Improved Constructions of Delaunay Based Contour Surfaces

Siu-Wing Cheng* Tamal K. Dey†

October 21, 1998

Abstract

Surface reconstruction from parallel slices is a well researched problem in solid modeling and computer graphics. The importance of the problem stems from its wide applicability such as in medical imaging for organ modeling, and in topography for terrain modeling. As pointed out in earlier literature, the three major issues for such surface reconstruction are *tiling problem*, *correspondence problem* and *branching problem*. Many of the earlier approaches concentrated primarily on the tiling problem, where the main concern is to generate a non self-intersecting surface connecting two contours with certain optimization. Lately, a few approaches take a more wholistic view to address all of them. In this paper we revisit one such method based on Delaunay triangulations. This method seems more promising and appropriate in handling correspondence and branching problem due to the inherent ability of Delaunay triangulations to capture proximity. Further, a non self-intersecting tiling is automatically guaranteed by these triangulations.

We improve upon the method in two ways. First, using simple tools from computational geometry we eliminate the costly step of computing the three dimensional Delaunay triangulation, and instead compute the surface triangles directly. This saves both space and time, in particular, for large data sets, which is typical for target applications. Secondly, a more accurate postprocessing step is employed to handle the correspondence and branching problem. The decisions for correspondence and branching are made by a simple natural topological assumption about the original object. Our experiment on some medical data shows that the method is effective. Although, in principle, this postprocessing step requires a three dimensional triangulation consisting of tetrahedra, we avoid computing them altogether and still are able to compute only necessary surface triangles.

1 Introduction

We revisit the problem of fitting a triangulated surface between polygonal contours on parallel slices in 3D. The problem is well studied due to its number of applications. For example, this problem arises in medical imaging where human organs are scanned through computed tomography (CT) or magnetic resonance imaging (MRI) techniques to obtain cross-sections and then a reconstruction of a surface connecting these cross-sections is sought. This contour surface models the boundary of the reconstructed organ, which is used for graphics display and in some cases for generating prototypes. Contour surface construction also finds use in terrain modeling from topographic elevation contours, and morphing in computer graphics [12].

The input to the surface reconstruction problem is a series of parallel *slices*, each representing a cross-section of the solid to be modeled. An appropriate edge detection technique is assumed to have detected the boundary of the contours from their raw images. Each slice contains possibly several *contours*, whose boundaries consist of non-intersecting closed polygonal curves. In the case of a contour with hole(s), these polygonal curves are nested.

Given the slices of contours, the goal is to reconstruct a triangulated surface that bounds an object, called *solid connection*, whose geometry is more likely to resemble the original sampled object. This uncertainty in reconstruction is bound to crop up in the absence of any concrete apriori knowledge about the sampled object. Branching and correspondence problem arise from this very uncertainty [1, 14]. However, certain geometry

*Department of Computer Science, HKUST, Hong Kong. email: scheng@cs.ust.hk.

†Department of CSE, IIT Karagpur, India 721302. email: dey@mpi-sb.mpg.de and dey@cse.iitkgp.ernet.in. Research supported in part by a DST grant, 1997, Government of India.

can be ruled out as they are unlikely to occur in practice. For example, in Figure 1 there is a choice about connecting (corresponding) the contour c to contour a alone, or to both a and b above it. A more likely solution to this correspondence problem is Figure 1(c) instead of Figure 1(b). Here the choice is favored by geometric symmetry, which is turn, is dependent on geometric proximity. Even if one decides to correspond c to both a and b, it is not entirely clear how one can or should realize the branching. Branching in Figure 1(c) is more acceptable than the branching in Figure 1(d). Here, the choice is dictated both by proximity and topology. In the neighborhood of a cross section the object boundary is a 2-manifold. In particular, human organs are bounded by smooth surfaces and thus reconstruction should disallow any non-manifold behavior such as in Figure 1(d). Delaunay triangulations, by their definitions, capture geometric proximity automatically. This is the key to the success of the method proposed by Boissonat [4]. Geiger [10] refined the method adding to it a better postprocessing step. We further improve this promising approach in two ways stated under *Results*.

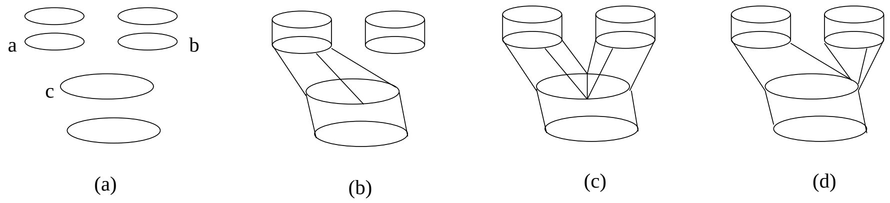

Figure 1: Illustration of correspondence and branching problems

Related Work. Almost all known approaches reconstruct the solid connection between two consecutive slices and then concatenate them in series to model the object. Much of the earlier work assumes a single contour without any hole on each slice, thus concentrating primarily on the tiling problem of generating a non self-intersecting surface with certain optimization. Representative work appear in [6, 8, 9, 13]. A survey for this one-to-one case appears in [16]. Various optimization criteria such as minimum surface area, maximum volume, minimum edge length were considered while constructing the surface, but none could guarantee a non self-intersecting surface. Finally, [11] explained this shortcoming by exhibiting two contours that cannot be triangulated without self-intersection unless new vertices are inserted on the contours. Subsequent approaches allowed extra points on the contours and concentrated more on the problem of correspondence and branching with multiple contours on slices.

Several approaches that deal with multiple contours on two slices have been proposed over the last few years [1, 3, 4, 7, 10, 14, 15]. The methods proposed by [7, 14, 15] only work for restricted cases as pointed out in [1]. The algorithm in [3] is reported to work well in practice based on the substantial experimental evidence presented. However, it requires to discretize the contour boundary sufficiently fine and given an arbitrary data set, it is not clear how to determine a good discretization quantitatively. The method in [1] combines the approach of [3] and [10] to handle the branching problem. However, it requires a strong assumption on input sampling distance. A competing approach appears in [4] which is based on Delaunay triangulations. This is refined further in [10]. As we have argued this Delaunay based method seems to be very effective in dealing with the correspondence and branching problem automatically.

Results. We investigate the approach of [4, 10] further to improve it in two ways. First, the output surface is generated without the costly step of computing three dimensional Delaunay triangulation. The surface triangles are detected directly from the two dimensional Voronoi diagrams of the contours. This requires a proper geometric characterization of these triangles and certain clever use of geometric properties of Voronoi diagrams. Nevertheless, the algorithm operates on simple data structures such as linked lists and priority queues and uses simple paradigms such as binary search and plane-sweep.

Secondly, we improve the postprocessing of [4, 10] that handle the branching and correspondence problem. In the process we fill up several loopholes undetected erstwhile. Unlike previous algorithms, we carry out a major part of these postprocessing steps while computing a *basic_surface*. Again, we avoid computing three dimensional Delaunay tetrahedra and extract relevant information for postprocessing only from the two dimensional Voronoi diagrams of contours.

Our algorithms are simple to implement and imminently practical. We have developed an implementation of our algorithm to examine visually the surface generated. Figure 2 shows an output surface for human heart generated by our implementation. We observe that, our contour surface produce the expected branchings and make necessary correspondences.

Paper Outline. Section 2 gives some definitions. Section 3 describes the algorithm to generate *basic_surface*. Section 4 describes the procedures for handling correspondence and branching problems. In Section 5, we comment on the implementation.

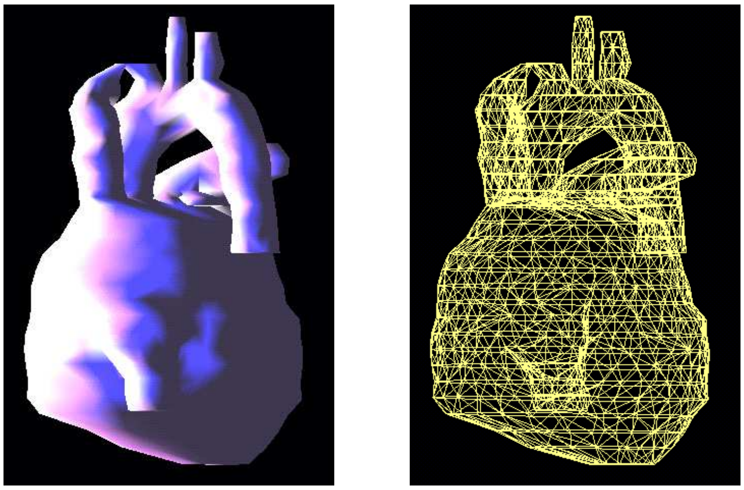

Figure 2: Human heart.

2 Preliminaries

Our approach makes extensive use of Voronoi diagram and its dual Delaunay triangulation. We briefly review their definitions. Given a set of points $U \in \mathbf{R}^2$, a *Voronoi face* ν_p for a point $p \in U$ is the set of all points in plane that are closer to p than any other point in U. Closure of each Voronoi face is a convex polygon. The boundaries of Voronoi faces consist of vertices and edges called *Voronoi vertices* and *Voronoi edges* respectively. In the absence of degeneracy, every Voronoi vertex has degree 3. Voronoi vertices, Voronoi edges and Voronoi faces form a planar subdivision of $\mathbf{R}^2$ which is called *Voronoi diagram*.

A dual planar subdivision called *Delaunay triangulation* can be defined from Voronoi diagram as follows. Dual of a Voronoi face ν_x is the vertex x. Dual of a Voronoi edge shared by two Voronoi faces ν_x and ν_y is the *Delaunay edge* defined as the line segment connecting x and y. Dual of a Voronoi vertex where three Voronoi faces ν_x, ν_y, ν_z meet, is a *Delaunay triangle* xyz with vertices x, y and z. For each Delaunay triangle xyz, the Voronoi vertex is the circumcenter and thus, the circumcircle of xyz does not contain a fourth point in U. Equivalently, a 2D Delaunay triangulation of U can be defined as the collection of triangles with vertices in U whose circumcircles are empty. This can be generalized to defining the 3D Delaunay triangulation as the collection of Delaunay tetrahedra whose circumspheres are empty.

Our goal is to construct a triangulated contour surface M interpolating between two parallel slices P_0 and P_1. We call each triangle on M a *vertical triangle* and each edge on M connecting P_0 and P_1 a *vertical edge*. The region bounded by M, P_0, and P_1 is called a *solid connection*. A solid connection may consist of several components when there are more than one contour on P_0 and P_1. Our algorithm constructs a surface M that is embedded in the 3D Delaunay triangulation of the vertices on the boundaries of the contours in P_0 and P_1. The following can be shown using empty sphere property of Delaunay triangulation.

Fact 1 *Let T_0, T_1 be the two dimensional Delaunay triangulations of the two sets of points S_0 and S_1 on two* *parallel planes. The three dimensional Delaunay triangulation of points in $S_0 \cup S_1$ contains T_0 and T_1 as subsets.*

Fact 1 allows the freedom of constructing the two dimensional triangulations of S_0 and S_1 independently of the three dimensional Delaunay triangulation, and still being guaranteed that they will be part of it. The solid connection between two slices, P_0 and P_1, must respect their boundaries, i.e., the boundaries of P_0 and P_1 must appear on the reconstructed surface. However, in general, the boundary edges of P_i, $i = 0, 1$, may not appear in the two dimensional Delaunay triangulations of its vertex set S_i. To ensure this we repeatedly add vertices on the edges of P_i and recompute the Delaunay triangulation until all edges of P_i appear as union of Delaunay edges. The new vertices are added in the middle of the so far unrealized segments. Our experiments show that this preprocessing increases the number of vertices only by a small factor, typically 3 to 5. The other approaches [1, 4, 3, 10] also indulge in adding extra points in different ways. Given the result of [11], this cannot be avoided if a non self-intersecting surface is to be output.

3 The basic method

Let D_i denote the 2D Delaunay triangulation of the augmented set of vertices in P_i as described above. Let $\mathcal{D}$ denote the 3D Delaunay triangulation of vertices in P_0 and P_1. Each tetrahedron in $\mathcal{D}$ is formed by connecting either (i) an edge e in one slice to another edge f in the other slice, or (ii) one vertex p in one slice to a triangle t in the other slice. We denote the tetrahedron in (i) by $e * f$ (tetrahedron made by e and p_0p_1 in Figure 4(d)) and in (ii) by $p * t$ (tetrahedron made by p_0 and t in Figure 4(a)). Similarly, a triangle in $\mathcal{D}$ formed by a vertex p in one slice and an edge e from the other slice is denoted by $p * e$. Define $contour(D_i)$ to be the set of vertices, edges, and triangles in D_i that lie inside or on the boundaries of the contours in P_i. Figure 3 shows $contour(D_i)$ for a slice P_i.

Let V_i denote the Voronoi diagrams of vertices in P_i, which is the dual of the Delaunay triangulation D_i. For each Delaunay vertex p, Delaunay edge e, and Delaunay triangle t, we use ν_p, ν_e, and ν_t to denote the dual Voronoi face, dual Voronoi edge, and dual Voronoi vertex respectively. Corresponding to $contour(D_i)$, we define $contour(V_i)$ to be $\{\nu_\sigma | \sigma \in contour(D_i)\}$. In words, $contour(V_i)$ is the set of all Voronoi vertices, edges and faces that are dual to Delaunay triangles, edges and vertices respectively lying inside or on the boundary of the contour. Trivially all vertices of the contours are in $contour(D_i)$ and hence their dual, all Voronoi faces are in $contour(V_i)$. However, this is not true for Delaunay edges anymore and they are classified as lying inside or lying outside the contour and hence partition the Voronoi edges accordingly. Similar statement can be made about the Delaunay triangles and hence Voronoi vertices. Figure 3 shows a slice P_i, D_i, $contour(D_i)$, and $contour(V_i)$.

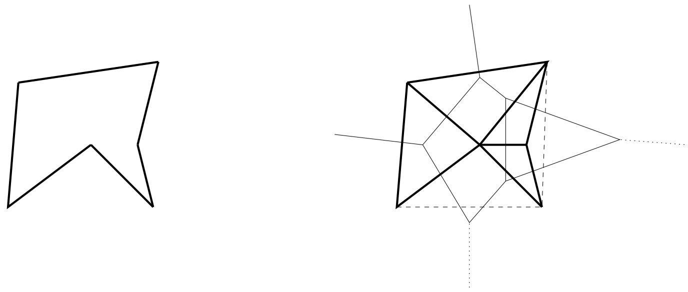

Figure 3: A slice P_i with one contour is on the left. On the right, D_i is the triangulation made by bold and dashed segments, out of which bold ones denote $contour(D_i)$. V_i is the union of regions made by solid and dotted segments, out of which $contour(V_i)$ is in solid.

A surface respecting the boundaries of P_i can be constructed in the following manner. First, compute the two dimensional Delaunay triangulation of the vertices in S_i. Assume that S_i has been augmented as described before to accommodate all the edges of P_i. Next, consider the three dimensional Delaunay triangulation of the vertices in S_0 and S_1 together. Remove all tetrahedra from this triangulation that are incident with any edge outside the boundaries of the contours P_0 and P_1. This peeling of tetrahedra exposes a surface connecting two slices, which we call a *basic_surface*. Of course, this tetrahedra peeling cannot be done explicitly as we wish to avoid computing the three dimensional triangulation. We elaborate a method that extracts exactly the triangles of the *basic_surface* using the two dimensional Delaunay triangulations D_0 and D_1 and their duals V_0 and V_1 respectively.

The following property of Delaunay triangulations is proved in [4] which gives an alternative method of determining Delaunay tetrahedra from two dimensional triangulations on two parallel planes. Let π be the function that projects an object in $\mathbf{R}^3$ vertically onto a plane parallel to P_0 and P_1.

Fact 2 *(i) Let e and f be two edges in D_0 and D_1 respectively. Then, $e * f$ is a tetrahedron in the three dimensional Delaunay triangulation $\mathcal{D}$ iff the projections of the dual Voronoi edges, i.e., $\pi(\nu_e)$ and $\pi(\nu_f)$ intersect.*
(ii) Let p be a vertex and t be a triangles in D_0 and D_1 respectively. Then, $p * t$ is a tetrahedron in $\mathcal{D}$ iff $\pi(\nu_t)$ lies in $\pi(\nu_p)$.

Now we proceed to characterize the triangles that constitute the *basic_surface* and then outline an algorithm to compute them efficiently.

3.1 Characterization of vertical triangles

The vertical triangles on basic_surface can be singled out by the following crucial property. *Any vertical triangle on basic_surface must be either incident on one tetrahedron in $\mathcal{D}$, or on two tetrahedra, one of which is in solid connection, and the other is not.* In essence this is the reformulation of the fact that the triangles on the basic_surface are exposed on one side. This fact allows us to characterize these triangles in terms of the elements of D_0 and D_1.

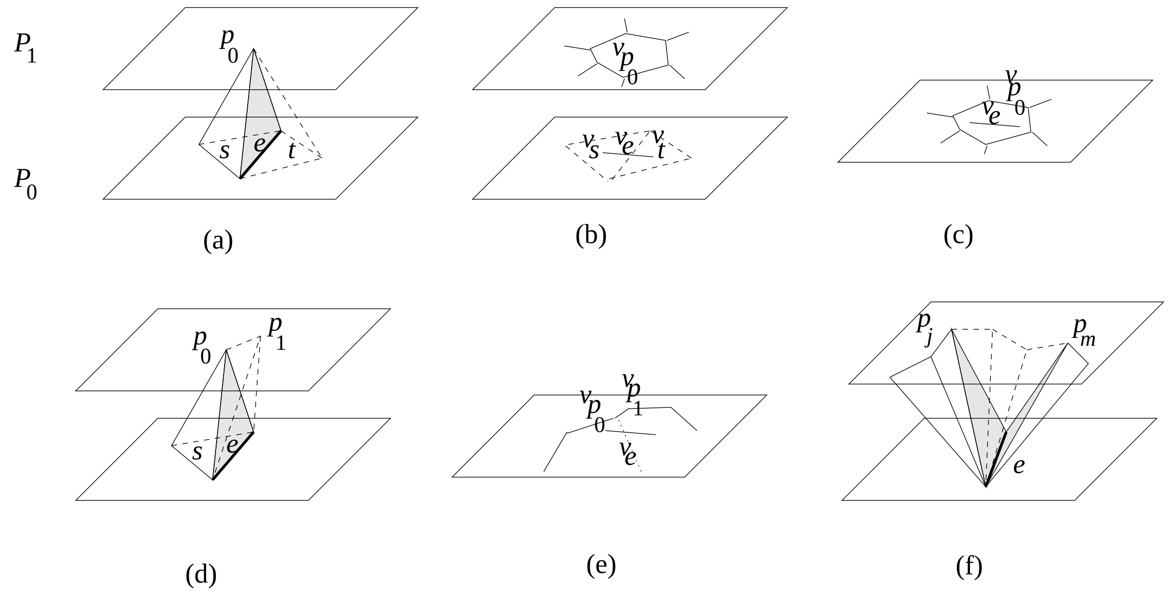

Figure 4: Illustration for conditions (C1)-(C5). The shaded triangles $p_0 * e$, $p_j * e$ and $p_m * e$ are on *basic_surface*.

Consider the illustration on the upper half of Figure 4. A single vertex p_0 from D_0 makes two tetrahedra in $\mathcal{D}$ with two triangles s and t in D_1. Consider projecting two Voronoi diagrams V_0 and V_1 onto a same plane. Then, by Fact 2, ν_s and ν_t, the dual Voronoi vertices of triangles s and t respectively lie inside the Voronoi face ν_{p_0} dual to p_0; see Figures 4(b) and 4(c). This means ν_e lies entirely inside the projected Voronoi face ν_{p_0}. If e is a boundary edge demarking the triangle s inside a contour in P_0 and the triangle t outside, then $p_0 * e$ is a triangle on *basic_surface*. This can be restated as: if ν_s and ν_t both lie inside ν_{p_0} and ν_t does not belong to $contour(V_0)$ then $p_0 * e$ is a triangle on basic_surface. This is precisely the condition stated in (C1) below. Of course, we have to consider more generalized intersections among projected Voronoi edges and faces to find out other conditions. In general, a Voronoi edge of one diagram may intersect several Voronoi faces of the other. We make it precise in the following.

Let e be a Delaunay edge inside a contour in P_0, i.e., in $contour(D_0)$. By definition, the dual Voronoi edge ν_e lies in $contour(V_0)$. Let ν_s and ν_t be two endpoints of ν_e. Since ν_e lies in $contour(V_0)$ at least one of the endpoints, say ν_s, lies in $contour(V_0)$. Correspondingly, s and t are triangles incident to e where at least s is in $contour(D_0)$. Let the sequence of Voronoi faces in V_1 intersected by ν_e in the projection be ν_{p_0}, ν_{p_1}, ..., ν_{p_k} ordered along ν_e from ν_s to ν_t. With these notations we characterize the vertical triangles incident to e in (C1)-(C5).

(C1) $p_0 * e$ is a vertical triangle if $p_0 = p_k$ and $\nu_t \notin contour(V_0)$.

The reason for (C1) is already explained. The reason for (C2) is illustrated in Figure 4(d) and (e). In this case ν_s lies in the projected Voronoi face ν_{p_0} but the other endpoint ν_t does not. Thus, ν_e intersects the projected Voronoi edge shared between the faces ν_{p_0} and ν_{p_1}. By Fact 2, p_0 forms a tetrahedron with s (ν_s lies in ν_{p_0}) and p_0p_1 forms a tetrahedron with e (ν_e intersects $\nu_{p_0p_1}$, shown dotted in Figure 4(e)). Now $p_0 * e$ is a vertical triangle if the tetrahedron formed by p_0p_1 and e do not belong to the solid connection. This happens when $p_0p_1 \notin contour(D_1)$ or equivalently the Voronoi edge dual to p_0p_1 does not belong to $contour(V_1)$. This condition can be stated precisely as follows.

(C2) $p_0 * e$ is a vertical triangle if $p_0 \neq p_k$ and the Voronoi edge $\nu_{p_0} \cap \nu_{p_1} \notin contour(V_1)$.

Similar analysis can be carried out for the last projected Voronoi face ν_{p_k} that ν_e intersects to obtain the following conditions.

(C3) $p_k * e$ is a vertical triangle if $\nu_{p_{k-1}} \cap \nu_{p_k} \in contour(V_1)$ and $\nu_t \notin contour(V_0)$.

(C4) $p_k * e$ is a vertical triangle if $\nu_{p_{k-1}} \cap \nu_{p_k} \notin contour(V_1)$ and $\nu_t \in contour(V_0)$.

Finally, consider Figure 4(f). The edge e makes a series of tetrahedra with a sequence of Delaunay edges through the vertices $p_0, p_1, ..., p_k$ in D_1. Out of the two edges $p_{j-1}p_j$ and p_jp_{j+1} incident with p_j in the chain of edges $p_0p_1 \cdots p_k$, one is in $contour(D_1)$ and the other is not. Equivalently, one dual edge in $contour(V_1)$ and the other is not. This means that exactly one of the two tetrahedra incident with the triangle $p_j * e$ (shaded in the figure) is in the solid connection. Therefore, $p_j * e$ is on *basic_surface*. This condition can be stated precisely as follows.

(C5) For $1 \leq j \leq k-1$, $p_j * e$ is a vertical triangle if exactly one of $\nu_{p_{j-1}} \cap \nu_{p_j}$ and $\nu_{p_j} \cap \nu_{p_{j+1}}$ lies in $contour(V_1)$.

One can repeatedly check conditions (C1)–(C5) for each edge in $contour(D_0)$ to find vertical triangles incident to them. Similar conditions single out vertical triangles incident on edges in $contour(D_1)$. It remains to describe an efficient method to check these conditions.

3.2 Algorithm for basic_surface

The algorithm to compute the vertical triangles in basic_surface use elementary data structures such as linked lists and priority queues and simple algorithmic paradigms from computational geometry such as binary search in a convex chain of edges and plane-sweeps. Figures 2 and 7 show two example outputs of our implementation.

After computing D_0 and D_1 into quad-edge data structures, it is a matter of simple walk over the edges and their adjacencies to mark the vertices, edges and faces that constitute $contour(D_0)$ and $contour(D_1)$. We discuss how to find vertical triangles incident to edges in $contour(D_0)$. A symmetric method works for edges in $contour(D_1)$. First, we locate the endpoints of the projected Voronoi edges of $contour(V_0)$ in the Voronoi diagram V_1. This is done with a simple plane-sweep algorithm of [2]. Then for each edge ν_e in $contour(V_0)$, we determine whether $p_0 * e$ and $p_k * e$ are vertical triangles by checking conditions (C1)–(C4). To check conditions (C1) and (C2), we carry out a binary search on the boundary of ν_{p_0} to determine if ν_e crosses any edge of ν_{p_0} in the projection. This binary search is possible since the boundary of a Voronoi face is convex. Similarly, to check conditions (C3) and (C4), the edge shared by $\nu_{p_{k-1}}$ and ν_{p_k} can be determined by a binary search on the boundary of ν_{p_k}. Hence, checking conditions (C1)–(C4) for all edges in $contour(D_0)$ involve only simple binary searches in convex chain of edges after the point locations through plane-sweep.

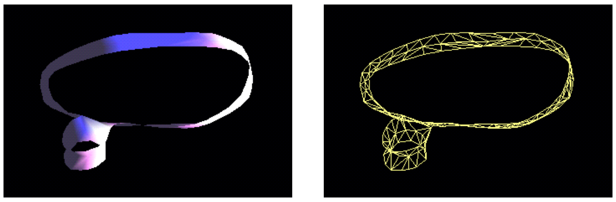

Figure 5: Branching in basic_surface of heart data.

Chords for condition (C5) Checking condition (C5) requires some nontrivial work. An obvious simple algorithm will be to go through the list of Voronoi faces $\nu_{p_0}, ..., \nu_{p_k}$ in V_1 intersected by the edge ν_e in V_0 in the projection, and determine the faces that have two edges intersecting ν_e (an edge can intersect at most two edges of a convex polygon), one is in $contour(V_1)$ and the other is not. This algorithm can take a lot of time since in the worst-case the edge ν_e can intersect a lot Voronoi faces of V_1 in the projection. In fact, if there are n Voronoi faces and edges in total, the number of such edge-face intersections could be as bad as $O(n^2)$. After all, we are trying to avoid computing the three dimensional Delaunay triangulation, one reason being that the number of tetrahedra could be as bad as quadratic.

We avoid this difficulty with the help of *chords* that allow us to detect the relevant faces satisfying (C5) quickly. It can be deduced from geometry that the bounding edges of a Voronoi face ν_p that are in $contour(V_1)$ form a connected chain. We draw the chord connecting the endpoints of this chain and call it $chord(p)$. For example, in Figure 6, the chord joining x and y forms $chord(p)$. Now, given an edge $\nu_e \in contour(V_0)$ and $\nu_p \in V_1$, if ν_e enters and exits ν_p through two edges, say ν_{f_1} and ν_{f_2}, then by (C5) we need to determine if exactly one of ν_{f_i} is in $contour(V_1)$. This is the case if and only if ν_e intersects $chord(p)$ and ν_e intersects the boundary of ν_p twice. Among all Voronoi faces intersected by ν_e, only ν_{p_0} and ν_{p_k} can have their chords intersected by ν_e but ν_e intersects their boundaries only once. This exception can be recognized easily by marking these faces while checking conditions (C1)-(C4). Hence, it suffices to determine all the chords intersecting ν_e. Observe that the chords in V_1 are non-intersecting and so are the edges in $contour(V_0)$. Thus, all the intersections among chords in V_1 (colored red) and the edges in $contour(V_0)$ (colored blue) can be determined by the algorithms for well-known red-blue intersection searching problem in computational geometry. In [5], a simple algorithm based on plane-sweep is proposed which reports all red-blue intersections in $O(n \log n + k)$ time, where k is the number of intersections reported. Observe that each such red-blue

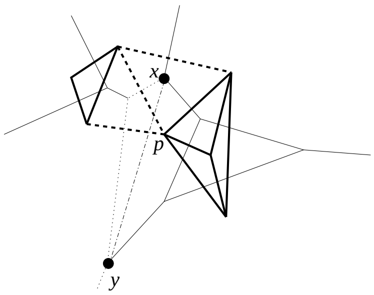

Figure 6: The Delaunay edges inside and outside $contour(D_i)$ are shown in bold and bold dashed respectively. The Voronoi edges inside and outside $contour(V_i)$ are shown in solid and solid dotted respectively. The solid dashed line segment joining x and y is $chord(p)$.

intersection corresponds to a vertical triangle to be constituted according to (C5). Therefore, considering the time required for all steps, we obtain an $O(n \log n + k)$-time algorithm that detects the k vertical triangles on *basic_surface*. Asymptotically, this is best possible since one cannot compute Delaunay triangles in less than $O(n \log n)$ time and k in the complexity cannot be avoided since that is the output size. Thus, our algorithm is output sensitive and asymptotically best possible.

Figure 5 shows the *basic_surface* computed for two slices taken from the heart data shown in Figure 2. The branching is detected correctly in the *basic_surface* itself without any postprocessing.

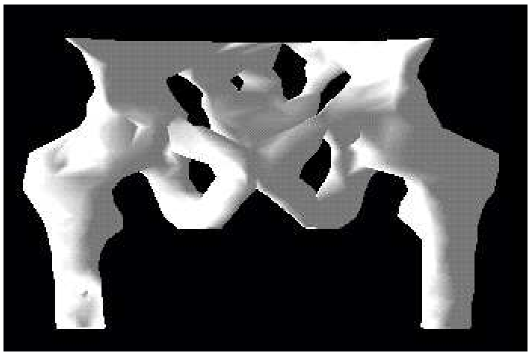

Figure 7: Human pelvis bone.

4 Handling correspondence and branching

The *basic_surface* may incorporate unnatural branchings and correspondence as shown in Figures 8 and 9. The surface may touch itself generating *pinchings*. A pinching occurs at an edge or a vertex if there is a point in their relative interior whose neighborhood on the surface (closure of vertical triangles) is not a topological disk. Such a pinching is undesirable because it cannot be realized by artificial or natural means. We formalize it with the following topological criterion.

Topological criterion Assume P_1 is lying above P_0. Let P_0^+ (P_1^+) be the solid obtained after thickening P_0 (P_1 respectively) by small amount in the downward (upward respectively) direction. Let U be the union of P_0^+, P_1^+, and the region bounded by a contour surface M respecting P_0, and P_1. Then there is a pinching in M if and only if there is a point in the boundary of U whose neighborhood is not a topological disk. In other words, the final surface connecting all slices should be a manifold. This assumption is justified since the target objects, human organs in particular, have manifold boundaries.

Pinchings can occur at edges or vertices of *basic_surface*. Table 1 tabulates the characterizations of the pinchings at vertices and edges in $contour(D_0)$ and pinchings at vertical edges that may happen in *basic_surface*. The same applies to vertices and edges in $contour(D_1)$. Figure 8 and Figure 9 illustrate the different types of pinchings. This classification takes into account all possibilities some of which are missing in [4, 10].

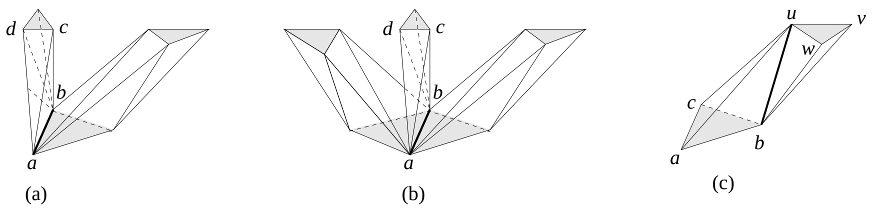

Figure 8: Pinching occurs at the bold boundary edge, bold internal edge, and bold vertical edge from left to right.

Table 1:

Pinching type	Characterization
boundary edge $e \in contour(D_0)$	e is incident to at least two vertical triangles, Figure 8(a)
internal edge $e \in contour(D_0)$	e is incident to at least three vertical triangles, Figure 8(b)
vertical edge e	e is incident to at least three vertical triangles, Figure 8(c)
hole-pinch at $p \in contour(D_0)$	The surface contains a simple cycle C of edges on $contour(D_1)$ so that $p * e$ is a vertical triangle for each $e \in C$ and C bounds a hole in $contour(D_1)$, Figure 9(a)
cone-pinch at $p \in contour(D_0)$	The surface contains a simple cycle C of edges on $contour(D_1)$ so that $p*e$ is a vertical triangle for each $e \in C$ and C does not bound a hole in $contour(D_1)$, Figure 9(b)
gorge-pinch at $p \in contour(D_0)$	The surface contains a simple cycle C of edges so that $p * e$ is a vertical triangle for each $e \in C$ and C contains at least one vertex of $contour(D_0)$, Figure 9(c).

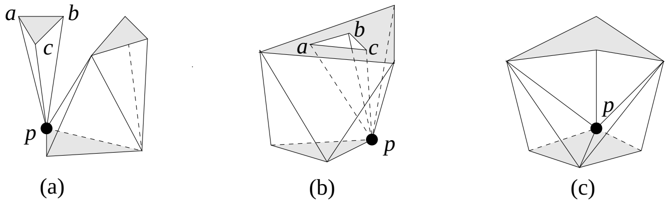

Figure 9: Pinching occurs at the black dots. The pinchings are cone-pinch, hole-pinch, and gorge-pinch from left to right.

Boissonat [4] proposed a method for removing pinchings from *basic_surface* which is refined by Geiger [10]. Unfortunately, although the proposed method removes these pinchings, it may produce new ones and this was completely overlooked. We suggest a remedy for this, and more importantly, we detect and remove pinchings on the fly while constructing the basic_surface, thus completely eliminating any computation of three dimensional triangulation.

Overview It is easier to understand our handling of pinchings in *basic_surface* by referring to the solid connection defined by it keeping in mind that we never compute any tetrahedron and refer to them only for explaining our method. Pinchings at edges in $contour(D_i)$ imply the presence of tetrahedra in the solid connection that are connected to the rest of the solid connection via a single edge. We call such tetrahedra *loose tetrahedra*. Removing them clears the contour surface of pinchings at edges in $contour(D_0)$ and $contour(D_1)$. For example, removing tetrahedra $abcd$ on the left and middle of Figure 8 eliminates the pinching at ab. A cone-pinch at a vertex p is incident to a collection of tetrahedra $\{p * t_i\}$. If we remove these tetrahedra, then the cone-pinch will be removed. For example, after removing tetrahedron $pabc$ on the left of Figure 9, the cone-pinch at p disappears. We also observe that pinchings at vertical edges must come together with cone-pinches and so pinchings at vertical edges will be removed as a side-effect. For example, removing tetrahedra $uabc$ and $buvw$ eliminates the cone-pinches at u and b as well as the pinching at bu. Finally, what remains are hole-pinches. A hole-pinch at a vertex p can be removed by adding tetrahedra incident to p to fill the "hole". For example, the hole-pinch at p in the middle of Figure 9 can be removed by adding tetrahedron $pabc$. However, to respect the boundaries of contours, we cannot do this directly. Instead, we introduce hypothetical slices between P_0 and P_1. A gorge-pinch at a vertex p is actually a poorly realized branching. For better realization, we will add more "material" around p to remove the pinching. This can intuitively be accomplished by adding appropriate tetrahedra in $\mathcal{D}$ that are incident to p. Since this may introduce new pinchings, we will again introduce hypothetical slices or make use of hypothetical slices introduced. An implementation result for the heart data is shown in Figure 10, where branchings for arteries have been well detected after removing pinchings.

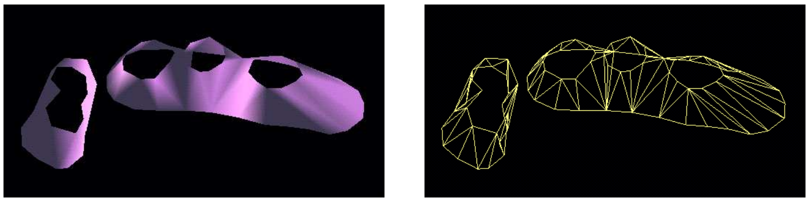

Figure 10: Branching in major arteries

Correspondence It is obvious that our handling of pinchings may leave some triangles on $contour(D_0)$ and $contour(D_1)$ not incident to any tetrahedra. This is desirable since these triangles may form the base or top of new solid connections that do not lie between P_0 and P_1. This takes care of the false correspondences detected in basic_surface. In Figure 2 such a detection was made, where the beginning of an aorta is cleared of any false connection to any contour below it. Figure 11 shows the multiple correspondence detected correctly by our algorithm for two slices of the heart data.

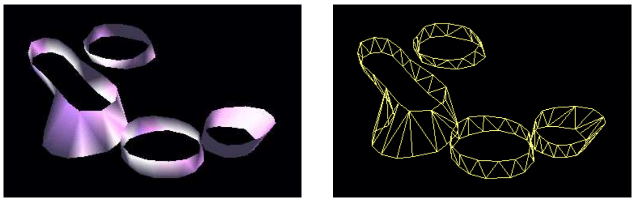

Figure 11: Correspondence in heart data

4.1 Removal of pinchings at edges on slices

A tetrahedron $e * f$, where $e \in contour(D_0)$ and $f \in contour(D_1)$, is loose if $e * f$ is not connected through a sequence of face-to-face attached tetrahedra to some tetrahedra with base in $contour(D_0)$ and $contour(D_1)$. They can be detected from the projections of V_0 and V_1.

Critical and Non-critical Segments. For each Voronoi edge $\nu_e \in V_i$, we compute a *non-critical segment* $seg(\nu_e)$ as follows. Let ν_t be the left endpoint of ν_e. If $\nu_t \notin contour(V_i)$, then ν_t is the left endpoint of $seg(\nu_e)$. Otherwise, the left endpoint of $seg(\nu_e)$ is the leftmost intersection between ν_e and ν_f for some $\nu_f \notin contour(V_{i+1 \bmod 2})$. Similarly, let ν_s be the right endpoint of ν_e, if $\nu_s \notin contour(V_i)$, then $\pi(\nu_s)$ is the right endpoint of $seg(\nu_e)$. Otherwise, the right endpoint of $seg(\nu_e)$ is the rightmost intersection between $\pi(\nu_e)$ and $\pi(\nu_f)$ for some $\nu_f \notin contour(V_{i+1 \bmod 2})$. Using plane-sweep algorithms, we can identify a set of non-critical segments are easily identified in $O(n \log n)$ time. For each projected Voronoi edge ν_e, we call $\nu_e - seg(\nu_e)$ the *critical segments.* Non-critical segments are useful because of the following observation.

Fact 3 *A tetrahedron $e * f$ is loose if the intersection of ν_e and ν_f lies on a non-critical segment of ν_e or ν_f.*

When the condition in Fact 3 is true, say the intersection $\nu_e \cap \nu_f$ lies on the non-critical segment of ν_e, the tetrahedron $e * f$ is in a gap delimited by two tetrahedra that are missing from the solid connection. Thus, $e * f$ must be loose.

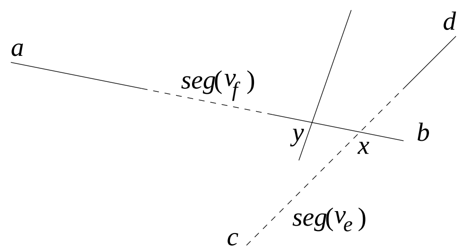

Figure 12: Solid line segments denote critical segments. ab denotes ν_f and the dashed line segment on it denotes its non-critical segment $seg(\nu_f)$. cd denotes ν_e and the dashed line segment on it denotes $seg(\nu_e)$. In the current round, the tetrahedron corresponding to the intersection y is not discovered to be loose. But after this round, $seg(\nu_f)$ should be lengthened to the point x and so the tetrahedron corresponding to y will become loose in the next round.

On-fly detection of loose tetrahedra. By Fact 3, if we compute the set of tetrahedra corresponding to intersections among critical segments, then we will successfully eliminate some loose tetrahedra. In fact, for each critical segment, only the two extreme intersections are needed to generate the vertical triangles on the modified solid connection. Thus, a left-to-right and a right-to-left plane-sweep are sufficient to identify all these extreme intersections. Unfortunately, a non-critical segment discovered may intersect a critical segment and such an intersection means a loose tetrahedron. Thus, new pinchings may be produced. For each Voronoi edge ν_e, we should shorten its critical segments and correspondingly lengthen its non-critical segment using non-critical segments discovered. This calls for another round of plane-sweep. For example, see Figure 12, the intersection x between the critical segment $seg(\nu_e)$ and the projected Voronoi edge ν_f corresponds to a loose tetrahedron which should be removed. After removing this loose tetrahedron, the tetrahedra corresponding to intersections y and z become loose but they are not detected in this round. One solution is to lengthen $seg(\nu_f)$ to include x as a new endpoint. If we find that any critical segment has been lengthened in the current round, we can plane-sweep again to identify extreme intersections on each critical segment. Theoretically, many rounds may be needed to get rid of all of them, but our experiments on human data took only one to two rounds.

4.2 Removal of gorge-pinch

Let M_1 be the contour surface after removal of pinchings at all edges and removal of cone-pinches. We handle gorge-pinches in M_1 by transforming them to hole-pinches. First, we traverse the incident vertical triangles for each vertex in M_1 to identify cycles matching the characterization of gorge-pinches. This finds all gorge-pinches. Let $\mathcal{D}_1$ be the solid connection bounded by M_1. Let p be a gorge-pinch in $contour(D_0)$ with C being the cycle of edges characterizing the gorge-pinch p. Let $\{v_i \,|\, 1 \leq i \leq m\}$ be the vertices in C that lie on $contour(D_0)$. By definition, S_1 contains the edges pv_i for all i. By the absence of loose tetrahedron, each pv_i is an internal edge and incident to exactly two vertical triangles pv_ix_i and pv_iy_i. When removing the gorge-pinch at p, we try to prevent the corrected surface from being nearly pinched at p. In [4], it is suggested to remove a gorge-pinch p by adding tetrahedra outside $\mathcal{D}_1$. One possibility is as follows. For $1 \leq i \leq m$, add the tetrahedra $pv_i * e$, where the edge $e \notin contour(D_1)$ is in the chain of edges from x_i to y_i such that ν_e intersects ν_{pv_i}. Intuitively, this erects a wall around p, see the left figure in Figure 13. This transforms p to a hole-pinch, see the middle left figure of Figure 13. There are two problems with this strategy. First, it does not respect the boundaries of contours. Second, it may introduce new pinchings, for example, there is a pinching at the edge ab in the middle right figure of Figure 13. In [4], the boundaries of original contours are respected by adding hypothetical slices and interpolating between the hypothetical slices instead. But this does not prevent introducing new pinchings. We improve this strategy by chopping off the top parts of these additional tetrahedra, see the right figure in Figure 13. These chopped tetrahedra transform the gorge-pinch at p to a hole-pinch. Moreover, the boundaries of $contour(D_1)$ are respected and no new pinching is introduced. This is equivalent to adding a hypothetical slice Q, which is similar to a union of $contour(D_0)$ and $contour(D_1)$ with the planar polygon $abcdwxyz$ added, between P_0 and P_1. In general, we need to introduce two hypothetical

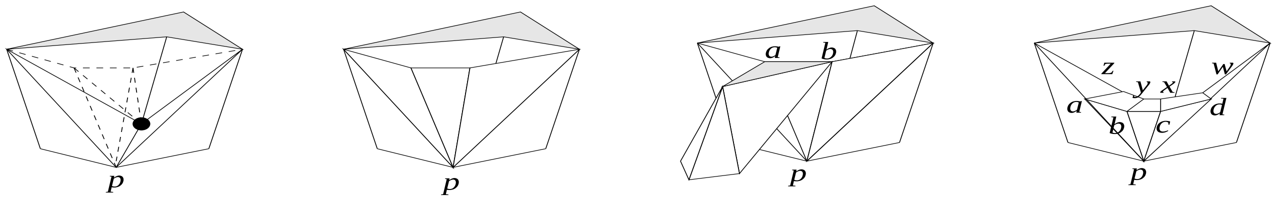

Figure 13:

slices to handle gorge-pinches on both P_0 and P_1.

5 Implementation

Figure 2 and Figure 7 show the result of our implementation on data for a human heart and a human pelvis bone. The implementation in C++ is relatively simple since it only manipulates two dimensional data structures for Voronoi diagrams and embodies simple algorithmic paradigms of binary search and plane-sweeps. The program is made robust against numerical errors and degeneracies using a special package developed at University of Illinois, Urabana Champaign (SOS) that uses adaptive filter arithmetic for speed and simulation of simplicity for degeneracies. The display uses a graphical platform JYAMITI developed at IIT Kharagpur, India on Silicon Graphics Indy workstation using openGL.

We observed the following facts. In most cases the *basic_surface* detects the correct branching and correspondence, Figure 5 and 11. The gorge pinch never occured. Pinchings were mostly in the form of edge pinchings that got removed with at most two passes, Figure 10. Cone pinches occured few times where new branching such as the beginning of an artery had to be realized. The results demonstrate that our algorithm derives the correct connection across the slices without user intervention. Running time of our program increased with the use of SOS package, which cannot be avoided if robustness against numerical errors is desired. We do not have available data on other methods using similar packages to make a fair comparison. However, the program computed far fewer triangles than the 3D Delaunay triangulation of the vertices on the contours would require. Finally, we mention that the visual quality of the contour surface can be improved by known mesh refinement schemes.

References

[1] C. Bajaj, K. Lin, and E. Coyle. Arbitrary topology shape reconstruction from planar cross sections. *Graphical Models and Image Processing* **58** (1996), 524–543.

[2] G. Blankenagel and R.H. Guting. Internal and external algorithms for the points-in-regions problem-the INSIDE join of geo-relational algebra. *Algorithmica* **5** (1990), 251–276.

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

[4] J. D. Boissonnat. Shape reconstruction from planar cross-sections. *Computer Vision, Graphics, and Image Processing* **44** (1988), 1–29.

[5] T. Chan. A simple trapezoid sweep algorithm for reporting red/blue segment intersections. *In* "Proc. 6th Canad. Conf. Comput. Geom. 1994", 263–268.

[6] H. N. Christiansen and T. W. Sederberg. Conversion of complex contour line definitions into polygonal element mosaics. *Computer Graohics* **12** (1978), 187–192.

[7] A. B. Ekoule, F. C. Peyrin and C. L. Odet. A triangulation algorithm from arbitrary shaped multiple planar contours. *ACM Trans. Graphics* **10**, 1991, 182–199.

[8] H. Fuchs, Z. Kedem and S. P. Uselton. Optimal surface reconstruction from planar contours. *Communications of the ACM* **20** (1977), 693–702.

[9] S. Ganapathy and T. G. Dennehy. A new general triangulation method for planar contours. *Computer Graphics* **16** (1982), 69–75.

[10] B. Geiger. Three-dimensional modeling of human organs and its application to diagnosis and surgical planning. Technical report, 2105, INRIA, France, 1993.

[11] C. Gitlin, J. O'Rourke and V. Subramanian. On reconstructing polyhedra from parallel slices. *Intern. J. Comput. Geom. Appl.* **6** (1996), 103–122.

[12] A. Kaul and J. Rossignac. Solid-interpolating deformations: construction and animation of PIPs. Comput. & Graphics, 16 (1992), 107–115.

[13] E. Keppel. Approximating complex surfaces by triangulation of contour lines. *IBM J. Research Development* **19**, 1975, 2–11.

[14] D. Meyers, S. Skinner and K. Sloan. Surfaces from contours: The correspondence and branching problems. *In* "Proc. Graphics Interface '91, 1991", 246–254.

[15] Y. Shinagawa and T. L. Kunii. The homotopy model: a generalized model for smooth surface generation from cross sectional data. *The Visual Computer* **7** (1991), 72–86.

[16] K. R. Sloan and J. Painter. From contours to surfaces: Testbed and initial results. *In* "Proc. CHI+GI '87, 1987", 115-120.