Multilevel Sensitive Reconstruction of Polyhedral Surfaces from Parallel Slices

Gill Barequet* Daniel Shapiro† Ayellet Tal‡

Abstract

We present an algorithm for reconstructing a solid model from a series of planar cross-sections. In most previous works the layers are assumed to be independent: each layer is interpolated separately and the concatenation of the interpolated layers is considered the solution to the whole problem. The resulting surface can therefore exhibit abrupt changes. The main contribution of this work is avoiding this assumption. We use the slopes of triangles created in the interpolation of neighboring layers to guide the interpolation of the current layer. As a result, consecutive layers are connected smoothly. We also discuss in this paper various objective functions which aim to optimize the reconstruction and evaluate these functions using various criteria.

Keywords: surface reconstruction, interpolation, triangulation.

1 Introduction

The reconstruction of a polyhedral object from a series of cross-sections has been an intriguing problem during the last couple of decades. The main motivations for this problem come from medical imaging, digitization of objects, and GIS systems. Data obtained by CT or MRI apparata, range sensors, or elevation contours, are

*Faculty of Computer Science, The Technion—Israel Institute of Technology, Haifa 32000, Israel. E-mail: `barequet@cs.technion.ac.il` (This work was done while this author was affiliated with the Dept. of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel, and with the Center for Geometric Computing, Dept. of Computer Science, Johns Hopkins University, Baltimore, MD 21218.)

†Dept. of Computer Science, Princeton University, Princeton, NJ 08540. E-mail: `dans@cs.princeton.edu`

‡Faculty of Electrical Engineering, Technion—Israel Institute of Technology, Haifa 3200, Israel. E-mail: `ayellet@ee.technion.ac.il` (Most of this work was done while this author was affiliated with the Dept. of Applied Mathematics, The Weizmann Institute of Science, Rehovot 76100, Israel.)

interpolated in order to represent and visualize human organs, CAD objects, or topographic terrains. Visualizing the three-dimensional dataset is usually done in one of two ways (Kaufman 1991): either by projecting the data primitives directly into two-dimensions, or by first converting them into an imaginary approximating surface, which is then visualized by projecting to 2D.

Continuous three-dimensional reconstruction methods are inspired by image processing techniques and treat the data as a three-dimensional image obtained by piling up the images of the cross-sections. Either interpolation (linear, bilinear, trilinear, non-linear, shaped-based, etc.) or slice replication is used to fill in the gaps between slices; see, e.g., (Vannier et al. 1983; Dev et al. 1984; Farrell et al. 1985; Woolson et al. 1985; Raya and Udupa 1990). Then, volume rendering techniques (see, e.g., (Drebin et al. 1988; Levoy 1988; Upson and Keeler 1988; Frenkel 1989)) project the three-dimensional image into a two-dimensional pixel space. These methods enable the user to view the interior parts of an object. These methods are very simple; however, they require large memory resources and powerful processing units.

Alternatively, the volume data can first be converted into geometric primitives (e.g., polygon meshes) and then be displayed on screen by conventional surface rendering. Surface reconstruction results in a significant reduction in data storage and can exploit existing software for fast manipulation and visualization. This allows the user to quickly examine many different viewing spaces. The major drawback of this strategy is the use of a thresholding scheme (for determining the boundary voxels), which is not always appropriate.

Surface reconstruction algorithms are usually grouped into two basic categories (Kalvin 1992): two- and three-dimensional based algorithms. In the three-dimensional approach (Lorensen and Cline 1987; Wilhelms and van Gelder 1990; Nielson and Hamann 1991) the connectivity between the boundary elements is derived directly from the voxel structure with no intermediate extraction of two-dimensional contours. The resulting surface consists of a large number of very fine triangles.

In the two-dimensional based approach the surface reconstruction is done in two stages. First, the closed two-dimensional contours, which delimit the material regions on each slice, are extracted. Then, an imaginary approximating surface that tiles these contours is constructed. This approach leads to a significantly more concise polyhedral representation since the number of vertices, edges, and faces can be greatly reduced. In this paper we describe a two-dimensional based surface construction algorithm.

Various algorithms for two-dimensional based polyhedral surface construction have been suggested in the literature (e.g., in (Keppel 1975; Fuchs et al. 1977; Batnitzky et al. 1981; Ganapathy and Dennehy 1982; Wang and Aggarwal 1986; Kehtarnavaz and de Figueiredo 1988; Kehtarnavaz, Simar, and de Figueiredo 1988; Sloan and Painter 1988; Welzl and Wolfers 1994; Fix and Ladner 1998)). The earlier algorithms suffer from various problems. Many of them fail in complex instances (such as multiple branching), leave gaps between the contours, and generate unacceptable solutions (e.g., self-intersecting surfaces). Some algorithms (Christiansen and Sederberg 1978; Shantz 1981; Zyda et al. 1987; Ekoule et al. 1991; Meyers et al. 1992; Choi and Park 1994) reduce the more involved cases to the simple case where each slice contains only one contour. There have been only a few attempts (Boissonnat 1988; Boissonnat and Geiger 1993; Barequet and Sharir 1996; Bajaj et al. 1996) to handle the interpolation problem in full generality without limiting the number of contours in the slices, their geometries, or their containment hierarchies.

A practical simplification assumed in most of the previous works is that adjacent layers are independent. Thus, only a single pair of successive parallel slices are considered and interpolated at a time. The reconstructed object is the concatenation of the interpolating models computed for all the layers. This assumption, however, is not always practical, and may result in an object whose boundary makes sharp turns in the transition from one layer to another. We are not aware of any polyhedral interpolation method which handles the interpolation in a given layer while considering the interpolations in adjacent layers.

A novel aspect of our work is avoiding this simplification. Our work is based on the algorithm of (Barequet and Sharir 1996) for interpolating between two successive slices. We extend this algorithm so that it incorporates data from neighboring layers. This extension improves considerably the quality of the resulting surface. Moreover, the complexity of the algorithm and its actual running time are not increased.

The algorithm makes no prior assumption about the input. It operates on any kind and number of contours, and handles all branching situations and hierarchical structures. It avoids many of the problems mentioned above. The surfaces generated are geometrically acceptable, they do not have gaps, and they do not seem deformed in any way.

In a nutshell, the original algorithm is composed of two steps. The first step matches *similar* portions of contour between every pair of adjacent slices and tiles each pair of matched subcontours. Each pair of matched contours is tiled by a sequence of adjacent triangles which form a "band" between the contours. The second step triangulates the remaining *clefts* between contours. Clefts are three-dimensional polygons that consist of unmatched portions of contours. The cleft triangulation attempts to optimize some objective function.

Since the first step of the algorithm matches similar portions of the contours, we expect the resulting interpolation of this step to be smooth. The second step (cleft triangulation), however, matches the non-similar parts of the contours. The resulting interpolation may therefore exhibit abrupt changes. It is thus the triangulation step that we aim to improve. We introduce a new approach for considering the adjacent layers while triangulating the clefts of the current layer, which we call *multi-level sensitive* cleft triangulation. Our experiments show that information produced by the already interpolated surface patches in the vicinity of a cleft can be used to efficiently triangulating it so that it connects smoothly with the surrounding surface.

In addition, we study various objective functions for cleft triangulation. We compare the quality of the resulting triangulations by using various measures. We note, however, that there is no unique solution to the interpolation problem, and that a measure of 'goodness' of a proposed solution is rather subjective and intuitive.

The paper is organized as follows. In the next section we state the problem and make the necessary definitions. In Section 3 we describe the algorithm and in particular, the method of multi-level sensitive cleft triangulation. In Section 4 we present several objective functions for optimizing a triangulation of a cleft and compare the resulting surfaces. In Section 5 we extend the algorithm to avoid self-intersections of the interpolated surface. In Section 6 we briefly analyze the complexity of the algorithm and present some experimental results. We terminate in Section 7 with some concluding remarks. This paper is an extended version of (Barequet, Shapiro, and Tal 1996).

2 Statement of the Problem

The input to the algorithm is a sequence of n parallel planar slices. The slices are assumed to be parallel to the xy-plane, and to lie at heights $z = z_1, \ldots, z_n$. Each slice consists of a list of closed simple polygonal contours, which do not intersect each other (but they can be nested within each other). The goal is to construct a polyhedral solid, bounded by an orientable manifold, which is contained in $z_1 \leq z \leq z_n$. We require that the intersection of the resulting surface with the plane $z = z_i$ coincides with the material regions of slice i (for $i \leq i \leq n$). Moreover, the surface is required to not intersect itself.

We first define three-dimensional polygons and their triangulations.

Definition 1 (Three-dimensional polygon) *A* three-dimensional polygon *is an unknotted closed polygonal chain (a topological circle) in three dimensions.*

Note that a three-dimensional polygon need not be planar. In case there exists a plane onto which the orthogonal projection of the polygon is simple (that is, no two edges of the projected polygon intersect), we say that the polygon has a simple xy-projection.

Definition 2 (Three-dimensional polygon triangulation) *A* three-dimensional polygon triangulation *of a three-dimensional polygon P is an orientable piecewise-linear simply-connected surface (a topological disk) whose vertices are the vertices of P, whose faces are triangular, and whose boundary is P.*

We typically require that the triangulation do not intersect itself, that is, that no two triangles intersect in their interiors.

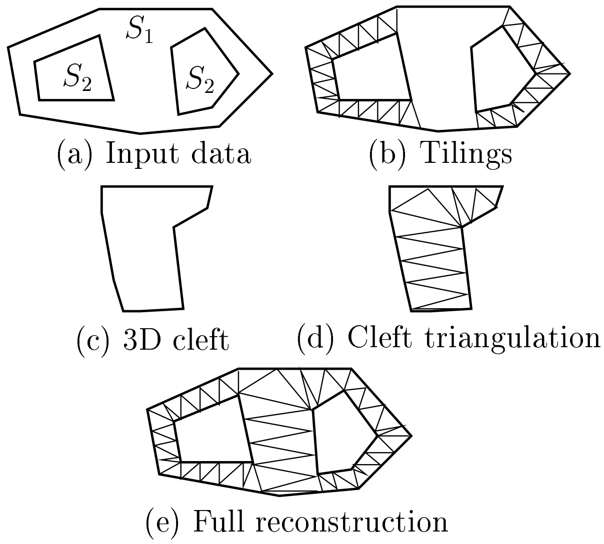


Figure 1: Overview of the basic algorithm

3 The Reconstruction Algorithm

In this section we present the reconstruction algorithm. The algorithm is based on that of (Barequet and Sharir 1996). We improve its second stage so that it uses data of neighboring layers during the interpolation process. In other words, data gathered from the portions of the object that have already been interpolated influence the further surface construction. While in (Barequet and Sharir 1996) each layer was interpolated separately, we first invoke the tiling phase (see below) for *all* the layers, and only then perform the triangulation of all the remaining clefts (in all the layers).

As mentioned before, the algorithm consists of two main phases: *contour matching and tiling* and *cleft triangulation*. The first phase matches similar contour portions of each pair of adjacent slices and tiles the matched contour portions by a sequence of adjacent triangles. The matching and tiling are based solely on the similarity between the xy-projections of the portions of two contours; no other contours in the same or other slices affect this step. During this phase, information from adjacent layers is not being used. This is so because we expect the connection between similar portions of contours to be smooth. During the second phase, the binary sum of the original polygons and the tiling triangles is considered. (In other words, we cancel out every edge that appear twice, in opposite directions, in the union of the original polygons and the new triangles.) The result is a collection of three-dimensional polygons, so-called *clefts*, which are now triangulated. During this phase we use data from neighboring layers. Sharp corners in the reconstructed surface can occur when such data is not used.

A schematic description of the algorithm is shown in Fig. 1, where the various steps of the algorithm are applied to a typical branching case.

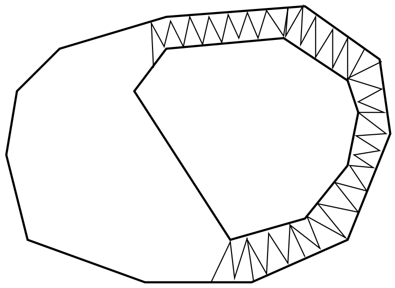


Figure 2: Tiling similar portions of contours

3.1 Contour Matching and Tiling

Matching similar contour portions is performed by using a *partial curve matching* technique, which was first suggested in (Kalvin et al. 1986; Schwartz and Sharir 1987). This technique uses the *geometric hashing* method. We proceed in four steps (the reader is referred to (Barequet and Sharir 1996) for more details).

Step 1: Discretization. The edges of the contours are first discretized so that the arclength between each pair of consecutive vertices is equal to a given parameter.

Step 2: Voting. Each vertex is attributed by a *footprint* which is invariant under projection (of one slice onto the other). In our case the footprint is the xy-coordinates of the vertex. Pairs of vertices, one of each slice, with sufficiently close footprints (i.e., vertices which are geometrically close when considering their xy-projections) are located. We say that the vertices "vote" for each other. Note that a vertex of one slice can vote for more than one vertex of the other slice.

Step 3: Matching. The voting scheme of Step 2 is used for computing the alignment of contours, that admits a sufficiently large fit between them. Two subcontours, one of each slice, are said to match each other, if, for some chosen parameter $\varepsilon > 0$, there are sufficiently many pairs of vertices at the same relative location ("shift") along the subcontours, such that the differences between the corresponding footprints do not exceed ε. Such a match is considered "good" if the number of votes for the specific shift is large, the number of mismatches is small, and the vertices are "close" to each other. An important property that can be achieved by choosing good matching parameters is that all the intersections of the xy-projections of the contours belong to matching subcontours.

Step 4: Tiling. Each pair of matching subcontours is tiled by a sequence of adjacent triangles which form a "band" between the portions. Fig. 2 shows a tiling between similar portions of contours. The "stitching" is performed in a greedy fashion. At any point we have two lists of vertices of the two subcontours. There are two options of selecting the next triangle. We can either add a vertex from the upper slice or from the lower slice. The rule that we implemented minimizes the perimeter of the newly-created triangle. Other advancing rules can be applied as well.

3.2 Cleft Triangulation

At this stage all the matching subcontours of all the layers have already been tiled. For each layer, the binary sum of the original polygons and the new triangles is computed and duplicate edges are canceled out. The result is a collection of spatial polygons, the *clefts*, which are composed of pieces of contours of both slices and of edges of the connecting triangles (see Fig. 1(c)). Recall that the matching step guarantees that the xy-projections of the clefts are simple and pairwise disjoint (although they may be nested within each other). If nesting occurs within a certain slice, the containment hierarchy is simplified by adding *bridges* (pairs of oppositely oriented segments) between the clefts, using a minimum-spanning-tree heuristic. These bridges do not conflict with the geometries of the clefts since their xy-projections do not intersect.

Our next goal is to triangulate the clefts at all layers. Moreover, we wish to interpolate the current layer "consistently" with the way its neighboring layers have been interpolated. Here "consistent" means that we attempt to avoid the creation of sharp dihedral angles between triangles of the interpolated surface. Improving the cleft triangulation is of major concern since this is where we interpolate a surface which connects between non-similar parts of the layers. We are able to employ not only the data produced during the tiling phase, but also the the data created during the cleft triangulation of previous levels (since triangulation is done in ascending order of the layers).

In order to do that we maintain the normal vectors (hereafter called "normals") of the created triangles (generated either during tiling or during previous triangulations). Each of the three edges of an existing triangle is associated with the triangle's normal. We consider this as a "recommendation" for the normals of the adjacent triangles which we are about to build.

Now, when we attempt to triangulate a particular cleft, some of its edges may already have a "recommended" direction, while others may not. If none of the edges of a candidate triangle is attributed with such recommendation, the triangle weight is set to the value of the particular objective function we try to optimize (e.g., area). However, if some of the edges are associated with recommendations of normal directions, then the weight of the candidate triangle is also affected by the direction of its normal. A triangle receives a smaller (i.e., better) weight if the direction of its normal is more consistent with the "recommended" direction (that is, when the angle between them is small). To find the weight of a candidate triangle, its original weight (without considering the normals) is multiplied by a factor which is a function of the angle between the recommended normal and the actual normal. In case more than one edge is attributed with a previous normal vector, the average of these directions is taken. Thus we favor triangles whose normals agree with the directions of their neighboring triangles. Full consistency (resp., inconsistency) occurs when the angle is $0°$ (resp., $180°$).

The function of the dihedral angle θ which we use to determine the above factor is $f(\theta) = \frac{14\theta}{\pi} - \frac{4\theta^2}{\pi^2}$. This function was chosen so that it would favor small angles. Moreover, it gives a much higher "bonus" to small angles (deviations between normals) since it decreases in a non-linear rate. In particular, we selected the parabola that satisfies the constraints $f(0) = 0$, $f(\frac{\pi}{2}) = 6$, and $f(\pi) = 10$. We experimented with various functions and found out that this function works very well in practice.

We next illustrate how knowledge of already-interpolated pieces of the surface may help in preventing sharp angles between facets in successive layers that share the same edge. Fig. 3(a) shows a synthetic example in which a cleft is about to be triangulated. This example was designed so that the same cleft appears with two different environments, and each environment results in a different triangulation of the cleft. Figs. 3(b,c) show the two triangulations of the cleft, while Figs. 3(d,e) show the environments of these triangulations. The reconstructions of the neighboring layers are shown in blue, the matches of the current (middle) layer are shown in green, and the cleft triangulations are shown in red. Note that the different cleft triangulations are indeed connected smoothly with their environments.

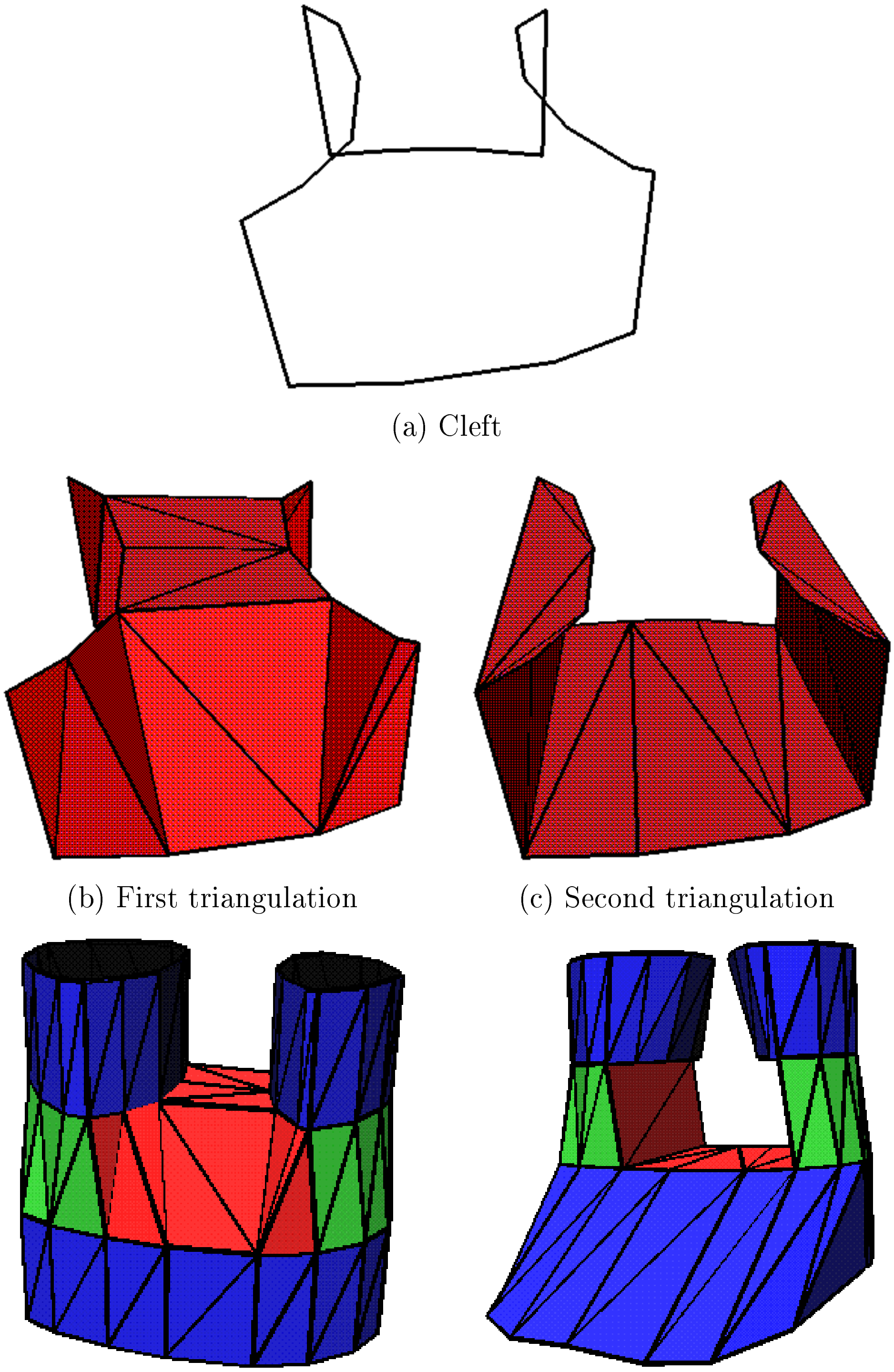
We demonstrate in Section 6 how our algorithm works on real data.

4 Optimizing 3D-Polygon Triangulation

Theoretically, a cleft may have no non-self-intersecting triangulation. It is shown in (Barequet et al. 1998) that the decision problem of whether a three-dimensional polygon is *triangulable* (that is, it has a non-self-intersecting triangulation) is $\mathcal{NP}$-complete. However, a three-dimensional polygon with a simple xy-projection (which is our case) is always triangulable (ibid.).

Assume that a cleft consists of n vertices numbered from 0 to $n-1$. We use a dynamic-programming approach for cleft triangulation. In this approach we find the best vertex i (for $1 \leq i \leq n-2$) for which the total weight (objective function) of the triangle $(0, i, n)$ and the optimum triangulations of the subpolygons $(0, \ldots, i)$ and $(i, \ldots, n-1)$ is optimized. See (Barequet and Sharir 1996, Section 6.2) for details. This technique optimizes the given objective function but does not guarantee a non-self-intersecting solution. In practice, however, dynamic programming works extremely well. It produced non-self-intersecting triangulations for all the clefts we experimented with.

We define four objective functions to be optimized (three are minimized and one



(a) Cleft

(b) First triangulation

(c) Second triangulation

(d) First context

(e) Second context

Figure 3: A synthetic example

is maximized). Each function is a mapping of every triangle to some positive real number. Each triangulation of an n-gon consists of exactly $n-2$ triangles and $n-3$ internal edges, so we measure in all cases the total sum of the function over the triangulation and omit the average. Here are descriptions of the objective functions:

1. **Area** (minimized). This objective function obtains a minimum-area triangulation. The intuition behind choosing this function is that we expect a non-self-intersecting triangulation of a three-dimensional polygon with a simple xy-projection to have a small area.

2. **Area**^{2} (squared area, minimized). This function prefers more substantially small triangles over large ones. It thus “punishes” large triangles more severely than the previous measure, since the growth in their weight is squared. This function has two advantages over the previous function. It is computed more efficiently, and it favors regular triangulations (those that contain triangles of similar areas) over non-regular triangulations. Its drawback is that it does not necessarily yield the minimum-area triangulation.

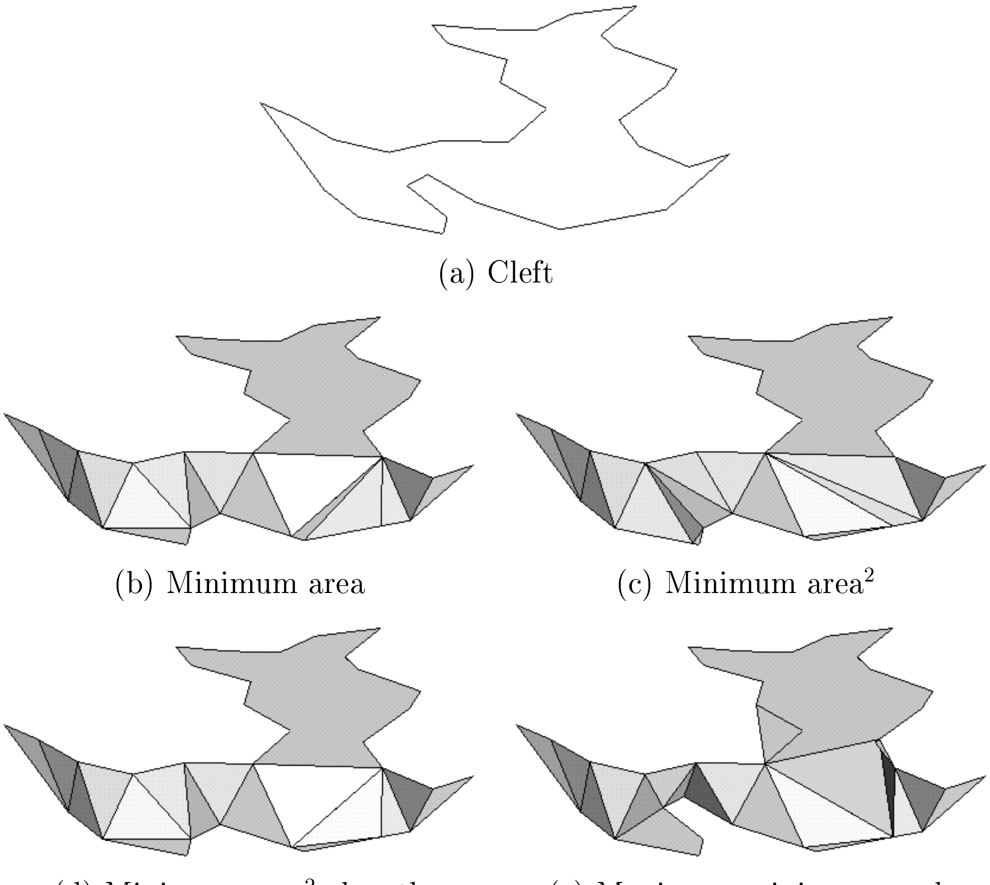
3. **Area**^{2} + **edge lengths** (minimized). This objective function not only selects small triangles, but also discourages long edges. The intuition behind this function is to avoid long skinny triangles (by considering the edge lengths) while aiming to avoid self-intersections (by considering the areas).

4. **Minimum angle** (maximized). Here we consider the minimum angle of each triangle and attempt to maximize the sum of these minima over the triangles of the triangulation. This objective function also aims at avoiding long skinny triangles by attempting to choose triangles whose minimum angle is maximal.

Fig. 4(a) shows a cleft with 37 vertices, while Figs. 4(b–e) show its triangulations produced by using the four objective functions. Coplanar neighboring triangles were united for better visualization of the reconstructed surfaces. Notice the “beak effect” in Fig. 4(e): since the minimum-angle objective function does not consider area at all, it chose to cover the upper slice with overlapping triangles that made the smaller angles as large as possible.

There exists no single criterion for measuring the quality of a triangulation, and thus of the quality of the reconstructed surface. A good triangulation should not intersect itself, but it should also be smooth, for certain applications it should not contain long skinny triangles, it should avoid “waves”, etc. We compare the mentioned-above objective functions by using the following criteria:

1. **Area**. The sum of the areas of all the triangles. It measures the total surface area of the particular triangulation. Often, a minimum-surface triangulation is considered the “best” guess of an unknown surface spanned by an input three-dimensional polygon.



(a) Cleft

(b) Minimum area

(c) Minimum area^2

(d) Minimum area^2+length

(e) Maximum minimum-angle

Figure 4: Different triangulations of a 37-edge polygon

2. **Average dihedral angle**. The average of the dihedral angles between neighboring triangles. In a sense, it gives an estimation to the degree of "smoothness" of the surface.

3. **Average ratio of edges**. The average ratio of the longest edge to the shortest edge of the triangles. It indicates the existence of undesired long skinny triangles. (For a perfect equilateral triangulation this measure is 1.)

4. **Maximum ratio of edges**. The maximum ratio of the longest edge to the shortest edge of the triangles. Unlike the previous measure, this one indicates the existence of a single long skinny triangle.

5. **Number of inversions**. A triangle forms an *inversion* when two of its edges are internal to the triangulation and the dihedral angle of one of them is more than π whereas that of the other edge is less than π. It measures the degree to which the surface is rippled.

6. **Number of intersections**. The number of intersections between triangles of the triangulation. Self-intersecting surfaces are unacceptable for most applications. This measure thus indicates the validity of the triangulation.

Table 1 summarizes our findings for a sample of eleven three-dimensional polygons. It demonstrates the tradeoffs between the various objective functions. The "Number of intersections" column was omitted since we did not encounter any intersections in our experiments. The chosen polygons were produced during the reconstruction of a pair of human lungs and a hip bone. They are typical to the kind of polygons produced in the course of this interpolation algorithm. Fig. 5 shows three of these polygons; the axis-parallel bounding boxes were included in order to demonstrate the nonplanarity of the polygons. Following is the analysis of the results.

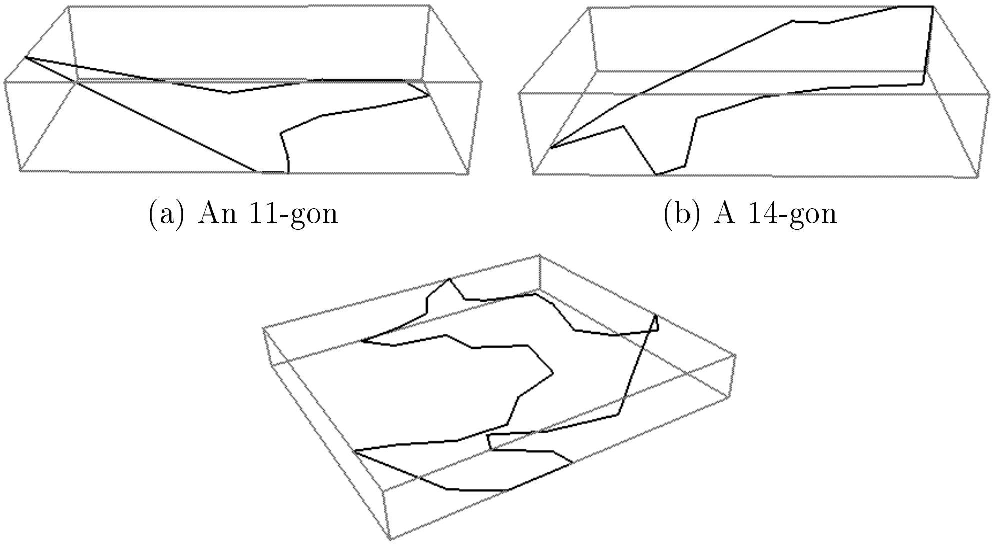
To minimize the area, the natural objective function is *area*. However, $area^2$ and $area^2+length$ produce small areas as well. In fact, $area^2+length$ works consistently better than $area^2$ and almost as good as *area*. This is a pleasant result since $area^2+length$ also avoids long skinny triangles.

The average dihedral angle indicates the "smoothness" of the surface. The two objective functions that produce the best results in this regard are *area* and $area^2+length$. The objective function $area^2$ also performs well because it takes into account the area of a triangle. The function *minimum angle* lags behind the other functions. This is anticipated since the regularity of the triangulation seems to be unrelated to its smoothness.

The average ratio of edges and its maximum point to the existence of long skinny triangles. The objective function *minimum angle* is the best in this regard. This is not surprising because the goal of this function is to avoid these undesired triangles. The

Polygon Complexity	Method	Area	Dihedral Angle	Av. Edge Ratio	Max. Edge Ratio	Number of Inversions
11	Area	0.0301	176.948	3.583	6.589	0
	Area^2	0.0315	158.796	4.777	9.822	3
	Area^2+Len.	0.0301	172.071	1.834	3.393	2
	Min. Angle	0.0319	131.948	1.549	2.222	0
14	Area	0.0329	153.165	2.273	5.398	6
	Area^2	0.0329	153.165	2.008	3.633	6
	Area^2+Len.	0.0329	153.165	2.008	3.633	6
	Min. Angle	0.0475	121.115	1.911	5.398	3
15	Area	0.0353	173.467	2.611	4.342	0
	Area^2	0.0407	152.314	5.273	8.386	1
	Area^2+Len.	0.0353	173.467	1.709	1.842	0
	Min. Angle	0.0353	173.467	1.709	1.842	0
24	Area	0.0707	167.011	2.865	15.526	3
	Area^2	0.0724	154.760	3.366	22.051	5
	Area^2+Len.	0.0707	167.011	2.168	7.511	3
	Min. Angle	0.0730	161.239	2.049	7.511	2
27	Area	0.1739	172.203	3.260	5.645	1
	Area^2	0.1739	172.029	4.744	5.645	3
	Area^2+Len.	0.1739	172.029	4.744	5.645	3
	Min. Angle	0.1756	148.574	2.157	2.729	2
33	Area	0.4048	137.534	3.365	5.080	3
	Area^2	0.4089	140.729	3.711	5.284	4
	Area^2+Len.	0.4089	146.349	3.709	5.222	2
	Min. Angle	0.4697	128.688	2.194	5.080	1
37	Area	0.1265	159.910	4.603	67.689	7
	Area^2	0.1271	158.393	3.515	37.689	8
	Area^2+Len.	0.1266	154.003	2.668	17.763	7
	Min. Angle	0.1397	139.450	2.492	17.763	12
48	Area	0.2208	173.028	4.100	5.184	6
	Area^2	0.2215	173.635	4.942	6.014	8
	Area^2+Len.	0.2211	171.955	4.376	5.184	6
	Min. Angle	0.2324	149.436	2.244	4.830	4
54	Area	0.1428	158.740	1.972	3.724	10
	Area^2	0.1443	152.965	2.087	3.729	11
	Area^2+Len.	0.1428	158.740	1.906	3.724	10
	Min. Angle	0.1496	147.822	1.807	3.724	10
55	Area	0.1406	155.126	2.370	7.376	12
	Area^2	0.1413	156.804	2.489	6.241	11
	Area^2+Len.	0.1406	157.186	1.903	5.406	10
	Min. Angle	0.1578	150.983	1.611	5.223	12
72	Area	0.2100	147.143	2.226	6.672	21
	Area^2	0.2116	144.603	2.425	12.576	25
	Area^2+Len.	0.2103	149.373	1.898	6.672	20
	Min. Angle	0.2453	134.138	1.841	6.672	23

Table 1: Comparison of heuristics



(a) An 11-gon (b) A 14-gon

(c) A 55-gon

Figure 5: Typical 3-dimensional polygons

objective function *area*^{2}+*length* also produces nice results due to its edge-length component. Both *area* and *area*^{2} may create (as observed in our experiments) undesired thin triangles both on average and in the worst case.

The number of inversions indicates the "rippleness" of the surface. We found no consistency in the results. For example, *minimum angle* is the best for the 14-gon and the worst for the 37-gon. One possible explanation for the inconsistency of the results is that natural data are typically wavy.

Finally, no self-intersecting triangulations were generated in any of the examples we experimented with. This may be so because our algorithm guarantees that the xy-projections of the three-dimensional polygons are always simple, and therefore also triangulable. Note that theoretically it is not guaranteed that our algorithm always finds a non-self-intersecting triangulation.

Our conclusion is that *area*^{2}+*length* is the objective function of choice. It works extremely well as far as the dihedral angle is considered. It comes second for the area criterion, where only the *area* objective function obviously performs better. It also comes second for the ratio of edges, where only *minimum angle* is better. In general, it exhibits a good behavior for all the criteria we considered.

5 Avoiding Self-Intersections

A major consideration in the reconstruction of surfaces is preventing self-intersections. In this section we present a variant of our algorithm which avoids self-intersections in the cleft triangulation.

As noted in Section 4, a sufficient condition for a 3-dimensional polygon to have a non-self-intersecting triangulation is to have a simple 2-dimensional orthographic projection in some direction (Barequet et al. 1998). Given such a direction d, it is straightforward to compute a non-self-intersecting triangulation: Simply project the polygon into a plane perpendicular to d, then triangulate the corresponding planar polygon, and finally lift the triangulation back to three dimensions. It is guaranteed that the lifted triangles do not intersect in their interiors.

The clefts which need to be triangulated during the second step of the interpolation algorithm are known to have simple xy-projections. This is so because all the intersections of the xy-projections of edges of the original contours have already been detected and handled during the first step of the algorithm (Barequet and Sharir 1996). This suggests an improved algorithm: project the cleft to the xy-plane, triangulate the projected cleft, and lift the triangulation back to three dimensions. This approach, however, will not allow us to make use of the objective functions defined on the spatial triangles, as described in Section 4. Instead, we carry out the cleft triangulation in three dimensions, but in a way that ensures that the xy-projection of the triangulation is simple.

Assume that we triangulate a cleft with h vertices. We add to the triangulation procedure a preprocessing step that computes an $h \times h$ matrix $\mathcal{F}$ of "forbidden" diagonals. The entry $\mathcal{F}_{ij}$ (for $1 \leq i < j \leq h$) indicates whether the xy-projection of the diagonal between the ith and the jth vertices of the projected cleft intersects with the xy-projection of any of the h cleft edges. We compute this matrix in a brute-force manner by comparing every diagonal with every edge. (As we show in the next section, this does not change the asymptotic running time of the algorithm.)

Recall that during the course of the dynamic-programming triangulation we assign to each triangle the cost of some objective function. We modify this step by taking into account the forbidden diagonals. If a triangle contains at least one diagonal between vertices i and j of the polygon, and $\mathcal{F}_{ij}$ indicates that this diagonal is forbidden, then the weight of this triangle is set to $+\infty$ or $-\infty$ (depending on whether we minimize or maximize the objective function). This way we rule out any triangulation that includes forbidden diagonals.

Thus, the matrix $\mathcal{F}$ is used for rejecting diagonals whose xy-projections intersect with the xy-projections of cleft edges. The following theorem guarantees that we do not have to check whether the xy-projections of two diagonals of a triangulation intersect.

Theorem 1 *The dynamic-programming triangulation algorithm that makes use of $\mathcal{F}$ prohibits xy-intersections of diagonals.*

Proof: Assume to the contrary that the triangulation obtained by the algorithm

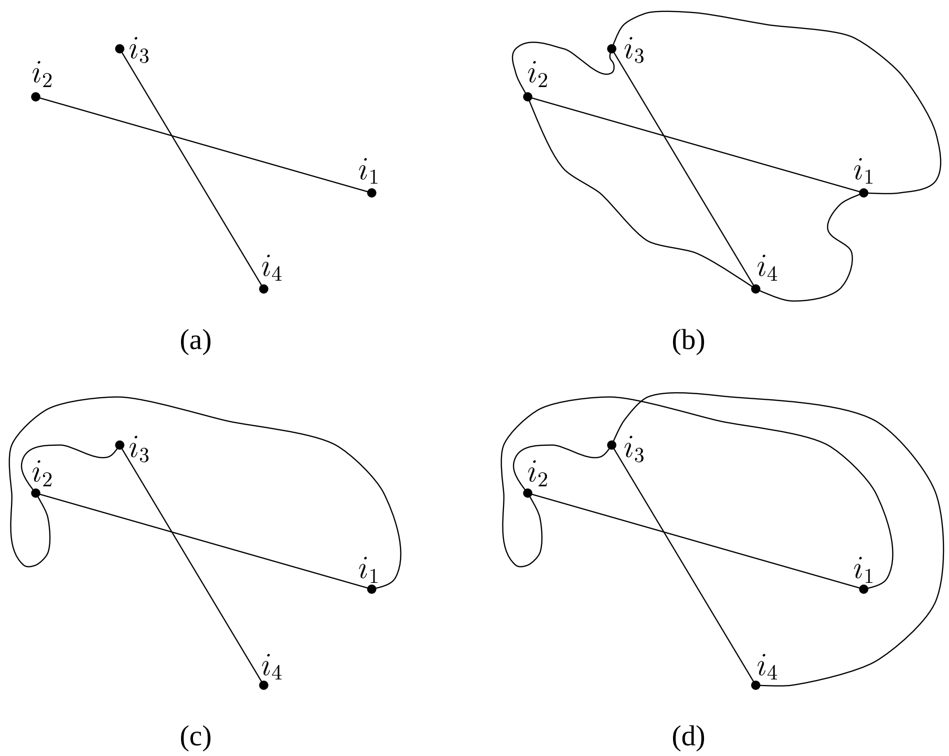


Figure 6: Intersecting diagonals

contains two diagonals whose xy-projections intersect. Denote the indices of the diagonal endpoints by (i_1, i_2) and (i_3, i_4) (see Fig. 6(a)). Alternating the ordering of the indices (e.g., $i_1 < i_3 < i_2 < i_4$) (Fig. 6(b)) is not possible since a solution of a dynamic-programming problem cannot simultaneously have such diagonals due to the separability of the problem. Assume then, without loss of generality, that $i_1 < i_2 < i_3 < i_4$. Now draw the cleft. The cleft portions from i_1 to i_2 and from i_2 to i_3 can be drawn easily (Fig. 6(c)). The portion from i_3 to i_4 cannot intersect the diagonal (i_1, i_2) (since we use $\mathcal{F}$), hence it must intersect the cleft portion from i_1 to i_2 (Fig. 6(d)). This is because the diagonal (i_1, i_2) and the cleft portion from i_1 to i_2 form a closed Jordan curve J, where i_3 and i_4 are inside and outside J, respectively. This is a contradiction to the assumption that the cleft has a simple xy-projection. Therefore no two diagonals with intersecting xy-projections are chosen by the algorithm that uses $\mathcal{F}$. $\square$

Note that the existence of a triangulation which does not make use of any forbidden diagonal is provided by the fact that the 3-dimensional polygon has a simple xy-projection, which can always be triangulated in the plane (Lennes 1911, proposition 30). Such a triangulation is guaranteed to be found by the dynamic-programming algorithm, since in the worst case it checks all the possible triangulations of a polygon.

To recap, the improved cleft-triangulation algorithm finds the optimum triangulation (according to some objective function) among all the triangulations with simple

xy-projections. Thus, the triangulation itself is guaranteed not to intersect itself.

We would like to note that in practice, it is often the case that the improved algorithm finds the same triangulation as the original algorithm. In particular, we found that this happens when the triangle area is one of the elements that define the objective function, such as *area*, *area*^{2}, or *area*^{2}*+length*. In these cases the produced triangulations were often xy-triangulations lifted to three dimensions.

6 Implementation and Experimental Results

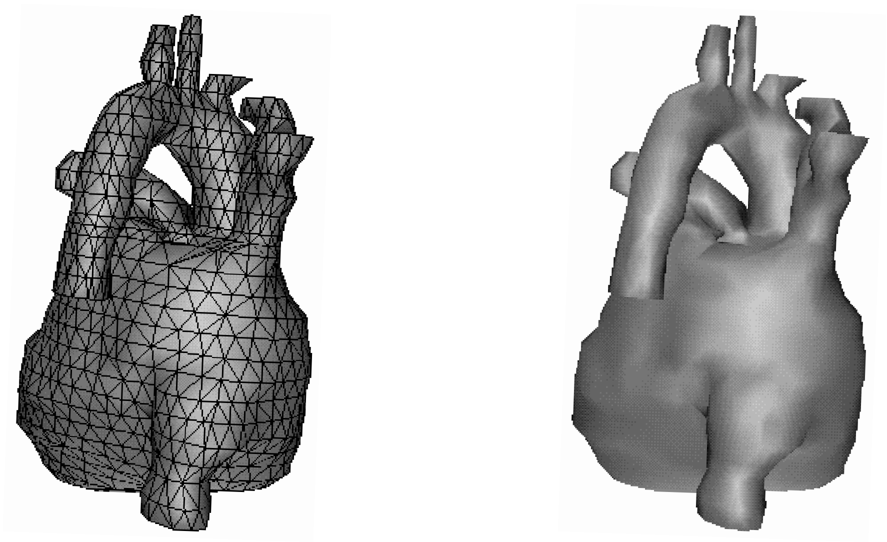
The algorithm was implemented in C on a Silicon Graphics Indy workstation. To guide our search for good heuristics and to visualize the results, we used the geometric animation system GASP (Tal and Dobkin 1995) and the visualization package Geomview (Amenta et al. 1995).

A formal analysis of the complexity of the algorithm is found in (Barequet and Sharir 1996). The original algorithm runs on practical instances in average $O((k+\xi)\log(k+\xi)+n+\xi h^3)$ time, where k is the total number of input vertices along the contours of two consecutive slices, n is the number of points after the discretization step, ξ is the number of clefts, and h is the size of the biggest cleft. Avoiding self-intersections by adding a preprocessing step before the triangulation of each cleft, as described in the previous section, requires $O(h^3)$ time per cleft, thus $O(\xi h^3)$ time in total, which does not change the asymptotic running time.

The last term of the expected running time of the algorithm ($O(\xi h^3)$) is the cost of the cleft triangulations. (The dynamic-programming triangulation runs in time cubic in the size of a cleft.) This is, theoretically, and in most situations also pragmatically, the most expensive step of the algorithm. Our method modifies this step but does not change its running time. Indeed, by using a hashing table, we can store all the "recommendation" data in expected $O(n)$ time. Fetching one such recommendation can be done in expected constant time, so computing the weight of a triangle can also be performed in constant time. Hence the optimal triangulation of a cleft can still be computed in $O(h^3)$ time. The total expected running time thus remains as specified above.

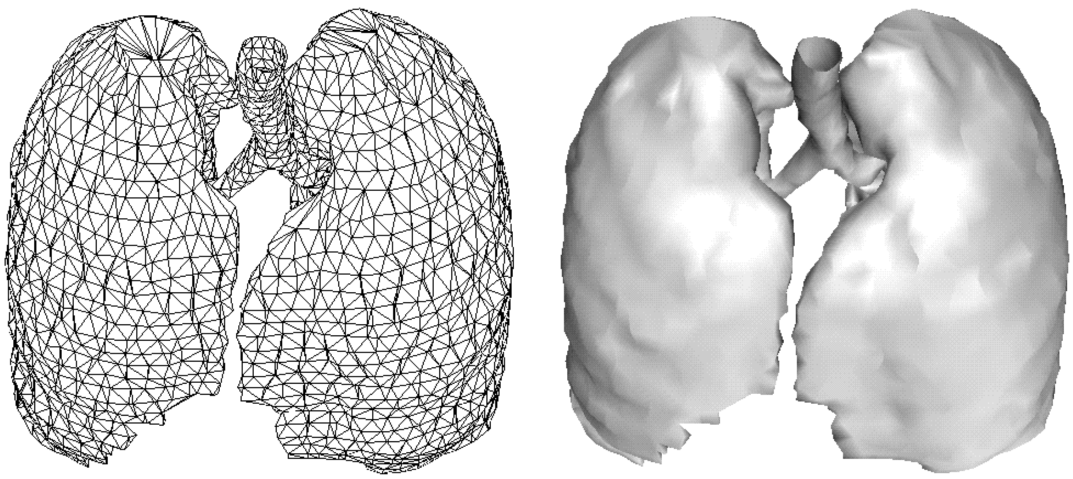
We reconstructed several human organs,[1] among them a heart, lungs, a pelvis, and a hip bone. Figs. 7(a,b) show a human heart. The input consists of 30 MRI slices, which contain 65 contours (2.2 per slice) and 1285 vertices (19.8 per contour). The slices are 0.5 mm thick. The reconstructed surface consists of 2552 triangles. Both the faceted and the smooth versions are displayed. Figs. 7(c,d) show human lungs. The input consists of 34 MRI slices, which contain 88 contours (2.6 per slice) and 3121 vertices (35.5 per contour). The slices are 0.5 mm thick. The reconstructed surface consists of 8382 triangles. Figs. 8(a,b) show a human pelvis. The input consists of

[1]The data files can be obtained from `ftp://ftp.math.tau.ac.il/pub/barequet/psdb/data` .



(a) Faceted model (b) Smooth model

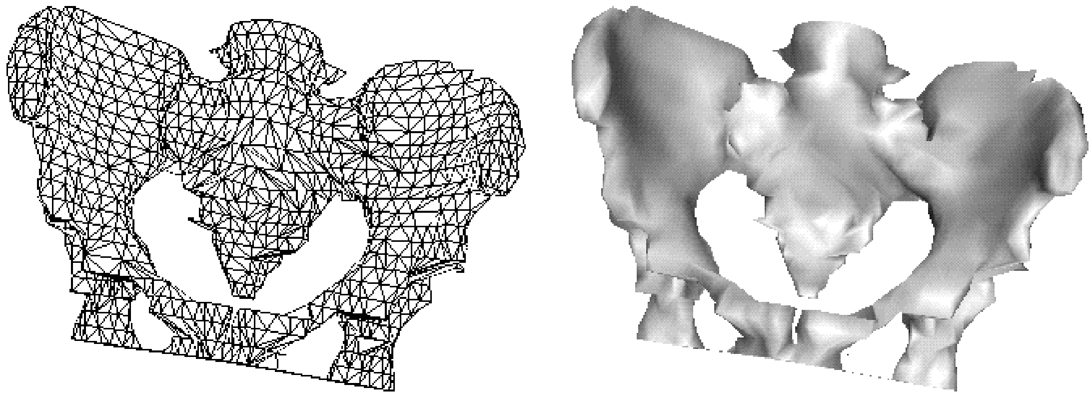
A human heart



(c) Faceted model (d) Smooth model

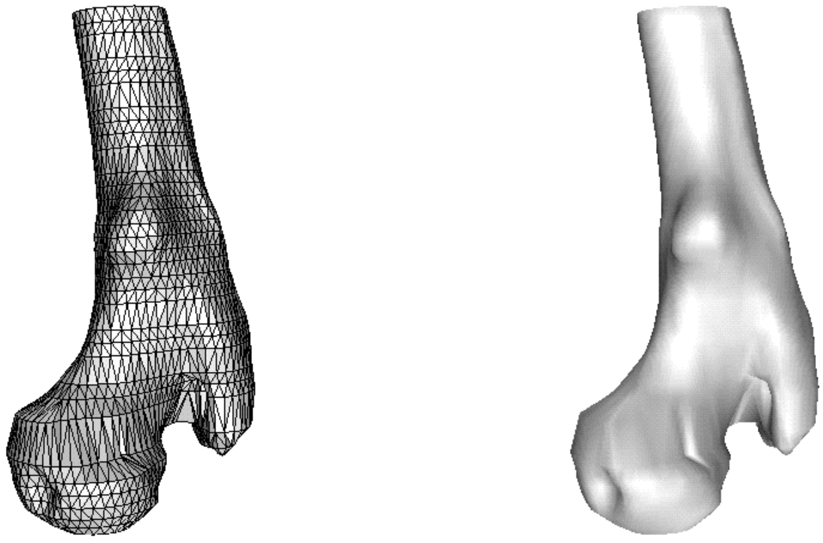
Human lungs

Figure 7: Reconstructions of human organs from MRI scans



(a) Faceted model (b) Smooth model

A human pelvis



(c) Faceted model (d) Smooth model

A human hip bone

Figure 8: Reconstructions of human organs from CT scans

23 CT slices, which contain 68 contours (3.4 per slice) and 1913 vertices (28.1 per contour). The reconstructed surface consists of 3756 triangles. Figs. 8(c,d) show a human hip bone. The input consists of 34 CT slices, which contain 38 contours (1.1 per slice) and 1686 vertices (44.4 per contour). The reconstructed surface consists of 9998 triangles. Fig. 9 shows in red the triangulated clefts in the reconstructions of the heart, pelvis, and lungs. (The tiling triangles are shown in blue.)

Fig. 10 shows a typical branching situation in the pelvis. The original slices and the reconstruction of the corresponding layer are shown in (a) and (b), respectively. The cleft that appears in this layer is shown in (c). (Actually, this instance is made of nested clefts which are simplified by adding bridges). The triangulation of the cleft is shown in (d,e). The simplified cleft contains 55 vertices, so its triangulation consists of 53 triangles.

As another measure of the "quality" of a reconstructed model obtained by our algorithm (in cases where we had the "correct" solution) we estimated the Hausdorff distance between the interpolated object and the correct solution:

Definition 3 *The Hausdorff distance between two point sets A and B is defined as $H(A,B) = \max(h(A,B), h(B,A))$, where $h(A,B) = \max_{a \in A} \min_{b \in B} ||a - b||$.*

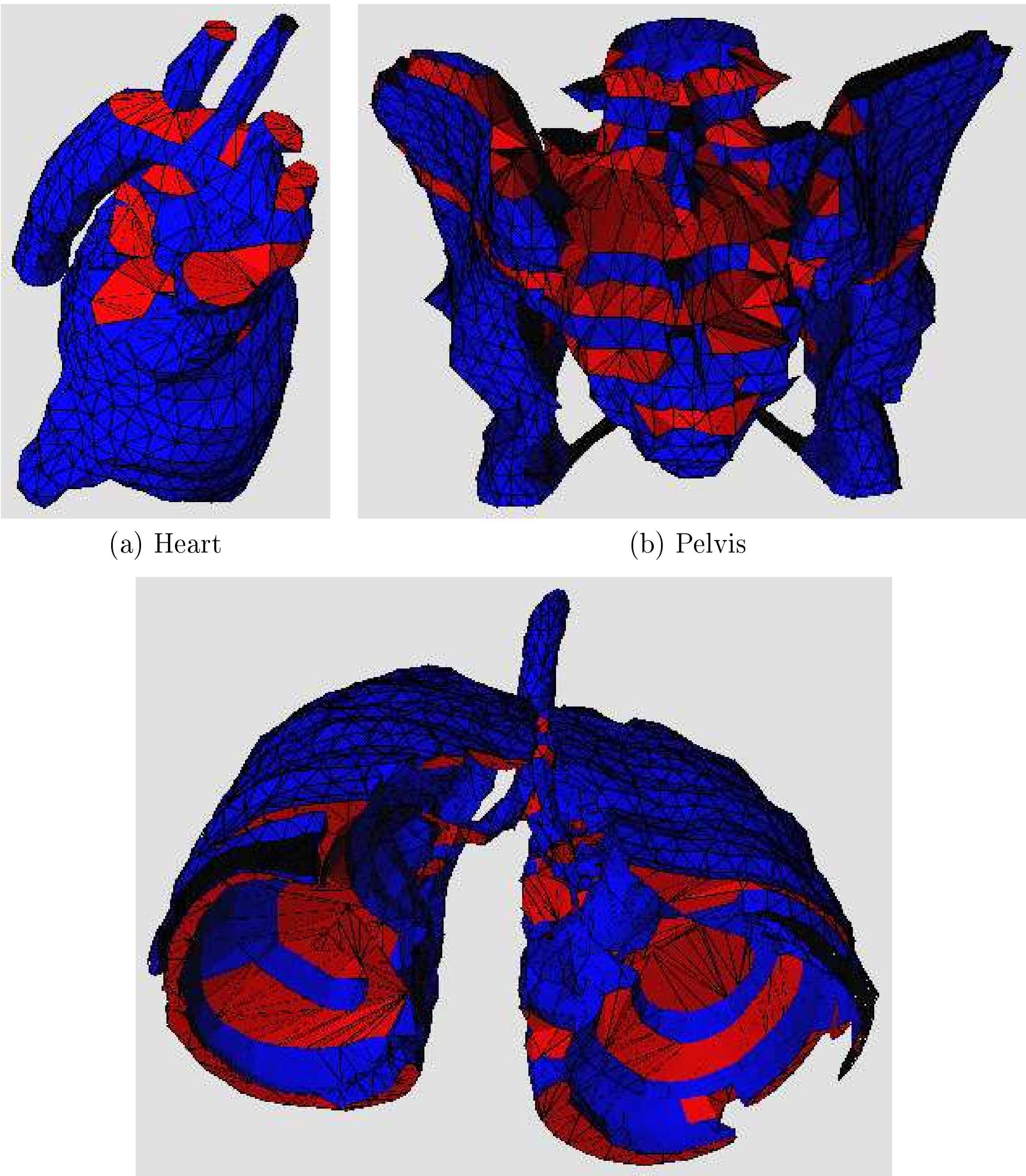
To do this, we first spread points uniformly on the facets of the original object and the reconstructed object, and then compute the Hausdorff distance between the two sets of points.

Consider, for example, the synthetic object shown in Fig. 11(a). The slices of this object are shown in Fig. 11(b) (from a slightly different view point), and the reconstructed object is shown in Figs. 11(c,d). The dimensions of the object are $11.5 \times 11.5 \times 4.3$, and the Hausdorff distance between the point sets representing the two objects is 0.362.

7 Conclusion

This paper investigates a novel method for interpolating a surface between polygonal slices while considering the interpolations of adjacent layers. The algorithm consists of two major steps. First, similar contour portions of every consecutive pair of slices are matched and tiled. We are then left with three-dimensional polygons, so-called clefts. The clefts are triangulated "consistently" with the interpolations in the neighboring layers. We have demonstrated how different environments of the same cleft generate completely different triangulations which match their neighborhoods.

We also address in this paper the issue of selecting an objective function for cleft triangulation. We implemented several optimization functions: minimum *area*, minimum *square of area*, minimum *square of area + the lengths of the edges*, and maximum

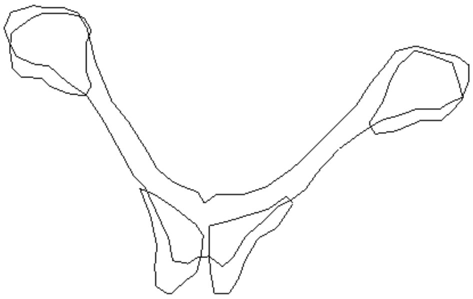


(a) Heart

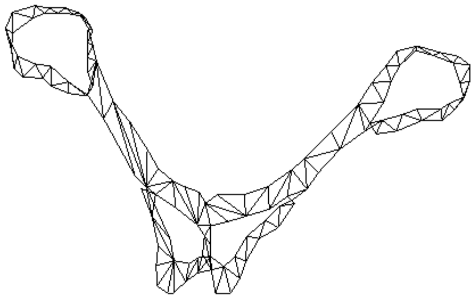
(b) Pelvis

(c) Lungs

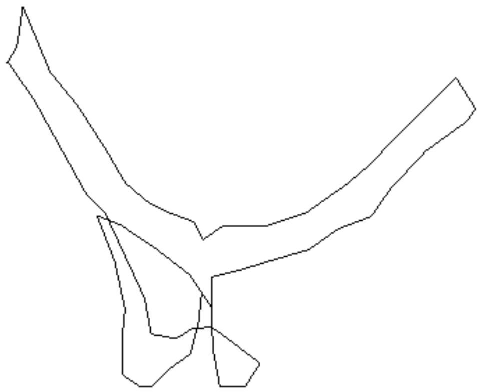
Figure 9: Triangulated clefts in organ reconstructions



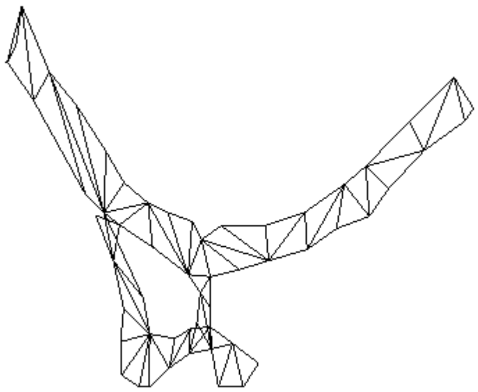
(a) Original slices



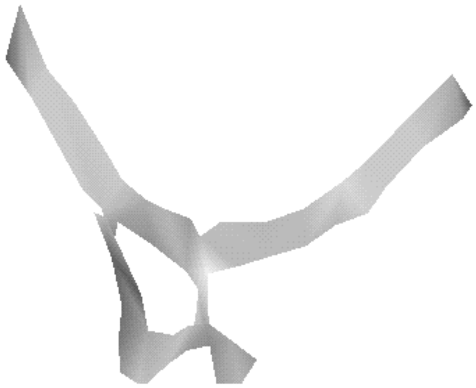
(b) Reconstructed layer



(c) Cleft

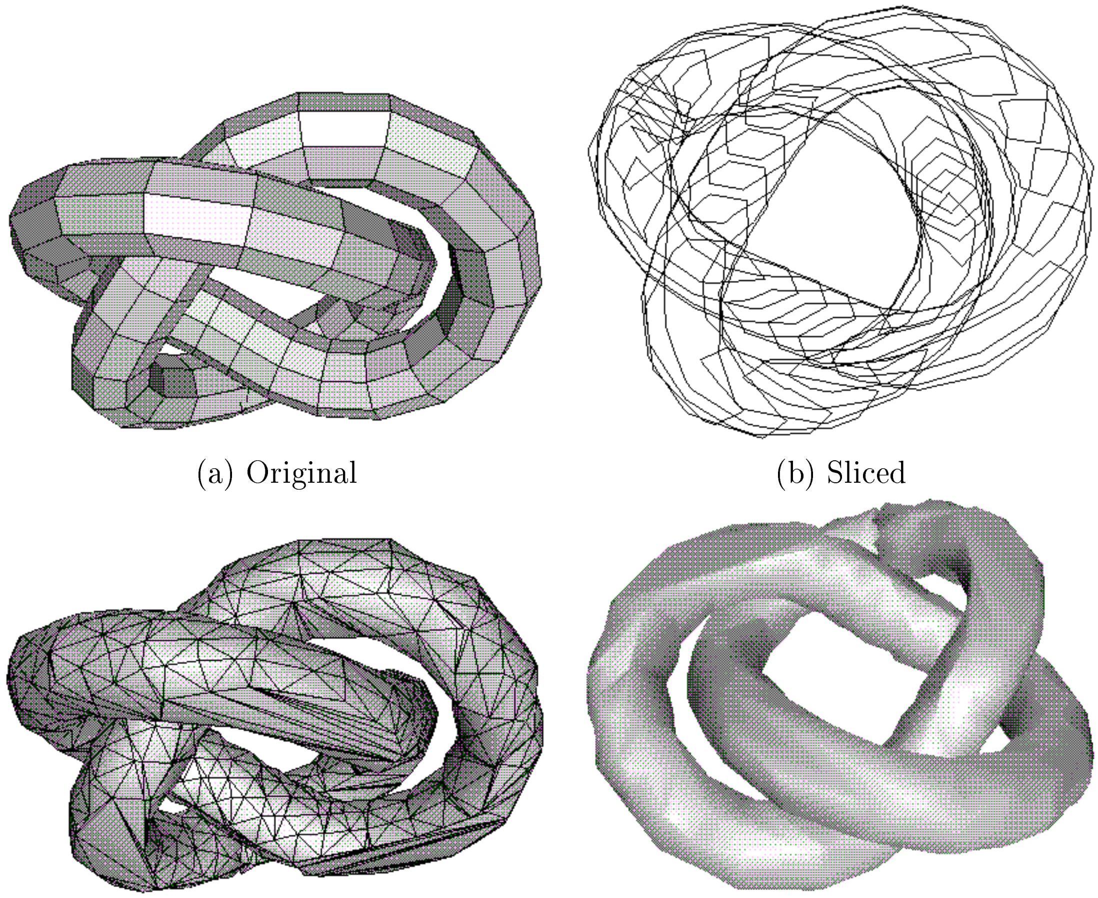


(d) Triangulation (faceted)



(e) Triangulation (smooth)

Figure 10: A typical pelvis layer



(a) Original

(b) Sliced

Faceted

Smooth

(c) Reconstructed

Figure 11: A synthetic object ("trefoil")

minimum-angle. We evaluated the "goodness" of each of the above heuristics by various criteria, and provided experimental results. Our conclusion is that in most cases the *minimum square of area + the lengths of the edges* should be the method of choice.

Our algorithm has been implemented and used for the reconstruction of several human organs, among them a human heart, lungs, a pelvis, and a hip bone. It works very well in practice even for completely different slices with various branching situations, and produces relatively smooth surfaces. It remains to be seen if more complex optimization functions can produce significantly better interpolations.

Acknowledgement

The descriptions of the human organs were contributed by Jean-Daniel Boissonnat and Bernhard Geiger from INRIA at Sophia Antipolis, France. We wish to thank Carole Gitlin and Joseph O'Rourke for their C code for detecting triangle-triangle intersections in 3-space. Work on this paper by G. Barequet has been supported by the Israeli Ministry of Science and the Arts, Eshkol Grants 0562-1-94 and 0562-2-95. Work by D. Shapiro has been supported by the Karyn Kupcinet Fellowship. Work by A. Tal has been supported by the Israeli Ministry of Science and the Arts, Eshkol Grants 0554-1-94 and 0554-2-96.

References

[1] Amenta N, Levy S, Munzner T, Phillips M (1995) Geomview: A system for geometric visualization. Proc. 11th Ann. ACM Symp. on Computational Geometry C12–13

[2] Bajaj CL, Coyle EJ, Lin KN (1996) Arbitrary topology shape reconstruction from planar cross sections. Graphical Models and Image Processing 58:524–543

[3] Barequet G, Dickerson M, Eppstein D (1998) On triangulating 3-dimensional polygons. Computational Geometry: Theory and Applications 10:155–170

[4] Barequet G, Shapiro D, Tal A (1996) History consideration in reconstructing polyhedral surfaces from parallel slices. Proc. Visualization 149–156

[5] Barequet G, Sharir M (1996) Piecewise-linear interpolation between polygonal slices. Computer Vision and Image Understanding 63:251–272

[6] Batnitzky S, Price HI, Cook PN, Cook LT, Dwyer III SJ (1981) Three-dimensional computer reconstruction from surface contours for head CT examinations. J. of Computer Assisted Tomography 5:60–67

[7] Boissonnat JD (1988) Shape reconstruction from planar cross sections. Computer Vision, Graphics and Image Processing 44:1–29

[8] Boissonnat JD, Geiger B (1993) Three dimensional reconstruction of complex shapes based on the Delaunay triangulation. Proc. Biomedical Image Processing and Biomedical Visualization (Acharya RS, Goldof DB, eds.), 1905, SPIE Press, Bellingham, WA, 964–975

[9] Choi YK, Park KH (1994) A heuristic triangulation algorithm for multiple planar contours using an extended double branching procedure. The Visual Computer 10:372–387

[10] Christiansen HN, Sederberg TW (1978) Conversion of complex contour line definitions into polygonal element mosaics. Computer Graphics 13:187–192

[11] Dev P, Fellingham L, Vassiliadis A (1984) 3D graphics for interactive surgical simulation and implant design. Processing and display of 3D data II, 507, Society of photo-optical instrumentation engineers, Bellingham, WA, 52

[12] Drebin RA, Carpenter L, Hanrahan P (1988) Volume rendering. Proc. SIGGRAPH 65–74

[13] Ekoule AB, Peyrin FC, Odet CL (1991) A triangulation algorithm from arbitrary shaped multiple planar contours. ACM Trans. on Graphics 10:182–199

[14] Farrell EJ, Yang WC, Zappulla RA (1985) Animated 3D CT imaging. IEEE Computer Graphics and Applications 5:26–32

[15] Fix JD, Ladner RE (1998) Multiresolution banded refinement to accelerate surface reconstruction from polygons. Proc. 14th Ann. ACM Symp. on Computational Geometry 240–248

[16] Frenkel KA (1989) Volume rendering. Comm. of the ACM 32:426–435

[17] Fuchs H, Kedem ZM, Uselton SP (1977) Optimal surface reconstruction from planar contours. Comm. of the ACM 20:693–702

[18] Ganapathy S, Dennehy TG (1982) A new general triangulation method for planar contours. ACM Trans. on Computer Graphics 16:69–75

[19] Kalvin AD (1992) A survey of algorithms for constructing surfaces from 3D volume data, IBM Technical Report RC-17600

[20] Kalvin A, Schonberg E, Schwartz JT, Sharir M (1986) Two-dimensional, model-based, boundary matching using footprints. Int. J. of Robotics Research 5:38–55

[21] Kaufman A (1991) Volume Visualization, IEEE Computer Society Press

[22] Kehtarnavaz N, de Figueiredo RJP (1988a) A framework for surface reconstruction from 3D contours. Computer Vision, Graphics and Image Processing 42:32–47

[23] Kehtarnavaz N, Simar LR, de Figueiredo RJP (1988b) A syntactic/semantic technique for surface reconstruction from cross-sectional contours. Computer Vision, Graphics and Image Processing 42:399–409

[24] Keppel E (1975) Approximating complex surfaces by triangulation of contour lines. IBM J. of Research and Development 19:2–11

[25] Lennes NJ (1911) Theorems on the simple finite polygon and polyhedron. American J. of Mathematics 33:37–62

[26] Levoy M (1988) Display of surfaces from volume data. IEEE Computer Graphics and Applications 8:29–37

[27] Lorensen WE, Cline HE (1987) Marching cubes: A high resolution 3D surface construction algorithm. Computer Graphics 21:163–169

[28] Meyers D, Skinner S, Sloan K (1992) Surfaces from contours. ACM Trans. on Graphics 11:228–258

[29] Nielson GM, Hamann B (1991) The asymptotic decider: Resolving the ambiguity in marching cubes. Proc. Visualization 83–91.

[30] Raya SP, Udupa JK (1990) Shape-based interpolation of multidimensional objects. IEEE Trans. on Medical Imaging 9:32–42

[31] Schwartz JT, Sharir M (1987) Identification of partially obscured objects in two and three dimensions by matching noisy characteristic curves. Int. J. of Robotics Research 6:29–44

[32] Shantz M (1981) Surface definition for branching contour-defined objects. Computer Graphics 15:242–270

[33] Sloan KR, Painter J (1988) Pessimal guesses may be optimal: A counterintuitive search result. IEEE Trans. on Pattern Analysis and Machine Intelligence 10:949–955

[34] Tal A, Dobkin DP (1995) Visualization of geometric algorithms. IEEE Trans. on Visualization and Computer Graphics 1:194–204

[35] Upson C, Keeler M (1988) V-buffer: Visible volume rendering. Proc. SIGGRAPH 59–64

[36] Vannier MW, Marsh JL, Warren JO (1983) 3D computer graphics for craniofacial surgical planning and evaluation. Computer Graphics 17:263–273

[37] Wang YF, Aggarwal JK (1986) Surface reconstruction and representation of 3D scenes. Pattern Recognition 19:197–207

[38] Welzl E, Wolfers B (1994) Surface reconstruction between simple polygons via angle criteria. J. of Symbolic Computation 17:351–369

[39] Wilhelms J, van Gelder A (1990) Topological considerations in isosurface generation. ACM Computer Graphics 24:79–86

[40] Woolson ST, Fellingham LL, Dev P, Vassiliadis A (1985) 3D imaging of bone from analysis of computer tomography data. Orthopedics 8:1269

[41] Zyda MJ, Jones AR, Hogan PG (1987) Surface construction from planar contours. Computers and Graphics 11:393–408