewise smooth approximation to an arbitrary surface U. As in the previous section, we begin with a set of points on U and an initial mesh. We then run phases 2 and 3 of the surface reconstruction procedure.

As an example, Figure 5.7 shows the approximation of a NURBS surface, given a dense tessellation. The output is a piecewise smooth subdivision surface, whose control mesh is more sparse than the PL approximation from Figure 5.5 (now 108 vertices instead of 161). Also note that the optimization algorithm automatically inferred the sharp features in the surface.

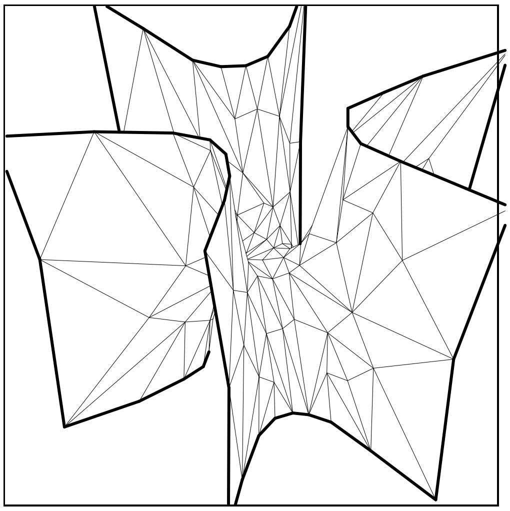
We have experimented with CSG surfaces (Figure 4.3), NURBS surfaces (Figure 4.15j), swept surfaces (Figure 4.15h), surfaces of revolution (Figure 5.8b), implicit surfaces (Figure 5.8d), and procedurally-defined models (Figure 5.8f).

In the surface of Figure 5.8d, the corners of the original surface are smoothed out in the reconstruction. The reason is that the current set of subdivision rules in our piecewise smooth scheme (Section 4.3.1) allow a sharp corner on a boundary only if an incident

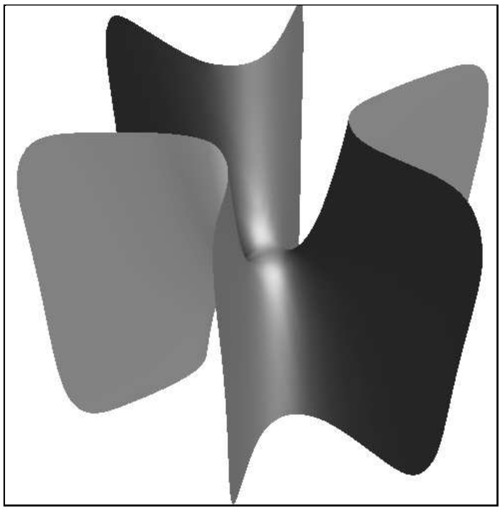


(a) Concise tagged control mesh

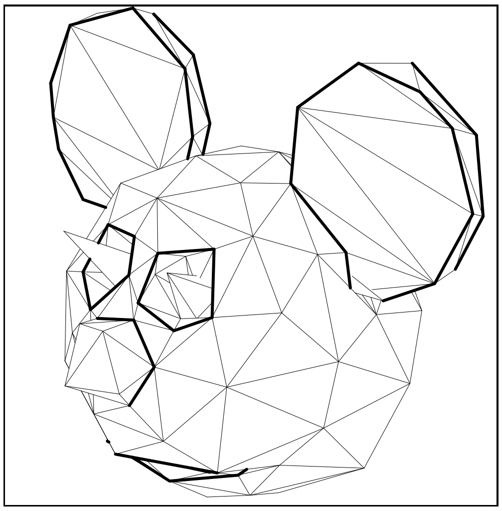
(b) Subdivision surface approximation



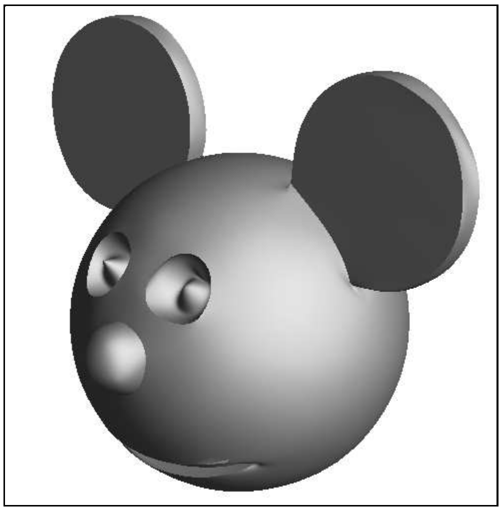
(c) Concise tagged control mesh



(d) Subdivision surface approximation



(e) Concise tagged control mesh



(f) Subdivision surface approximation

Figure 5.8: Results of piecewise smooth approximation. The inputs were the dense tessellations shown in Figure 5.6.

interior edge is tagged as sharp. A possible remedy is to modify the subdivision rules to let boundary vertices of valence 2 be corner vertices. We have not explored that modification.

5.5 Discussion

The problem of surface approximation belongs to the larger field of approximation theory. However, most of the approximation work to date has addressed function approximation, and can therefore only deal with surfaces defined as functions over simple domains (often 1D domains). In our context, little is known about approximation of manifolds of arbitrary topological type. Our approach does not advance theoretical matters, as we do not make any statements concerning asymptotic convergence rates of our surface approximations. However, we do offer a practical solution to a rather difficult problem.

Obtaining a sample of points When sampling points on a surface for the purpose of surface approximation, the parameterizations of the points (their barycentric coordinates) should be recorded along with their positions. Doing so avoids the need to initially project the points. More importantly, it allows the simplification of models that have co-incident surfaces. When several components of a surface share a surface region, points sampled from that region might, if projected, be associated with only one of the components, leaving the other components with no sample points there.

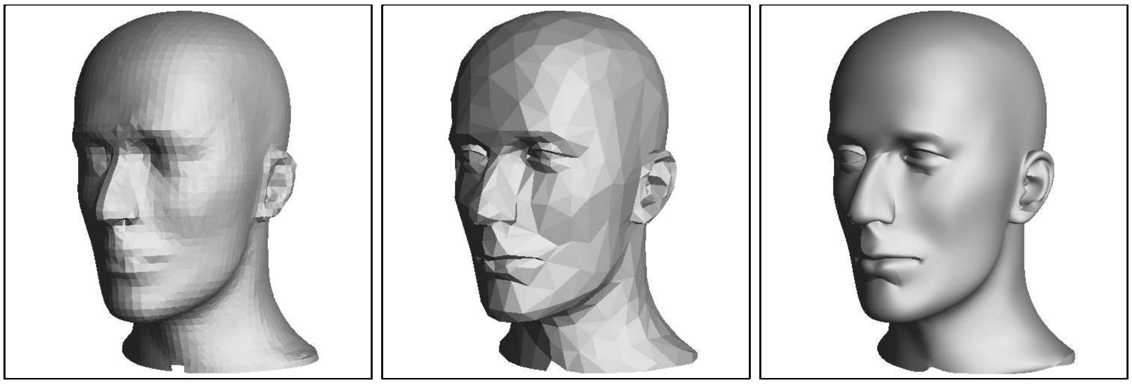
Mesh generation for analysis Our approximating meshes usually have far fewer triangles than the original ones, making them attractive for analysis (e.g. finite element analysis). However, in such analyses, the singularity of the solution system is strongly influenced by the geometric aspect ratios of the elements. It would therefore be advantageous to discourage the presence of long skinny triangles, which make the system ill-conditioned, by adding an additional penalty term to the energy function.

Simplification of complex models The surface approximation techniques developed in this chapter are effective at simplifying models at a given scale. However, dealing with more complex models involves many issues beyond the scope of this work. For instance, an interactive walk-through of a Boeing 777 CAD model involves millions of parts on a wide range of scales (e.g. from a fuselage down to individual bolts). To reduce the complexity, it is essential to organize these parts into hierarchical structures that can be displayed at different levels of detail. Surface approximations for parts and groups of parts should be allowed to change topologically at various levels (as in the method of Rossignac and Borrel [53]). Another approach, or one to use in conjunction with the above, is to encode geometric detail of distant surfaces using texture maps.

Chapter 6

SUMMARY AND FUTURE WORK

We have described a surface reconstruction procedure consisting of three major phases (Figure 6.1). The goal of phase 1 is to robustly determine the topological type of the surface (including the presence of boundaries), and to find an approximation of its geometry, in the form of a mesh. In phase 2 we defined an energy function embodying the two goals of accuracy and conciseness, and described an optimization algorithm for finding a new mesh of the same topological type minimizing this energy. Finally in phase 3 we introduced a new surface representation (piecewise smooth subdivision surfaces) allowing the convenient modeling of sharp surface features, and we showed how such surfaces could be optimized using an extension of the phase 2 algorithm. The surface reconstruction procedure was demonstrated on a number of examples including real laser range data. In addition, we also showed how the phase 2 and 3 optimization algorithms could be used effectively for surface approximation.



(a) Phase 1 surface (b) Phase 2 surface (c) Phase 3 surface

Figure 6.1: Example summarizing the 3 phases of surface reconstruction.

6.1 Analysis of the reconstruction method

As shown in Table 6.1, the accuracy and conciseness of the surface model was successively improved in each phase. It is interesting to note that the shift to smooth surfaces in phase 3 did not provide as drastic an improvement for noisy sampled data as it did for exact data. However, these numbers do not reflect the fact that the piecewise smooth surfaces are likely to be much better predictors of the underlying surfaces than the piecewise linear approximations. In other words, the piecewise linear approximations may happen to accurately fit the finite set of sample points, but yet be inaccurate representations of the underlying surfaces. As an example, Table 6.2 shows the distance energies of the approximations of the "knot" surface as measured using a different set of sample points.

Table 6.1: Comparison of accuracy and conciseness of the surfaces after each phase. Accuracy is measured by the residual sum of squares E_{dist}; conciseness is measured by the number of kilobytes required to store the representation in compressed form.

	measured data				exact points	
	oilpmp		distcap		knot	
	E_{dist}	size (KB)	E_{dist}	size (KB)	E_{dist}	size (KB)
point set	-	212	-	131	-	112
phase 1	0.05830	373	0.13000	153	0.15200	67
phase 2	0.00493	20	0.00405	15	0.00308	21
phase 3	0.00414	14	0.00385	10	0.00232	4

Table 6.2: Validation results: E_{dist} to another point set sampled on U.

	original points	new points
phase 1	0.15200	0.15300
phase 2	0.00308	0.00934
phase 3	0.00232	0.00264

6.2 Specialization to curve reconstruction

The three phases of the reconstruction procedure can easily be adapted to the reconstruction of curves in $\mathbf{R}^2$. We have implemented phases 1 and 2 for curve reconstruction. Although adapting phase 3 would be equally straightforward (as will be outlined below), we have not yet done so.

As an example, Figure 6.2 shows the reconstruction of curves from two sets of 200 points in the plane.

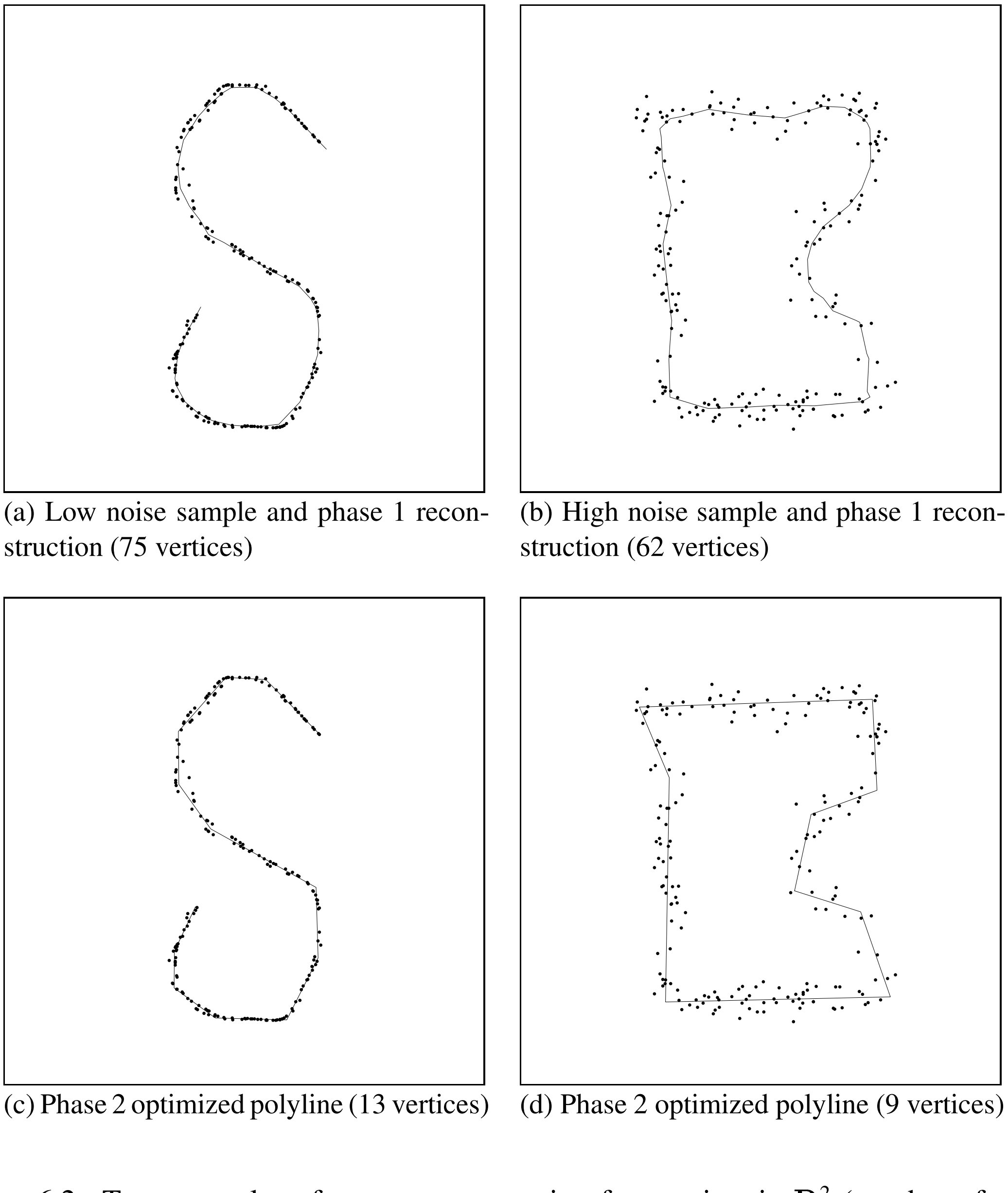
The phase 1 algorithm is essentially the same as described in Chapter 2, with only minor differences. Instead of tangent planes, tangent lines are estimated from the data points. Instead of marching over cubes in $\mathbf{R}^3$, the contour tracing algorithm marches over squares in $\mathbf{R}^2$. As the phase 1 results of Figures 6.2a and 6.2b show, the algorithm is able to handle considerable noise when the sampling is dense. Note that the curve reconstruction problem is fundamentally simpler than surface reconstruction as there are only two topologically distinguishable curves: open curves and closed curves.

Phase 2 adapts easily to the optimization of piecewise linear 1-dimensional manifolds (*polylines*) in $\mathbf{R}^d, d \geq 2$. To optimize over 1-dimensional simplicial complexes K, the algorithm considers edge collapse and edge split transformations. From the dense curves obtained above, phase 2 produces the optimized curves of Figure 6.2c and 6.2d. As Table 3.1 indicated, accuracy and conciseness are improved in both curves.

Phase 3 adapts easily to the optimization of piecewise smooth subdivision curves. Although we have not yet implemented this, we could do so as follows: We would tag vertices as either smooth or corner, and consider three transformations: edge collapse, edge split, and vertex tag. The curve would converge to a uniform cubic B-spline away from corners with the following three simple subdivision masks: a $(1, 1)$ edge mask, a $(1, 6, 1)$ smooth vertex mask, and a $(0, 1, 0)$ corner vertex mask. The limit masks would simply be: a $(1, 4, 1)$ smooth position mask, a $(0, 1, 0)$ corner position mask, a $(-1, 0, 1)$ smooth tangent mask, and $(1, -1, 0)$, $(0, -1, 1)$ corner tangent masks.

Alternatively, phase 3 could represent piecewise smooth curves using non-uniform cubic B-splines. Discontinuities (endpoints and corners) would be introduced at selected vertices by triplication of knot values.

These proposed piecewise smooth curve optimization schemes are similar to the parametric curve fitting method of Plass and Stone [49]. Their method also casts fitting as non-linear optimization and produces piecewise smooth, rather than everywhere smooth



(a) Low noise sample and phase 1 reconstruction (75 vertices)

(b) High noise sample and phase 1 reconstruction (62 vertices)

(c) Phase 2 optimized polyline (13 vertices)

(d) Phase 2 optimized polyline (9 vertices)

Figure 6.2: Two examples of curve reconstruction from points in $\mathbf{R}^2$ (number of curve vertices in parenthesis).

models. However, unlike our approach, they do not optimize globally over representation size and presence of sharp features.

6.3 Future work on surface reconstruction

There are a number of avenues for future work in extending the surface reconstruction method:

Further experimentation with non-uniform, sparse, and noisy data As discussed in Section 2.1, to obtain a proper reconstruction, several assumptions must be made concerning the sampling process that generates the data points. Several shortcomings of our method can be observed when the data is not uniform, dense, and accurate:

Non-uniform data: Although the reconstruction method can handle some non-uniformity in the point sample surprisingly well (e.g. Figures 2.11e and 4.15f), it is not designed for data sampled at varying resolutions. Phase 1 requires an estimate of the sampling density ($\rho + \delta$), and this parameter is currently user-specified and global. In phases 2 and 3, since the distance energies E_{dist} weigh all points equally, the optimizations seek to balance residual distances uniformly over all points without regard to scale. If the points are sampled from several objects (possibly of different sizes), it would therefore be beneficial to partition the points into subsets associated with each object, and to process each subset independently, in a scale-invariant fashion as is already done in phases 2 and 3.

Contour data (e.g. Figure 2.10a) is often *anisotropic*—the distance between points within a contour is typically smaller than the distance between contours. When phase 2 is applied to this type of data, many faces in the optimized meshes orient themselves to lie within the planes containing the contours, as such geometric configurations provide "excellent fit"; similar problems occur in phase 3. The underlying problem lies in the definition of our energy function, whose minimum in this case simply does not correspond to a desirable surface. Although the point set shown in Figure 2.11a also originates from contour data, its anisotropy is much less pronounced and we were able to obtain a good reconstruction.

Sparse data: When the data is sparse, that is, when there are few points relative to the amount of detail on the underlying surface, phase 3 often infers a desultory set of sharp features. For instance, from a sparse sample of 1,000 points (Figure 2.5b) sampled from a mesh of 700 faces (Figure 2.5a), phase 3 produced the relatively poor reconstruction shown in Figure 4.15l; however, significant improvements would be difficult with so little data.

Noise: In many of our examples, the data points were obtained by a physical scanning process (with finite precision), so the data points contain some amount of noise. We were fortunate in that the magnitude of this noise was relatively small. Further experiments with noisier data may reveal the need in phase 3 for a regularizing term like the phase 2 spring energy term.

Statement of correctness Phase 1 has produced surfaces of the correct topological type in all our examples. It would be desirable to develop formal guarantees on the correctness of the reconstruction, given constraints on the sample and the original surface as hinted in Section 2.1. One might consider generalizing results of sampling theory, such as formulating a Nyquist-like theorem for manifold reconstruction. However, even simple cases like the reconstruction of functions of two variables ($z = f(x, y)$)—in which the topological type of the surface is known—are not fully understood. Moreover, sampling theory results usually assume that the original signal is band-limited in frequency. Most surfaces of interest are not everywhere smooth and would therefore violate such assumptions.

Speedup of the algorithm Analysis in Section 2.4 concluded that phase 1 requires roughly $O(n \log n)$ time. Although we cannot make any precise statements concerning time complexity in phases 2 and 3, empirical evidence suggests that their execution times grow roughly as $O(n)$, consistent with the fact that the optimization problems are either local or involve sparse linear systems, and that we use spatial partitioning techniques to perform geometric searches in constant time.

However, for the sizes of data sets we considered ($n < 100,000$), execution time of the reconstruction method is dominated by phases 2 and 3; the ratios of execution times of phase 1 to phase 2 to phase 3 are roughly $1 : 50 : 300$. Significant speedups would be required for commercial applications. Implementation of the algorithm on parallel architectures should also be considered.

On-line algorithm The development of an on-line algorithm would allow for incremental reconstruction as data is acquired.

Symmetric distance metric The energy functions of phases 2 and 3 lack a measure of distance from the surface to the points—our distance energy E_{dist} only measures distance from the points to the surface. In practice this does not pose a problem, except at surface boundaries, where the reconstructed surface sometimes extends beyond the boundary indicated by the points. The regularizing spring energy term E_{spring} of phase 2 either counteracts or reinforces this deficiency, depending on whether the surface near the boundary is convex or concave, respectively.

Ideally, one would want to define an energy functional that measures distance in both directions, but we see no way to incorporate such a *symmetric distance* functional into our scheme. The main difficulty is that measuring distance from a surface to the points involves an area integral, instead of a discrete sum as in E_{dist}.

Control over maximum error The current distance energy minimizes a least squares functional, or L^2 norm. In industrial applications, specification is often done through tolerances, so it would be desirable to optimize over maximum error, or an L^∞ norm. Alternative optimization algorithms should be developed to allow direct control over maximum error.

Control over representation size Currently, the trade-off between accuracy and conciseness in phases 2 and 3 is specified by the user through the parameter c_{rep}. An advantage of this mode of specification is that, for a fixed value of c_{rep}, the representation size of resulting models adapts to the complexity of their geometry. (Almost all examples in Chapters 3 and 4 use the same value of $c_{rep} = 10^{-5}$.)

An alternative, equally useful approach would be to let the user specify the desired representation size (e.g. the number of vertices), and let the method try to find the best fitting model of that size.

Output of NURBS surfaces Since most CAD systems do not yet support subdivision surface representations, the phase 3 procedure could be extended or used as a starting point for the generation of NURBS surfaces. The automatic detection of sharp features in phase 3 may simplify the task of segmenting the surface into NURBS patches.

Inference of higher level primitives It may be desirable to detect and precisely recover geometric surface primitives, such as planar and quadric regions. Current subdivision surface schemes, including ours, do not have quadric precisions (i.e. they cannot model spheres and cylinders exactly). New research on subdivision surfaces may yet solve that limitation. Ideally, one would want to not only recover such primitives, but also infer higher level structure, such as constructive solid geometry (CSG) descriptions.

6.4 Future work on reconstruction of more general manifolds

This thesis has addressed the problem of reconstructing "surfaces"—orientable 2-dimensional manifolds embedded in $\mathbf{R}^3$. As discussed in Section 6.2, our scheme can be adapted to reconstruct curves (1-dimensional manifolds). Future research should explore the reconstruction of more general manifolds, such as non-orientable manifolds and higher dimensional manifolds, as well as non-manifold sets.

Non-orientable manifolds Our reconstruction method may be generalized to allow reconstruction of non-orientable manifolds. In phase 1, although a non-orientable manifold cannot be defined as the zero set of a globally defined signed distance function, it is possible to use such a description locally. Instead of globally orienting the tangent planes as we do now, it may be possible to determine their relative orientations on a cube by cube basis. That is, when generating the contour within a cube, the tangent planes contributing to the function values at the cube's vertices can be oriented relative to each other by considering only a local neighborhood of the Riemannian Graph.

In phases 2 and 3, the current implementation requires the surfaces to be orientable only because of the current half-edge data structure used to represent meshes [75]. Using a different data structure would remove this restriction.

Higher dimensional manifolds In principle, the phase 1 algorithm can be extended to reconstruct manifolds of co-dimension one in spaces of arbitrary dimension; that is, to reconstruct $(k-1)$ dimensional manifolds in k dimensional space. The case of $k = 2$ (curve reconstruction in the plane) was demonstrated in Section 6.2.

The phase 2 and 3 algorithms can trivially optimize 2-dimensional meshes in spaces of higher dimension (e.g. $\mathbf{R}^4$). However, unlike phase 1, these algorithms do not generalize easily to manifolds of higher dimension. A major difficulty is that of dealing with higher dimensional simplicial complexes. Defining a complete set of simplicial complex transformations to allow optimization over 3-dimensional simplicial complexes may be difficult (if at all feasible). Another research area is that of generalizing subdivision schemes to volumes and, more generally, hypersurfaces.

Non-manifolds It may useful to reconstruct non-manifold surfaces, such as three surface sheets meeting along an edge, as well as sets of varying dimensionality, such as surfaces with "hair", or combinations of volumes and surfaces. It is doubtful that the phase 1 algorithm can be extended to these ends. Edelsbrunner's α-shape approach is most encouraging in this regard.

6.5 Future work related to 3D scanning

The field of 3D scanning is likely to grow significantly in the next few decades, thereby motivating a number of pertinent research problems, including:

Automatic generation of scan paths In most current 3D scanning systems, the path taken by the scanhead is either predetermined or specified manually by an operator. When an object to be scanned has complicated geometry, finding a scan paths that completely sample its surface can be tedious and error-prone. Techniques should therefore be developed that, possibly given a partial set of scanned points (for instance, those of Figure 2.11c), automatically determine which surface regions require further sampling, and how to go about scanning them.

Hand-held scanners Most of the weight and expense in current scanning systems is in the machinery required for the precise positioning of the scanhead relative to the object; this machinery is the equivalent of a coordinate measuring machine. To allow the development of more portable and inexpensive scanning systems, an alternative is to only coarsely position the scanhead and to let software perform the task of accurately *registering* the data [5, 72].

Furthermore, for hand-held scanners to become commercially successful, reconstruction algorithms should be real-time and on-line. Even the current phase 1 execution times would require improvements in speed of several orders of magnitude.

Modeling of surface properties Several 3D scanners capture not only (x, y, z) coordinates on a surface, but also information about its color—for instance (r, g, b) color coordinates. This color data should be reconstructed in addition to the geometry of the surface.

One approach is to reconstruct a surface from the (x, y, z) data points, define scalar basis functions on the resulting surface, and finally fit the (r, g, b) color data using these basis functions. While there has been extensive research in the approximation of scalar functions over manifolds of simple topological type, extending this work to manifolds of arbitrary topological type is still a relatively unexplored area. The recent "surfaces on surfaces" work [2, 46] addresses this problem.

Another approach is to view this problem as the reconstruction of a surface in $\mathbf{R}^6$, where the data points have coordinates (x, y, z, r, g, b).

In either case, solving the problem is not as straightforward as it may seem, because color on a surface can have numerous discontinuities (consider the color of this page of text). While our piecewise smooth subdivision schemes can represent non-C^1 functions, color reconstruction requires a non-C^0 representation.

By color we have been referring to the light reflected from an object under a given set of lighting and viewing conditions. More generally, one may want to recover lighting-independent surface reflectance properties, so that the model can then be simulated in other lighting environments. Specialized instruments have been designed to measure bidirectional reflectance distribution functions (BRDF's) [74], but one can envision inferring such reflectance information directly from 3D scanner data.

BIBLIOGRAPHY

[1] E. L. Allgower and P. H. Schmidt. An algorithm for piecewise linear approximation of an implicitly defined manifold. *SIAM Journal of Numerical Analysis*, 22:322–346, April 1985.

[2] R. E. Barnhill, K. Opitz, and H. Pottman. Fat surfaces: a trivariate approach to triangle-based interpolation on surfaces. *CAGD*, 9(5):365–378, November 1992.

[3] J. L. Bentley. Multidimensional divide and conquer. *Comm. ACM*, 23(4):214–229, 1980.

[4] P. J. Besl. Active, optical range imaging sensors. *Machine Vision and Applications*, 1(2):127–152, 1988.

[5] P. J. Besl and H. D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, February 1992.

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

[7] Ruud M. Bolle and Baba C. Vemuri. On three-dimensional surface reconstruction methods. *IEEE Trans. Pat. Anal. Mach. Intell.*, 13(1):1–13, January 1991.

[8] Y. Breseler, J. A. Fessler, and A. Macovski. A Bayesian approach to reconstruction from incomplete projections of a multiple object 3D domain. *IEEE Trans. Pat. Anal. Mach. Intell.*, 11(8):840–858, August 1989.

[9] James F. Brinkley. Knowledge-driven ultrasonic three-dimensional organ modeling. *IEEE Trans. Pat. Anal. Mach. Intell.*, 7(4):431–441, July 1985.

[10] J. L. Brown. Vertex based data dependent triangulations. *CAGD*, 8(3):239–251, August 1991.

[11] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10:350–355, September 1978.

[12] T. DeRose, H. Hoppe, T. Duchamp, J. McDonald, and W. Stuetzle. Fitting of surfaces to scattered data. In J. Warren, editor, *Curves and Surfaces in Computer Vision and Graphics III*. Proc. SPIE 1830:212–220, 1992.

[13] David P. Dobkin, Silvio V. F. Levy, William P. Thurston, and Allan R. Wilks. Contour tracing by piecewise linear approximations. *ACM Transactions on Graphics*, 9(4):389–423, October 1990.

[14] D. Doo and M. Sabin. Behaviour of recursive division surfaces near extraordinary points. *Computer-Aided Design*, 10(6):356–360, September 1978.

[15] N. Dyn, D. Levin, and S. Rippa. Data dependent triangulations for piecewise linear interpolation. *IMA Journal of Numerical Analysis*, 10(1):137–154, January 1990.

[16] N. Dyn and S. Rippa. Data-dependent triangulations for scattered data interpolation and finite element approximation. *Applied Numerical Mathematics*, 12(1-3):89–105, May 1993.

[17] Herbert Edelsbrunner. Weighted alpha shapes. Report 92-1760, Univ. of Illinois at Urbana-Champaign, July 1992.

[18] Herbert Edelsbrunner and Ernst P. Mücke. Three-dimensional alpha shapes. In A. Kaufman and W.E. Lorensen, editors, *Proceedings of 1992 Workshop on Volume Visualization*, pages 75–82, October 1992.

[19] John A. Eisenman. Graphical editing of composite bezier curves. Master's thesis, Department of Electrical Engineering and Computer Science, M.I.T., 1988.

[20] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, 3rd edition, 1992.

[21] Chantal Favardin. *Détermination automatique de structures géométriques destinées à la reconstruction de courbes et de surfaces à partir de données ponctuelles*. PhD thesis, Université Paul Sabatier, Toulouse, France, 1993.

[22] T.A. Foley. Interpolation to scattered data on a spherical domain. In M. Cox and J. Mason, editors, *Algorithms for Approximation II*, pages 303–310. Chapman and Hall, London, 1990.

[23] Michael R. Garey and David S. Johnson. *Computers and Intractability*. W. H. Freeman and Company, 1979.

[24] Gene Golub and Charles Van Loan. *Matrix Computations*. John Hopkins University Press, 2nd edition, 1989.

[25] Ardeshir Goshtasby. Surface reconstruction from scattered measurements. *SPIE*, 1830:247–256, 1992.

[26] Mark Halstead, Michael Kass, and Tony DeRose. Efficient, fair interpolation using Catmull-Clark surfaces. *Computer Graphics (SIGGRAPH '93 Proceedings)*, pages 35–44, August 1993.

[27] T. Hastie and W. Stuetzle. Principal curves. *JASA*, 84:502–516, 1989.

[28] K. Hisanaga, A. Hisanaga, K. Nagata, and S. Yoshida. A new transesophageal real-time two-dimensional echocardiographic system using a flexible tube and its clinical application. *Proc. Jpn. Soc. Ultrason. Med.*, 32:43, 1977.

[29] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise smooth surface reconstruction. *Computer Graphics (SIGGRAPH '94 Proceedings)*, pages 295–302, July 1994.

[30] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):71–78, July 1992.

[31] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. *Computer Graphics (SIGGRAPH '93 Proceedings)*, pages 19–26, August 1993.

[32] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. TR 93-01-01, Dept. of Computer Science and Engineering, University of Washington, January 1993.

[33] Charles Loop. Smooth subdivision surfaces based on triangles. Master's thesis, Department of Mathematics, University of Utah, August 1987.

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

[35] Michael Lounsbery, Tony DeRose, and Joe Warren. Multiresolution analysis for surfaces of arbitrary topological type. TR 93-10-05b, Dept. of Computer Science and Engineering, University of Washington, October 1993.

[36] Michael Lounsbery, Stephen Mann, and Tony DeRose. Parametric surface interpolation. *IEEE Computer Graphics and Applications*, 12(5):45–52, September 1992.

[37] Samuel Marin and Philip Smith. Parametric approximation of data using ODR splines. GMR 7057, General Motors Research Laboratories, May 1990.

[38] Marshal L. Merriam. Experience with the cyberware 3D digitizer. In *NCGA Proceedings*, pages 125–133, March 1992.

[39] David Meyers. Multiresolution tiling. In *Proceedings of Graphics Interface '94*, pages 25–32, May 1994.

[40] David Meyers, Shelly Skinner, and Kenneth Sloan. Surfaces from contours. *ACM Transactions on Graphics*, 11(3):228–258, July 1992.

[41] J.V. Miller, D.E. Breen, W.E. Lorensen, R.M. O'Bara, and M.J. Wozny. Geometrically deformed models: A method for extracting closed geometric models from volume data. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):217–226, July 1991.

[42] Doug Moore and Joe Warren. Approximation of dense scattered data using algebraic surfaces. TR 90-135, Rice University, October 1990.

[43] Shigeru Muraki. Volumetric shape description of range data using "blobby model". *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):227–235, July 1991.

[44] Ahmad H. Nasri. Polyhedral subdivision methods for free-form surfaces. *ACM Transactions on Graphics*, 6(1):29–73, January 1987.

[45] Ahmad H. Nasri. Boundary-corner control in recursive-subdivision surfaces. *Computer Aided Design*, 23(6):405–410, July-August 1991.

[46] Gregory M. Nielson, Thomas A. Foley, Bernd Hamann, and David Lane. Visualizing and modeling scattered multivariate data. *IEEE CG&A*, 11(3):47–55, May 1991.

[47] Barrett O'Neill. *Elementary Differential Geometry*. Academic Press, Orlando, Florida, 1966.

[48] Joseph O'Rourke. Polyhedra of minimal area as 3D object models. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 664–666, 1981.

[49] Michael Plass and Maureen Stone. Curve-fitting with piecewise parametric cubics. *Computer Graphics (SIGGRAPH '83 Proceedings)*, 17(3):229–239, July 1983.

[50] Vaughan Pratt. Direct least-squares fitting of algebraic surfaces. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21(4):145–152, July 1987.

[51] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Fetterling. *Numerical Recipes*. Cambridge University Press, Cambridge, 1986.

[52] Ulrich Reif. A unified approach to subdivision algorithms. Mathematisches Institute A 92-16, Universität Stuttgart, 1992.

[53] Jarek Rossignac and Paul Borrel. Multi-resolution 3D approximations for rendering complex scenes. In B. Falcidieno and T. L. Kunii, editors, *Modeling in Computer Graphics*, pages 455–465. Springer-Verlag, New York, 1993.

[54] Emanuel Sachs, Andrew Roberts, and David Stoops. 3-Draw: A tool for designing 3D shapes. *IEEE Computer Graphics and Applications*, 11(6):18–26, November 1991.

[55] Hanan Samet. *Applications of Spatial Data Structures*. Addison-Wesley, 1990.

[56] Lori Scarlatos. *Spatial data representations for rapid visualization and analysis*. PhD thesis, Dept. of Computer Science, SUNY at Stony Brook, August 1993.

[57] F. Schmitt, B.A. Barsky, and W. Du. An adaptive subdivision method for surface fitting from sampled data. *Computer Graphics (SIGGRAPH '86 Proceedings)*, 20(4):179–188, 1986.

[58] F. Schmitt, X. Chen, W. Du, and F. Sair. Adaptive G^1 approximation of range data using triangular patches. In P.J. Laurent, A. Le Mehaute, and L.L. Schumaker, editors, *Curves and Surfaces*. Academic Press, 1991.

[59] Philip J. Schneider. Phoenix: An interactive curve design system based on the automatic fitting of hand-sketched curves. Master's thesis, Department of Computer Science, University of Washington, 1988.

[60] William Schroeder, Jonathan Zarge, and William Lorensen. Decimation of triangle meshes. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):65–70, July 1992.

[61] R. B. Schudy and D. H. Ballard. Model detection of cardiac chambers in ultrasound images. Technical Report 12, Computer Science Department, University of Rochester, 1978.

[62] R. B. Schudy and D. H. Ballard. Towards an anatomical model of heart motion as seen in 4-d cardiac ultrasound data. In *Proceedings of the 6th Conference on Computer Applications in Radiology and Computer-Aided Analysis of Radiological Images*, 1979.

[63] Larry Schumaker. Computing optimal triangulations using simulated annealing. *Computer-Aided Geometric Design*, 10(3-4):329–345, August 1993.

[64] Jean Schweitzer and Tom Duchamp. An analysis of piecewise smooth subdivision. In preparation.

[65] Stan Sclaroff and Alex Pentland. Generalized implicit functions for computer graphics. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):247–250, July 1991.

[66] E. H. Spanier. *Algebraic Topology*. McGraw-Hill, New York, 1966.

[67] R. Szeliski, D. Tonnesen, and D. Terzopoulos. Curvature and continuity control in particle-based surface models. In *SPIE Conference on Geometric Methods in Computer Vision II*, pages 172–181. SPIE, July 1993.

[68] Richard Szeliski and David Tonnesen. Surface modeling with oriented particle systems. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):185–194, July 1992.

[69] Gabriel Taubin. Estimation of planar curves, surfaces and nonplanar space curves defined by implicit equations, with applications to edge and range image segmentation. Technical Report LEMS-66, Division of Engineering, Brown University, 1990.

[70] Gabriel Taubin and Remi Ronfard. Implicit simplicial models I: adaptive curve reconstruction. Technical Report RC-18887, IBM Research Division, May 1993.

[71] Greg Turk. Re-tiling polygonal surfaces. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):55–64, July 1992.

[72] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. *Computer Graphics (SIGGRAPH '94 Proceedings)*, July 1994.

[73] Remco Veltkamp. *Closed Object Boundaries from Scattered Points*. PhD thesis, Erasmus Universiteit Rotterdam, The Netherlands, 1992.

[74] Gregory J. Ward. Measuring and modeling anisotropic reflection. *Computer Graphics*, 26(2):265–272, July 1992.

[75] K. Weiler. Edge-based data structures for solid modeling in curved-surface environments. *IEEE CG&A*, 5(1):21–40, January 1985.

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