Mesh Optimization

Hugues Hoppe* Tony DeRose* Tom Duchamp†
John McDonald‡ Werner Stuetzle‡

University of Washington
Seattle, WA 98195

Abstract

We present a method for solving the following problem: Given a set of data points scattered in three dimensions and an initial triangular mesh M_0, produce a mesh M, of the same topological type as M_0, that fits the data well and has a small number of vertices. Our approach is to minimize an energy function that explicitly models the competing desires of conciseness of representation and fidelity to the data. We show that mesh optimization can be effectively used in at least two applications: surface reconstruction from unorganized points, and mesh simplification (the reduction of the number of vertices in an initially dense mesh of triangles).

CR Categories and Subject Descriptors: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling.

Additional Keywords: Geometric Modeling, Surface Fitting, Three-Dimensional Shape Recovery, Range Data Analysis, Model Simplification.

1 Introduction

The *mesh optimization* problem considered in this paper can be roughly stated as follows: Given a collection of data points X in $\mathbf{R}^3$ and an initial triangular mesh M_0 near the data, find a mesh M of the same topological type as M_0 that fits the data well and has a small number of vertices.

As an example, Figure 7b shows a set of 4102 data points sampled from the object shown in Figure 7a. The input to the mesh optimization algorithm consists of the points together with the initial mesh shown in Figure 7c. The optimized mesh is shown in Figure 7h. Notice that the sharp edges and corners indicated by the data have been faithfully recovered and that the number of vertices has been significantly reduced (from 1572 to 163).

*Department of Computer Science and Engineering, FR-35
†Department of Mathematics, GN-50
‡Department of Statistics, GN-22

This work was supported in part by Bellcore, the Xerox Corporation, IBM, Hewlett-Packard, AT&T Bell Labs, the Digital Equipment Corporation, the Department of Energy under grant DE-FG06-85-ER25006, the National Library of Medicine under grant NIH LM-04174, and the National Science Foundation under grants CCR-8957323 and DMS-9103002.

To solve the mesh optimization problem we minimize an *energy function* that captures the competing desires of tight geometric fit and compact representation. The tradeoff between geometric fit and compact representation is controlled via a user-selectable parameter c_{rep}. A large value of c_{rep} indicates that a sparse representation is to be strongly preferred over a dense one, usually at the expense of degrading the fit.

We use the input mesh M_0 as a starting point for a non-linear optimization process. During the optimization we vary the number of vertices, their positions, and their connectivity. Although we can give no guarantee of finding a global minimum, we have run the method on a wide variety of data sets; the method has produced good results in all cases (see Figure 1).

We see at least two applications of mesh optimization: surface reconstruction and mesh simplification.

The problem of surface reconstruction from sampled data occurs in many scientific and engineering applications. In [2], we outlined a two phase procedure for reconstructing a surface from a set of unorganized data points. The goal of phase one is to determine the topological type of the unknown surface and to obtain a crude estimate of its geometry. An algorithm for phase one was described in [5]. The goal of phase two is to improve the fit and reduce the number of faces. Mesh optimization can be used for this purpose.

Although we were originally led to consider the mesh optimization problem by our research on surface reconstruction, the algorithm we have developed can also be applied to the problem of mesh simplification. Mesh simplification, as considered by Turk [15] and Schroeder et al. [10], refers to the problem of reducing the number of faces in a dense mesh while minimally perturbing the shape. Mesh optimization can be used to solve this problem as follows: sample data points X from the initial mesh and use the initial mesh as the starting point M_0 of the optimization procedure. For instance, Figure 7q shows a triangular approximation of a minimal surface with 2032 vertices. Application of our mesh optimization algorithm to a sample of 6752 points (Figure 7r) from this mesh produces the meshes shown in Figures 7s (487 vertices) and 7t (239 vertices). The mesh of Figure 7s corresponds to a relatively small value of c_{rep}, and therefore has more vertices than the mesh of Figure 7t which corresponds to a somewhat larger value of c_{rep}.

The principal contributions of this paper 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.). During the fitting process, the number and connectivity of the vertices, as well as their positions, are allowed to vary.
- It casts mesh simplification as an optimization problem with an energy function that directly measures deviation of the final mesh from the original. As a consequence, the final mesh

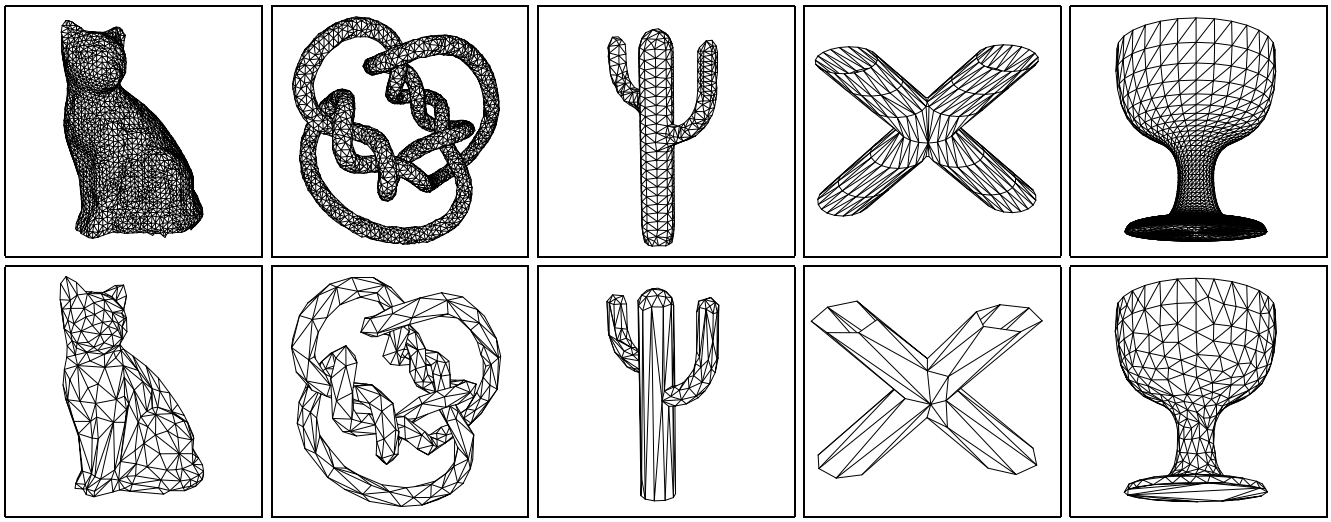


Figure 1: Examples of mesh optimization. The meshes in the top row are the initial meshes M_0; the meshes in the bottom row are the corresponding optimized meshes. The first 3 columns are reconstructions; the last 2 columns are simplifications.

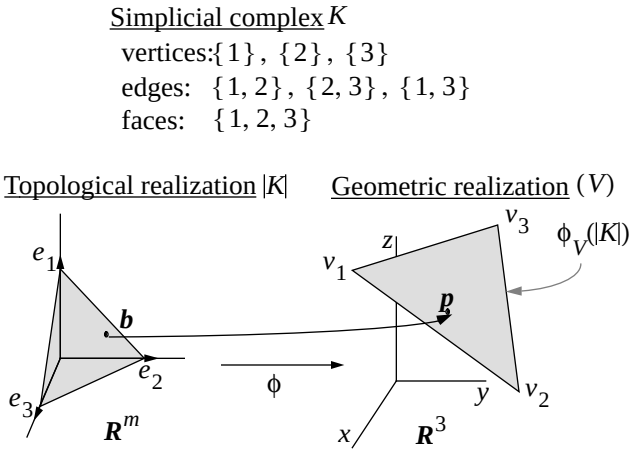


Figure 2: Example of mesh representation: a mesh consisting of a single face.

naturally adapts to curvature variations in the original mesh.

- 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 (see Figure 7i).

2 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 vertices and their geometric positions. 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).

A simplicial complex K consists of a set of vertices $\{1, \ldots, m\}$, together with a set of non-empty subsets of the vertices, called the simplices of K, such that any set consisting of exactly one vertex is a simplex in K, and every non-empty subset of a simplex in K is again a simplex in K (cf. Spanier [14]). The 0-simplices $\{i\} \in K$ are called vertices, 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 $\phi : \mathbf{R}^m \to \mathbf{R}^3$ 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 2).

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

If ϕ_V is an embedding, any point $\mathbf{p} \in \phi_V(|K|)$ can be parameterized by finding its unique pre-image on $|K|$. The vector $\mathbf{b} \in |K|$ with $\mathbf{p} = \phi_V(\mathbf{b})$ 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.

3 Definition of the Energy Function

Recall that the goal of mesh optimization is to obtain a mesh that provides a good fit to the point set X and has a small number of vertices. We find a simplicial complex K and a set of vertex positions V defining a mesh $M = (K, V)$ that minimizes the energy function

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

The first two terms correspond to the two stated goals; the third term is motivated below.

The distance energy E_{dist} is equal to the sum of squared distances from the points $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ to the mesh,

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

The representation energy E_{rep} penalizes meshes with a large number of vertices. It is set to be proportional to the number of vertices m 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} makes this operation incur a penalty 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 [8], that minimizing $E_{dist} + E_{rep}$ does not produce the desired results. As an illustration of what can go wrong, Figure 7d 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.

To guarantee the existence of a minimum [6], 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:

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

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 to a desirable local minimum. As the optimization converges to the solution, the magnitude of E_{spring} can be gradually reduced. We return to this issue in Section 4.4.

For some applications we want the procedure to be scale-invariant, which is 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 pre-scaled uniformly to fit in a unit cube. After optimization, a post-processing step can undo this initial transformation.

4 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. We now present an outline of our optimization algorithm, a pseudo-code version of which appears in Figure 3. The details are deferred to the next two subsections.

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

In Section 4.1 we describe an algorithm that solves the inner minimization problem. It finds $E(K) = \min_V E(K, V)$, 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.

```
OptimizeMesh(K0,V0) {
    K := K0
    V := OptimizeVertexPositions(K0,V0)
    – 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: An idealized pseudo-code version of the minimization algorithm.

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 4.2.

The energy function $E(K, V)$ depends on two parameters c_{rep} and κ. The parameter c_{rep} controls the tradeoff 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, ideally, would be chosen automatically. Our method of setting κ is described in Section 4.4.

4.1 Optimization for Fixed Simplicial Complex
(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 $M = \phi_V(|K|)$. Each of these distances is itself the solution to the minimization problem

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

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. Thus, minimizing $E(K,V)$ for fixed K is equivalent to minimizing the new objective function

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

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

To solve this optimization problem (procedure OptimizeVertexPositions), our 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. 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 7e shows the result of optimizing the mesh of Figure 7c 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. ϕ_V is not an embedding. While it is possible to check *a posteriori* whether ϕ_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.

4.1.1 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:

$$\mathbf{b}_i = \operatorname*{argmin}_{\mathbf{b}\in|K|} \|\mathbf{x}_i - \phi_V(\mathbf{b})\|$$

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

A naive approach to computing $\mathbf{b}_i$ is to project $\mathbf{x}_i$ onto all of the faces of M, and then find the projection with minimal distance. To speed up the projection, we first enter the faces of the mesh into a spatial partitioning data structure (similar to the one used in [16]). Then for each point $\mathbf{x}_i$ only a nearby subset of the faces needs to be considered, and the projection step takes expected time $O(n)$. For additional speedup we exploit coherence between iterations. Instead of projecting each point globally onto the mesh, we assume that a point's projection lies in a neighborhood of its projection in the previous iteration. Specifically, we project the point onto all faces that share a vertex with the previous face. Although this is a heuristic that can fail, it has performed well in practice.

4.1.2 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 $O(m)$. Let $\mathbf{v}^1$ be the m-vector whose i-th element is the first coordinate of $\mathbf{v}_i$. Let $\mathbf{d}^1$ be the $(n+e)$-vector whose first n elements are the first coordinates of the data points $\mathbf{x}_i$, and whose last e elements are zero. With these definitions we can express the least squares problem for the first coordinate as minimizing $\|A\mathbf{v}^1 - \mathbf{d}^1\|^2$ over $\mathbf{v}^1$. The design matrix A is an $(n+e) \times m$ matrix of scalars. The first n rows of A are the barycentric coordinate vectors $\mathbf{b}_i$. Each of the trailing e rows contains 2 non-zero entries with values $\sqrt{\kappa}$ and $-\sqrt{\kappa}$ in the columns corresponding to the indices of the edge's endpoints. The first n rows of the least squares problem correspond to $E_{dist}(K,V)$, while the last e rows correspond to $E_{spring}(K,V)$. 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.

To solve the least squares problem, we use the conjugate gradient method (cf. [3]). 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. We store A in a sparse form that requires only $O(n+m)$ space. Thus, an acceptable solution to the least squares problem is obtained in $O(n+m)$ time. 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.

4.2 Optimization over Simplicial Complexes
(Procedure OptimizeMesh)

To solve the outer minimization problem, minimizing $E(K)$ over K, we define a set of three elementary transformations, *edge collapse*, *edge split*, and *edge swap*, taking a simplicial complex K to another simplicial complex K' (see Figure 4).

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[1].

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 try again. If a large number of trials fails to produce an acceptable move, we terminate the search.

More elaborate selection strategies, such as steepest descent or simulated annealing, are possible. As we have obtained good results with the simple strategy of random descent, we have not yet implemented the other strategies.

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

[1] In fact, we prove in [6] 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.

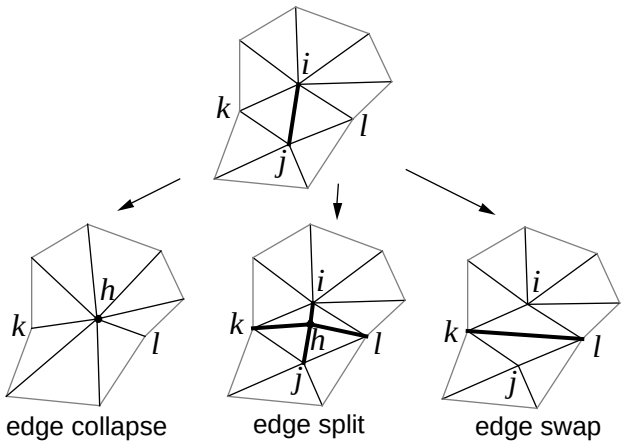


Figure 4: Local simplicial complex transformations

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 (proof in [6]):

- 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 $\{k, l\} \notin K$.

4.3 Exploiting Locality

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

4.3.1 Heuristics for Evaluating the Effect of Legal Moves

Our strategy for selecting legal moves 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 4.1 for this purpose, as indicated in Figure 3. 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 only considering the contribution of the submesh and the corresponding point set. This estimate is always pessimistic, as full optimization would only further reduce the energy.

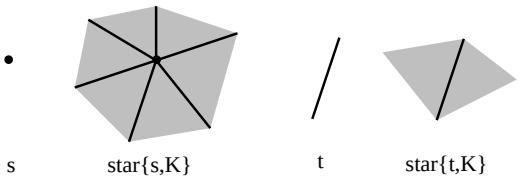


Figure 5: Neighborhood subsets of K.

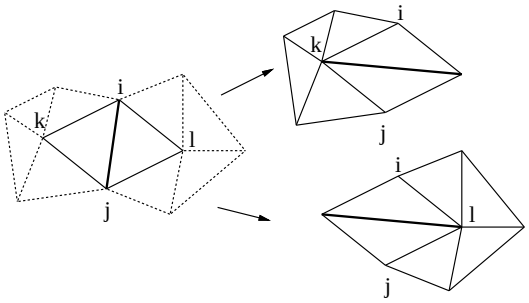


Figure 6: Two local optimizations to evaluate edge swap

Therefore, the heuristics never suggest changes that will increase the true energy of the mesh.

Definition of neighborhoods in a simplicial complex To refer to neighborhoods in a simplicial complex, we need to introduce some further notation. We write $s' \leq s$ to denote that simplex s' is a non-empty subset of simplex s. For simplex $s \in K$, $\text{star}(s; K) = \{s' \in K \ : \ s \leq s'\}$ (Figure 5).

Evaluation of Edge Collapse To evaluate a transformation $K \Rightarrow K'$ collapsing an edge $\{i, j\}$ into a single vertex $\{h\}$ (Figure 4), we take the submesh to be $\text{star}(\{i\}; K) \cup \text{star}(\{j\}; K)$, 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 attempt three optimizations, with $\mathbf{v}_h$ starting at $\mathbf{v}_i$, $\mathbf{v}_j$, and $\frac{1}{2}(\mathbf{v}_i + \mathbf{v}_j)$, and accept the best one.

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 $\text{star}(\{h\}; K')$ is greater than some threshold, the edge collapse is rejected.

Evaluation of Edge Split The procedure is the same as for edge collapse, except that the submesh is defined to be $\text{star}(\{i, j\}; K)$, and the initial value of the new vertex $\mathbf{v}_h$ is 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 $\text{star}(\{k\}; K')$, varying vertex $\mathbf{v}_k$, and one with submesh $\text{star}(\{l\}; K')$, varying vertex $\mathbf{v}_l$ (Figure 6). The change in energy is taken to best of these. 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.

4.3.2 Legal Move Selection Strategy
(Procedure GenerateLegalMove)

The simple strategy for selecting legal moves described in Section 4.2 can be improved by exploiting locality. Instead of selecting edges completely at random, edges are selected 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.

4.4 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 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 7h starting from the mesh in Figure 7c, we successively set κ to $10^{-2}, 10^{-3}, 10^{-4}$, and 10^{-8} (see Figures 7f–7h). This same schedule was used in all the examples.

5 Results

5.1 Surface Reconstruction

From the set of points shown in Figure 7b, phase one of our reconstruction algorithm [5] produces the mesh shown in Figure 7c; this mesh has the correct topological type, but it is rather dense, is far away from the data, and lacks the sharp features of the original model (Figure 7a). Using this mesh as a starting point, mesh optimization produces the mesh in Figure 7h.

Figures 7i–7k,7m–7o show two examples of surface reconstruction from actual laser range data (courtesy of Technical Arts, Redmond, WA). Figures 7j and 7n show sets of points obtained by sampling two physical objects (a distributor cap and a golf club head) with a laser range finder. The outputs of phase one are shown in Figures 7k and 7o. The holes present in the surface of Figure 7k 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 filled the holes, leaving a single boundary at the bottom. Figures 7l and 7p show the optimized meshes obtained with our algorithm.

5.2 Mesh Simplification

For mesh simplification, we first sample a set of points randomly from the original mesh using uniform random sampling over area. 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.

As an example of mesh simplification, we start with the mesh containing 2032 vertices shown in Figure 7q. From it, we obtain a sample of 6752 points shown in Figure 7r (4000 random points, 2032 vertex points, and 720 boundary points). Mesh optimization, with $c_{rep} = 10^{-5}$, reduces the mesh down to 487 vertices (Figure 7s). By setting $c_{rep} = 10^{-4}$, we obtain a coarser mesh of 239 vertices (Figure 7t).

Fig.	#vert. m	#faces	#data n	Parameters c_{rep}	Parameters κ	Resulting energies E_{dist}	Resulting energies E	time (min.)
7c	1572	3152	4102	-	-	8.57×10^{-2}	-	-
7e	1572	3152	4102	10^{-5}	10^{-2}	8.04×10^{-4}	4.84×10^{-2}	1.5
7f	508	1024	4102	10^{-5}	10^{-2}	6.84×10^{-4}	3.62×10^{-2}	(+3.0)
7g	270	548	4102	10^{-5}	10^{-3}	6.08×10^{-4}	6.94×10^{-3}	(+2.2)
7h	163	334	4102	10^{-5}	*varied*	4.86×10^{-4}	2.12×10^{-3}	17.0
7k	9220	18272	12745	-	-	6.41×10^{-2}	-	-
7l	690	1348	12745	10^{-5}	*varied*	4.23×10^{-3}	1.18×10^{-2}	47.0
7o	4059	8073	16864	-	-	2.20×10^{-2}	-	-
7p	262	515	16864	10^{-5}	*varied*	2.19×10^{-3}	4.95×10^{-3}	44.5
7q	2032	3832	-	-	-	-	-	-
7s	487	916	6752	10^{-5}	*varied*	1.86×10^{-3}	8.05×10^{-3}	9.9
7t	239	432	6752	10^{-4}	*varied*	9.19×10^{-3}	4.39×10^{-2}	10.2

Table 1: Performance statistics for meshes shown in Figure 7.

As these examples illustrate, basing mesh simplification on a measure of distance between the simplified mesh and the original 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.

5.3 Segmentation

Mesh optimization enables us to detect 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 7i shows the segmentation of the optimized mesh into 11 components. After segmentation, vertex normals can be estimated from neighboring faces within each component, and a smoothly shaded surface can be created (Figure 7m).

5.4 Parameter Settings and Performance Statistics

Table 1 lists the specific parameter values of c_{rep} and κ used to generate the meshes in the examples, along with other performance statistics. In all these examples, the table entry *"varied"* refers to a spring constant schedule of $\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-8}\}$. In fact, all meshes in Figure 1 are also created using the same parameters (except that c_{rep} was changed in two cases). Execution times were obtained on a DEC uniprocessor Alpha workstation.

6 Related Work

Surface Fitting There is a large body of literature on fitting embeddings of a rectangular domain; see Bolle and Vemuri [1] for a review. Schudy and Ballard [11, 12] fit embeddings of a sphere to point data. Goshtasby [4] works with embeddings of cylinders and tori. Sclaroff and Pentland [13] consider embeddings of a deformed superquadric. Miller et al. [9] approximate an isosurface of volume data by fitting a mesh homeomorphic to a sphere. While it appears that their method could be extended to finding isosurfaces of arbitrary topological type, it it less obvious how it could be modified to

handle point instead of volume data. Mallet [7] discusses interpolation of functions over simplicial complexes of arbitrary topological type.

Our method allows fitting of a parametric surface of arbitrary topological type to a set of three-dimensional points. In [2], we sketched an algorithm for fitting a mesh of *fixed* vertex connectivity to the data. The algorithm presented here is an extension of this idea in which we also allow the number of vertices and their connectivity to vary. To the best of our knowledge, this has not been done before.

Mesh Simplification Two notable papers discussing the mesh simplification problem are Schroeder et al. [10] and Turk [15].

The motivation of Schroeder et al. is to simplify meshes generated by "marching cubes" that may consist of more than a million 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.

Turk's goal is to reduce the amount of detail in a mesh while remaining faithful to the original topology and geometry. His basic idea is to distribute points on the existing mesh that are to become the new vertices. He then creates a triangulation containing both old and new vertices, and finally removes the old vertices. The density of the new vertices is chosen to be higher in areas of high curvature.

The principal advantage of our mesh simplification method compared to the techniques mentioned above 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 retiling 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.

7 Summary and Future Work

We have described an energy minimization approach to solving the mesh optimization problem. The energy function we use consists of three terms: a distance energy that measures the closeness of fit, a representation energy that penalizes meshes with a large number of vertices, and a regularizing term that conceptually places springs of rest length zero on the edges of the mesh. Our minimization algorithm partitions the problem into two nested subproblems: an inner continuous minimization and an outer discrete minimization. The search space consists of all meshes homeomorphic to the starting mesh.

Mesh optimization has proven effective as the second phase of our method for surface reconstruction from unorganized points, as discussed in [5]. (Phase two is responsible for improving the geometric fit and reducing the number of vertices of the mesh produced in phase one.)

Our method has also performed well for mesh simplification, that is, the reduction of the number of vertices in a dense triangular mesh. It produces meshes whose edges align themselves along directions of low curvature, and whose vertices concentrate in areas of high Gaussian curvature. Because the energy does not penalize surfaces with sharp dihedral angles, the method can recover sharp edges and corners.

A number of areas of future research still remain, including:

- Investigate the use of more sophisticated optimization methods, such as simulated annealing for discrete optimization and quadratic methods for non-linear least squares optimization, in order to avoid undesirable local minima in the energy and to accelerate convergence.
- Gain more insight into the use of the spring energy as a regularizing term, especially in the presence of appreciable noise.
- Improve the speed of the algorithm and investigate implementations on parallel architectures.
- Develop methods for fitting higher order splines to more accurately and concisely model curved surfaces.
- Experiment with sparse, non-uniform, and noisy data.
- Extend the current algorithm to other distance measures such as maximum error (L^∞ norm) or average error (L^1 norm), instead of the current L^2 norm.

References

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

[2] T. DeRose, H. Hoppe, T. Duchamp, J. McDonald, and W. Stuetzle. Fitting of surfaces to scattered data. *SPIE*, 1830:212–220, 1992.

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

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

[5] 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.

[6] 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.

[7] J.L. Mallet. Discrete smooth interpolation in geometric modeling. *CAD*, 24(4):178–191, April 1992.

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

[9] 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.

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

[11] 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.

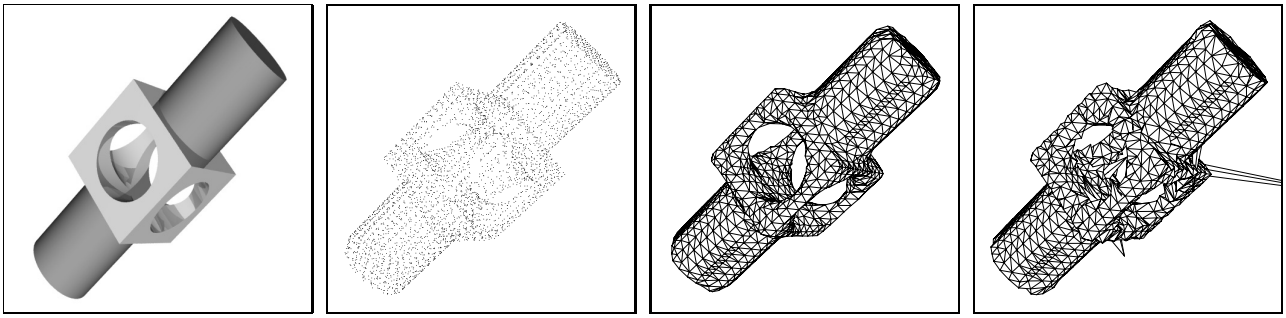
[12] R. B. Schudy and D. H. Ballard. Towards an anatomical model of heart motion as seen in 4-d cardiac ultrasound data. In *Proceedings of the 6th Conference on Computer Applications in Radiology and Computer-Aided Analysis of Radiological Images*, 1979.

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

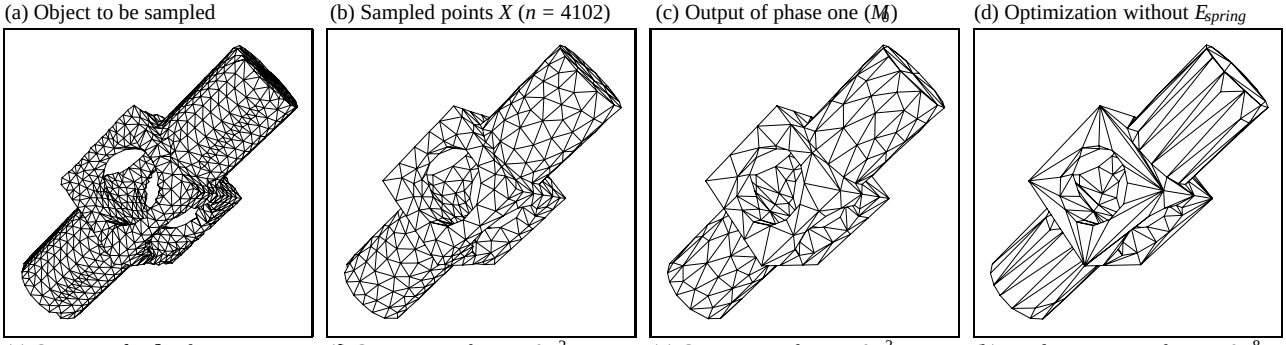
[14] E. H. Spanier. *Algebraic Topology*. McGraw-Hill, New York, 1966.

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

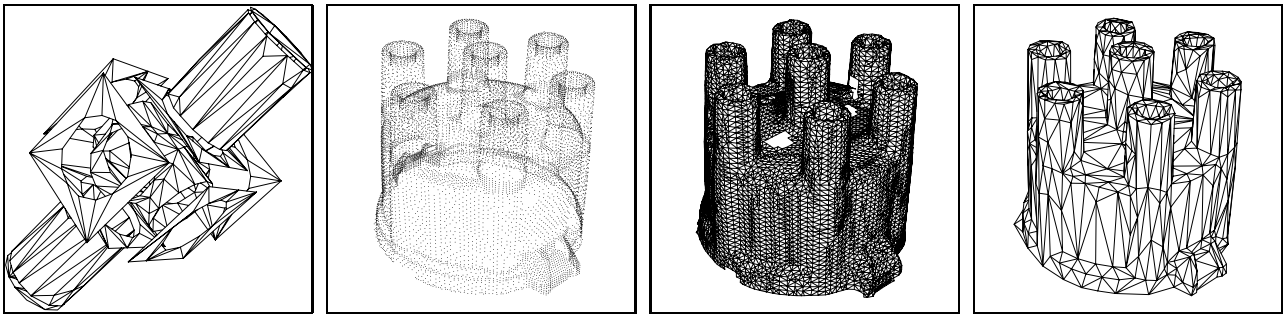
[16] G. Wyvill, C. McPheeters, and B. Wyvill. Data structures for soft objects. *The Visual Computer*, 2(4):227–234, August 1986.



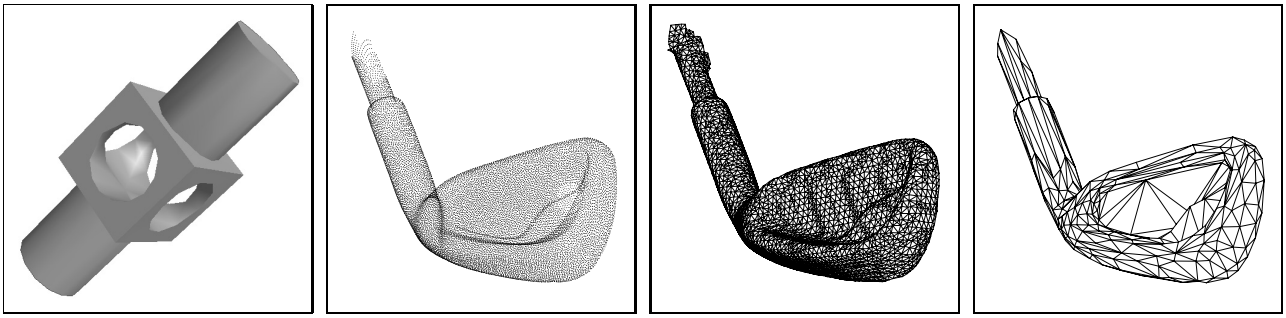
(a) Object to be sampled (b) Sampled points X (n = 4102) (c) Output of phase one (M_0) (d) Optimization without E_{spring}



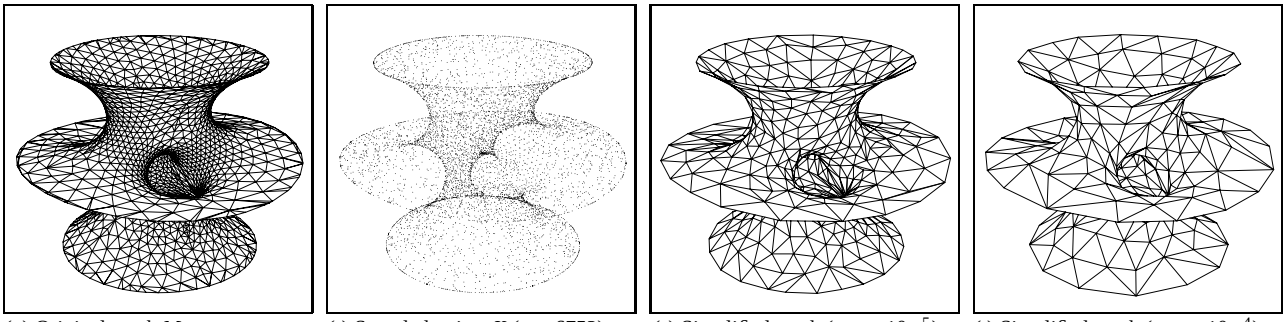
(e) Optimum for fixed K_0 (f) Optimum with $\kappa = 10^{-2}$ (g) Optimum with $\kappa = 10^{-3}$ (h) Final optimum with $\kappa = 10^{-8}$



(i) Segmented surface (11 comp.) (j) Laser range data ($n = 12,745$) (k) Output of phase one (l) Output of phase two



(m) Smooth shading from segments (n) Laser range data ($n = 16,864$) (o) Output of phase one (p) Output of phase two



(q) Original mesh M_0 (r) Sampled points X (n = 6752) (s) Simplified mesh ($c_{ep} = 10^{-5}$) (t) Simplified mesh ($c_{rep} = 10^{-4}$)

Figure 7: Examples of surface reconstruction and mesh simplification.