Straight-Skeleton Based Contour Interpolation*

Gill Barequet† Michael T. Goodrich‡ Aya Levi-Steiner§ Dvir Steiner¶

Abstract

In this paper we present an efficient method for interpolating a piecewise-linear surface between two parallel slices, each consisting of an arbitrary number of (possibly nested) polygons that define 'material' and 'nonmaterial' regions. This problem has applications to medical imaging, geographic information systems, etc. Our method is fully automatic and is guaranteed to produce non-self-intersecting surfaces in all cases regardless of the number of contours in each slice, their complexity and geometry, and the depth of their hierarchy of nesting. The method is based on computing cells in the overlay of the slices, that form the symmetric difference between them. Then, the straight skeletons of the selected cells guide the triangulation of these cells. Finally, the resulting triangles are lifted up in space to form an interpolating surface. We provide some experimental results on various complex examples to show the good and robust performance of our algorithm.

Keywords: Piecewise-linear interpolation, surface reconstruction.

1 Introduction.

The reconstruction of a polyhedral surface from a sequence of parallel polygonal slices has been an intriguing problem during the last thirty years. This problem arises primarily in the fields of medical imaging, digitization of objects, and geographical information systems. Data obtained by medical-imaging apparata, range sensors, or elevation contours are interpolated in order to represent, reconstruct, and visualize human organs, CAD objects, or topographic terrains. It is assumed that a preprocessing step has already extracted from the raw data (usually a sequence of pixel images) the closed two-dimensional contours, which delimit the material regions on each slice. Then, the goal becomes to compute a surface that tiles between these contours and forms a solid volume whose cross-sections at the given heights identify with the input slices.

Various algorithms for two-dimensional based polyhedral surface construction have been suggested in the literature (e.g., [Ke75, FKU77, GD82, WA86, KF88, KSF88, SP88, WW94]). Many of the early algorithms fail in complex instances (such as multiple branching), leave gaps between the contours, and/or generate unacceptable solutions (e.g., self-intersecting surfaces). Some algorithms [CS78, Sh81, ZJH87, MSS91, CP94] reduce the more involved cases to the simple case where each slice contains only one contour. There have been only a few attempts [Bo88, BG92, BS96, BCL96, OPC96, CD99] 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 almost all the previous works, as well as in this paper, is that adjacent layers are independent. Thus, only a single pair of successive parallel slices are considered and interpolated at a time, and the reconstructed object is the concatenation of the interpolating models computed for all the layers. To the best of our knowledge, the only work that avoided this assumption is [BST00].

The algorithm suggested in the current paper 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 is guaranteed to interpolate a valid surface for any possible input, and is intuitive in the sense that it tends to minimize the surface area of the reconstruction. This is because it uses an offset distance function to locally decide which contour features to bind.

In a nutshell, the algorithm analyzes the overlay of a pair of slices in order to identify sets of contour portions (bounding a subset of the set of cells of the arrangement of contours) which are to be bound together. Then, the straight skeleton (a linearized version of the medial axis [AAAG95]) of each one of these cells is computed and used to guide a Steiner triangulation of the cell. Finally, the topology of the skeleton is used again for lifting the triangulation up to three dimensions.

*Work on this paper by the first author has been supported in part by the Abraham and Jennie Fialkow Academic Lectureship. Work by the second author has been supported by ARO MURI Grant DAAH04-96-1-0013 and by NSF Grants CCR-9732300, PHY-9980044, and CCR-0098068.

†Faculty of Computer Science, The Technion—Israel Institute of Technology, Haifa 32000, Israel. E-mail: barequet@cs.technion.ac.il

‡Dept. of Information and Computer Science, Univ. of California, Irvine, CA 92697. E-mail: goodrich@ics.uci.edu

§Faculty of Mathematics, The Technion—Israel Institute of Technology, Haifa 32000, Israel. E-mail: saya@t2.technion.ac.il

¶Faculty of Mechanical Engineering, The Technion—Israel Institute of Technology, Haifa 32000, Israel. E-mail: ovir@tx.technion.ac.il

The union of the lifted-up triangulations of all the chosen cells is the output surface. We emphasize that the algorithm is fully automatic without any tuning parameters, which are a major disadvantage of some previously-suggested algorithms.

Our algorithm is somewhat similar to that of Oliva et al. [OPC96]. The latter algorithm also computes the symmetric difference of the slices as ours does, and computes straight skeletons of some cells of the arrangement of contours of the two slices. The differences between the two approaches are in the other steps of the algorithms: (1) Oliva et al. classify the active cells differently from us; (2) We apply a different triangulation scheme, which avoids overhanging constructions that may be created by the algorithm of Oliva et al. (partial remedies were later proposed by Felkel and Obdržálek [FO99]); and (3) We apply a different method for assigning heights for intermediate vertices: while Oliva et al. use Euclidean distance, we use offset distance.

The paper is organized as follows. In Section 2 we give an overview of the algorithm. Section 3 describes the data-acquisition phase, Section 4 describes the analysis of the overlay of two slices, and Section 5 describes the computation of surface patch out of the straight skeleton of one cell of the slices' overlay. In Section 6 we analyze the complexity of the algorithm, and in Section 7 we present some experimental results.

2 Overview of the Algorithm.

Our proposed algorithm consists of the following steps:

1. **Data acquisition**. Orient all the contours in each slice in consistent directions. If the input does not include this information, compute the contour nesting hierarchy in each slice, and use it to obtain the desired orientations.

2. **Analyzing the contour overlay**. Compute the overlay of the two slices. For each cell in the arrangement of polygons, attach a tag that identifies whether the cell lies in the material or the nonmaterial regions of each slice. Discard all the cells that either belong to the material or to the nonmaterial regions in both slices.

3. **Surface interpolation**. Compute the straight skeletons of all the remaining cells, separately triangulate each region in the maps induced by the skeletons, and lift the triangulations up to three dimensions.

The following three sections describe the algorithm steps in detail.

3 Data Acquisition.

The data consist of a sequence of slices, all in the same file. Each slice consists of a hierarchy of contours, that is, a forest of closed simple polygons with nonintersecting boundaries, where a parent polygon fully encloses all its children, and no other contour is enclosed in the parent polygon and encloses one of its children. Each slice is also marked by its height along the z axis; thus every vertex is specified by its three coordinates. In what follows we restrict our attention to a single pair of successive slices, and describe the interpolation of a solid within the layer delimited by the slices.

Contours of the root level (not contained in any other contour) are assigned level 0, their holes are assigned level 1, etc. Thus, every even level consists of contours whose interior, in a sufficiently-small neighborhood of the contour, is the "material," and every odd level consists of contours whose interior, sufficiently near them, is the "nonmaterial." We orient the contours consistently, for example, so that for each contour, when viewed from above, the material lies to its right. Thus all even-level contours are oriented in a clockwise direction, when viewed from above, and all odd-level contours are oriented in a counterclockwise direction. If the containment hierarchy of the contours is omitted, we compute it ourselves. The construction of the hierarchy and of the contour orientations is easily performed using a standard line-sweep procedure in each slice (see [BS96]).

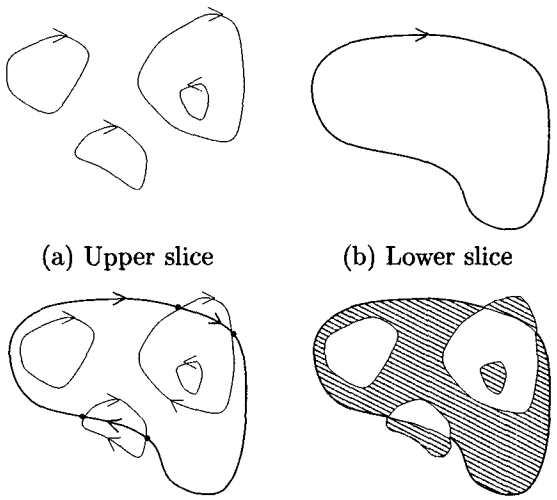
The internal representation of the contours that our system uses is the *quad-edge* data structure. This is done for maintaining efficiently the constructed polyhedral boundary of the interpolating solid object.

4 Analyzing the Contour Overlay.

We compute a representation of the arrangement of the contours of the two slices, obtained by projecting one slice onto the other (along the z direction). This can easily be done by applying a second line-sweep procedure on the overlay of the two slices. In fact, this step and the preceding step (the computation of contour hierarchy and orientation) can be performed at once. As part of sweeping the plane, each cell of the arrangement is attributed with indications whether it lies in the material or the nonmaterial regions of each of the slices.

We then discard all the cells that belong either to the material or to the nonmaterial regions of both slices. Thus we remain with only cells that correspond to material in one slice and nonmaterial in the other slice. Denote these as the *active* cells. Figure 1 shows two slices, their overlay, and the active cells of the overlay.

For the moment ignore the original polygon vertices



(a) Upper slice (b) Lower slice

(c) Slices overlay (d) Active cells

Figure 1: Active cells in the overlay of two slices

and consider only the vertices of the arrangement of the polygons (intersection points of two polygons, one of each slice). By 'contour portion' we mean a subpolygon whose endpoints two such vertices (intersections of the original polygons) and whose interior is free of other vertices. The endpoints of contour portions are seen in Figure 1(c).

THEOREM 4.1. *Each contour portion belongs to exactly one active cell.*

Proof. Consider any contour portion AB, where A and B are vertices of the arrangement of contours (intersection points of contours of the two slices). Assume without loss of generality that AB belongs to a contour of the first slice. Thus, it is shared by two cells of the arrangement, exactly one of which is material in the first slice. In the second slice, AB is fully contained in either the material or the nonmaterial region. In either case, by definition, AB bounds exactly one active cell.

Since we will use the boundaries of only the active cells for interpolating a surface between the two slices, we are now guaranteed that every contour portion will be used exactly once as a boundary of that surface. Together with the original contours, we will have a closed surface bounding a solid model.

5 Surface Interpolation.

5.1 Skeletons and Triangulations. At this point we have already found the boundaries of the interpolated surface. Our current goal is to construct a collection of pairwise-disjoint non-self-intersecting surface patches with known boundaries (the active cells). This is easy to achieve by forming a surface whose xy projection is simple, that is, every vertical line intersects the surface in at most one point. For ease of exposition we first describe the computation of the xy projection of the surface, and only then, its lifting up to three dimensions.

The xy projection of the interpolated surface is simply the union of all the active cells in the arrangement of all the contours, as is shown in Figure 1(d). We explain in detail how to create the triangulations of these cells (which after lifting up to three dimensions will contain the facets of the meshed surface).

We begin with computing the straight skeletons of all the active cells. Obviously, by construction, every face in the subdivision induced by the straight skeleton of a cell contains exactly one original polygon edge, and the face is monotone with respect to that edge [AAAG95, Lemma 3]. Then, among all possible triangulations of the face, we choose a triangulation that is as monotone as possible with respect to the polygon edge (see [BKOS97, pp. 55–58]). Figure 2 shows two examples of an overlay of two contours (shown with regular and thick lines), the straight skeletons of the active cells (shown with dashed lines), and their respective triangulations (shown with dotted lines). We chose this triangulation because it guides an intuitive reconstruction of a surface. However, any other triangulation will do. The union of the triangulations of all the active cells (guided by the respective skeletons) is the xy projection of the sought triangulated surface.

5.2 Lifting Up. We assume without loss of generality that the lower and the upper slices are at heights 0 and 1, respectively. In order to perform our final step—that is, to lift the surface up to three dimensions—all we have to do is to assign z coordinates to all the vertices of the straight skeletons. The following theorem will help us in doing so.

THEOREM 5.1. *The straight skeleton of every cell is the union of edge-disjoint trees whose roots are interior vertices of the straight skeleton which are equidistant from points on contours of both slices (with one exception: a cell bounded by a complete contour of one slice).*

This claim is shown in Figure 3. The portions of two contours which bound one cell are shown in regular and thick lines. The straight skeleton of the interior of the cell is divided into trees by vertices equidistant from the two contours. Two nontrivial trees are shown with dotted lines. The trivial trees (line segments) are shown with dashed lines.

Proof. Follows from the definition of the straight skeleton of a simple polygon.

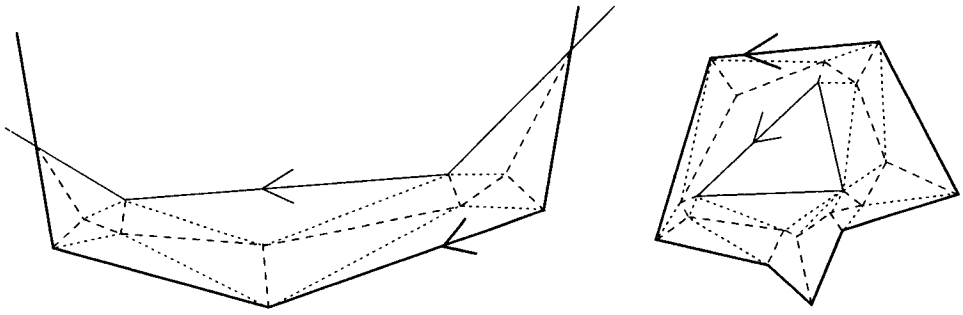


Figure 2: Active cells: Their straight skeletons and triangulations

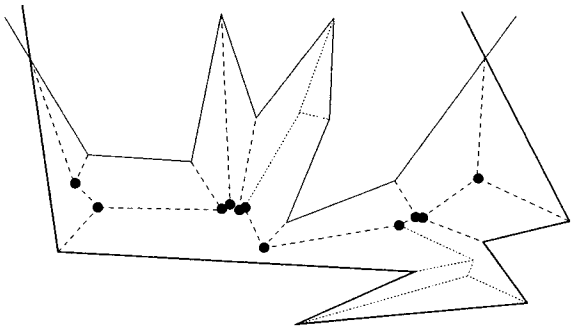


Figure 3: Trees in the skeleton, rooted at points equidistant from contours of the two slices

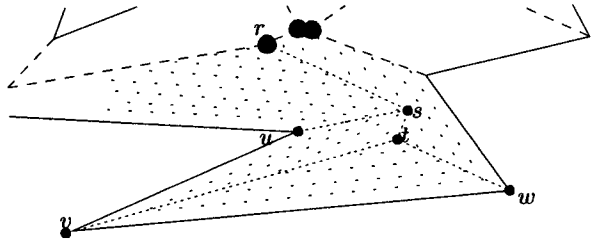


Figure 4: Setting vertex heights according to their offsets

In assigning z coordinates to vertices we distinguish between three cases:

1. Original polygon vertices. Here we naturally assign to the vertices the height of their respective slice, that is, either 0 or 1.

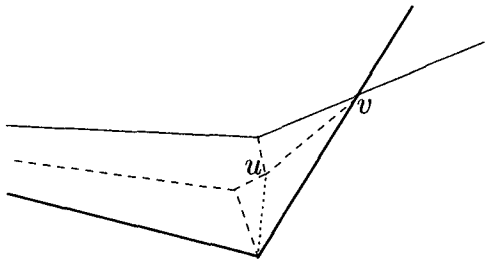
2. Internal vertices of the straight skeleton. Here we have three subcases:

   (a) Skeleton vertices which are equidistant from points on contours of both slices. We set the height of these vertices to 0.5.

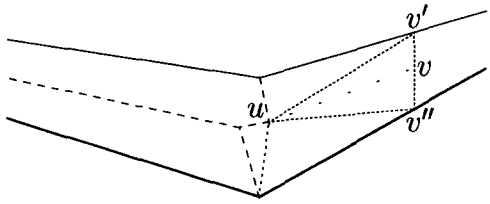
   (b) Skeleton vertices which are not equidistant from points on contours of the two slices. According to Theorem 5.1, these are internal vertices of trees, the heights of whose roots were already set to 0.5, and whose leaves are all at height either 0 or 1. Any monotone function can be used for setting the heights of the internal vertices of the trees. To reflect the relation to the straight skeleton, we use the *offset* distance function (see [BDG97]) from the contour, and normalize it so that its value is 0 or 1 on the contour and 0.5 at the root of the tree. Figure 4 shows a close-up of the skeleton tree at the bottom-right corner of Figure 3. The height of the vertices u, v, and w is 0 since they belong to the lower slice. The height of the root r is set to 0.5, whereas the heights of the internal tree vertices s and t are set to $\frac{3}{9} \cdot 0.5 = \frac{1}{6}$ and $\frac{5}{9} \cdot 0.5 = \frac{5}{18}$, respectively.[1] This choice of the z function fits our application due to the strong relation between the offset of a shape and its straight skeleton.

   (c) The special case of an active cell bounded completely by a contour of one slice indicates the vanishing or appearing of a feature of the three-dimensional object. Assume without loss of generality that the active cell is defined by a contour of the lower slice. Then, all the leaves of the skeleton are already assigned the height 0. We set the height of the skeleton vertex (or vertices) offset-wise furthest from the contour to 1, and use, as in the previous

[1] Obviously we must take care that the height of isolated branches of the tree does not exceed 1. For example, if the vertex t in Figure 4 was twice as far as r (offset-wise) from the boundary of the lower slice, which is possible since the contours are not necessarily convex, then its height would be more than 1. In this case we apply a secondary scaling on every isolated subtree.



(a) Top view



(b) Isometric view

Figure 5: Triangles sharing a contour-intersection point

case, the skeleton to guide the setting of the intermediate heights. We believe that setting the "vanishing height" to 1 is better than setting it to 0.5 (or to any other intermediate value in the open interval $(0,1)$), since this gives a smooth and continuous interpolation between the two slices, morphing a feature in one slice to its absence in the other slice. The other case (active cell modeled by a contour in the upper slice) is completely symmetric.

3. Intersection points of pairs of contours, one of each slice. In fact, such a point is actually three points whose xy projections identify with each other. Two of the points lie on the original contours, and thus have their original heights (0 and 1). The third point is a skeleton point which is set to be at height 0.5. Note that the two points on the original contours are not necessarily vertices of the input polygons, but only the intersection point of their xy projections. Figure 5(a) shows the xy projection of two slices, with a contour-intersection point v. Figure 5(b) shows the same scene from a lower viewpoint. The point v is actually the xy projection of three points v, v', and v'' (at heights 0.5, 1, and 0, respectively). The two supposed triangles in Figure 5(a) that share the vertex u are actually quadrangles, as is seen in Figure 5(b). We therefore triangulate them by adding two diagonals. In addition, the skeleton edge that coincides with the skeleton vertex v (uv in the figure) is deleted so as to form one triangle containing the edge $v'v''$ ($\Delta uv'v''$ in the figure).

After assigning the z values (heights) to all the vertices of the skeletons, we lift the collection of triangulated patches up to three dimensions. The result is the desired interpolation.

6 Complexity Analysis.

We measure the complexity of the algorithm as a function of n, the total complexity (number of vertices) of the two slices. We also denote by k the complexity of the overlay of the two slices. In the worst case k can be as high as $\Theta(n^2)$, but in most practical cases it is $O(n)$.

Computing the contour nesting hierarchy in each slice and orienting the contours in the correct directions takes $O(n \log n)$ time. This is performed by invoking a simple line-sweep procedure. If the input contours are guaranteed to be oriented well, we can skip this step of the algorithm.

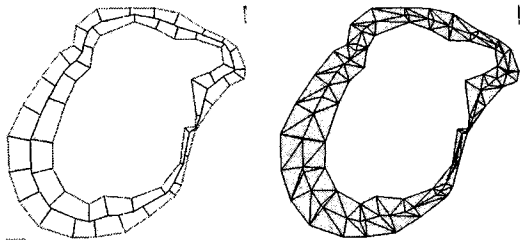
Computing and analyzing the overlay of the two slices takes $O(n \log n + k)$ time. This already includes the selection of the active cells of the overlay. Computing the straight skeletons of all the active cells can theoretically be done in $O(k^{17/11+\varepsilon})$ time, for every $\varepsilon > 0$ [EE99],[2] by using a sophisticated data-structure for ray-shooting queries, or even slightly better (in non-degenerate cases and on the average) in $O(k^{3/2} \log k)$ expected time [CV02]. Instead, we implemented the algorithm of [FO98] whose running time is $O(k^2)$ in the worst case, and in practice, much less than that. (Our experiments suggest that this step requires about $O(n^{1.6})$ time.) Triangulating the monotone subcells induced by the skeletons, as well as lifting the triangulations up to three dimensions to form the interpolating surfaces, take $O(k)$ time.

To conclude, the entire algorithm runs in $O(k^2)$ time. In the worst case this is $\Theta(n^4)$, but for most cases (in which the complexity of the slice overlay is linear in that of the original slices) the running time of the algorithm is theoretically $O(n^2)$, and in practice even less than that (around $O(n^{1.6})$ time in our experiments).

7 Experimental Results.

We implemented the entire algorithm in C++ on an HP Omnibook 6000 (a laptop). The computer was equipped with a Pentium III 850 MHz processor, 128 megabytes of memory, and an ATI Rage M1 AGP Mach 64 graphics card with 32 megabytes of memory. The implementation, performed by the third and fourth authors, took about two months, and the software consisted of about 8,500 noncomment lines of code. We experimented with the algorithm on several data files

[2] In fact, the precise running time is $O(k^{1+\varepsilon} + k^{8/11+\varepsilon} r^{9/11+\varepsilon})$, where r is the number of reflex contour vertices.



(a) Skeleton (b) Triangulated cell

Figure 6: A simple interpolation case

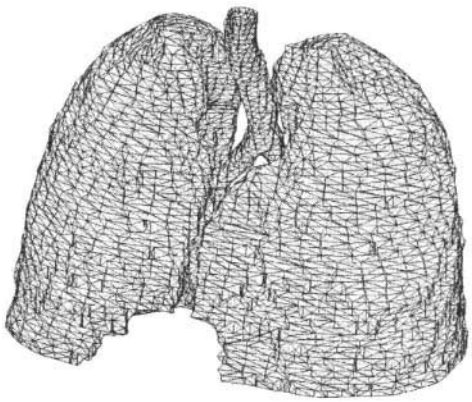
obtained by medical scanners, and obtained very good results in practically all cases.

Here are some specific examples of the performance of the algorithm:

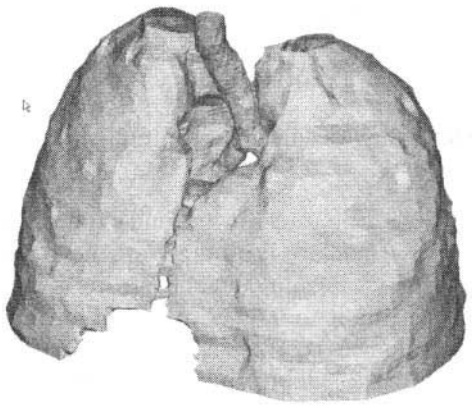
Figure 6(a) shows the overlay of two contours belonging to two successive slices (in black and grey). Since the two contours are nested, the single active cell is the ring bounded by the two contours. Its straight skeleton is shown with thick black lines. Figure 6(b) shows the triangulated ring.

Figure 7(a) shows an overlay of two complex slices (one in black and the other in grey) taken from a lungs data file. Note the almost-horizontal line that appears at the bottom of the figure. This is the swept line at the last discrete event that it handles. A close-up of the area in the middle of the overlay is seen in Figure 7(b). Figure 7(c) shows in thick black lines the straight skeletons of all the active cells in the arrangements of contours of the two slices. Similarly, Figure 7(d) is a close-up of Figure 7(c). Figure 7(e) shows a top-down view of the triangulated skeletons. A close-up of their middle area is shown in Figure 7(f). Figures 7(g,h) show a perspective view and a close-up of the surface interpolated between the two slices.

Figures 8(a,b) show a wire-frame and a shaded display of the fully-reconstructed pair of lungs. These data contained thirty slices, thus we invoked our algorithm 29 times. Table 1 displays statistics of these experiments. The running times of some stages were negligible and are thus omitted in the table. The experimental results show clearly that the most time-consuming step was the computation of the straight skeleton. In our implementation it indeed required time which was asymptotically quadratic in the size of the input, while all the other steps required time linear in the input size. This is clearly demonstrated in Figure 9, which shows a few relations between running times and output size to complexities of the input. (The displayed functions were approximated by the curve-fitting tool of Microsoft's Excel.) Overall, every layer was interpolated on average in less than one second.



(a) Wire-frame



(b) Shaded

Figure 8: A fully reconstructed pair of lungs

Figure 10 shows a reconstruction of part of a human pelvis. In the full paper we provide more examples and their statistics.

References

[AAAG95] O. Aichholzer, D. Alberts, F. Aurenhammer, and B. Gärtner, A novel type of skeleton for polygons, *J. of Universal Computer Science*, 1 (1995), 752–761.

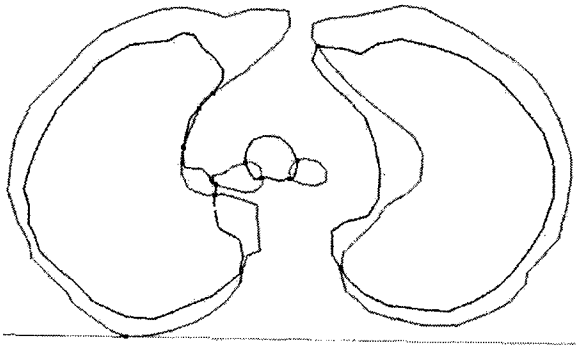
[BCL96] C.L. Bajaj, E.J. Coyle, and K.N. Lin, Arbitrary topology shape reconstruction from planar cross sections, *Graphical Models and Image Processing*, 58 (1996), 524–543.

[BDG97] G. Barequet, M.T. Dickerson, and M.T. Goodrich, Voronoi diagrams for polygon-offset distance functions, *Discrete & Computational Geometry*, 25 (2001), 271–291.

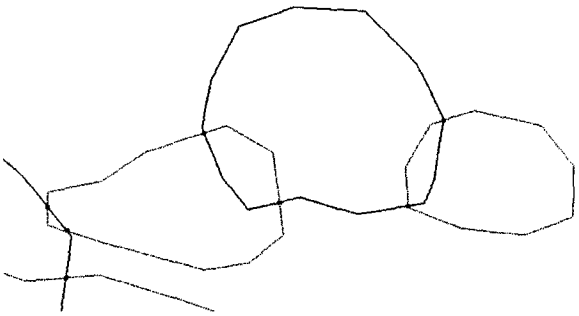
[BST00] G. Barequet, D. Shapiro, and A. Tal, Multilevel sensitive reconstruction of polyhedral surfaces from parallel slices, *The Visual Computer*, 16 (2000), 116–133.

[BS96] G. Barequet and M. Sharir, Piecewise-linear interpolation between polygonal slices, *Computer Vision and Image Understanding*, 63 (1996), 251–272.

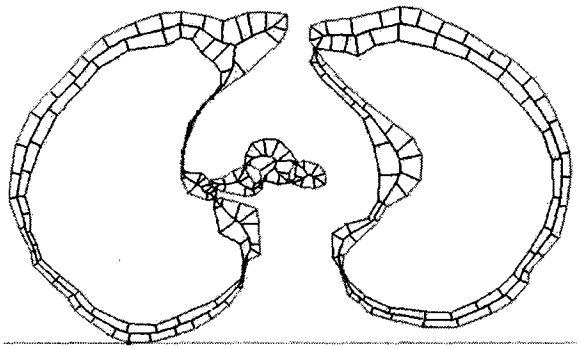
[BKOS97] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, Springer-Verlag, Germany, 1997.



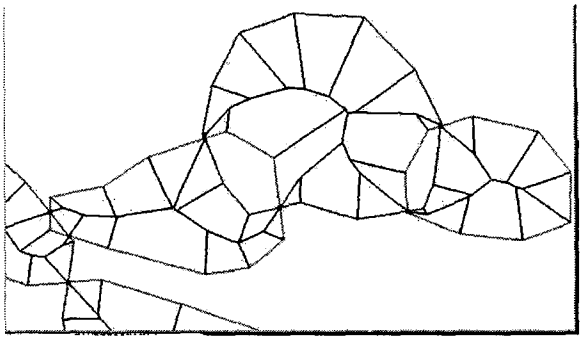
(a) Overlay of two slices



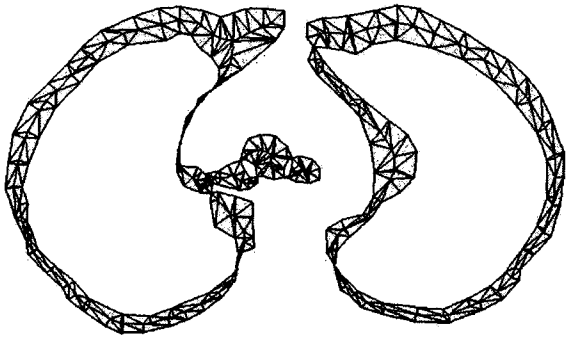
(b) Close-up of (a)



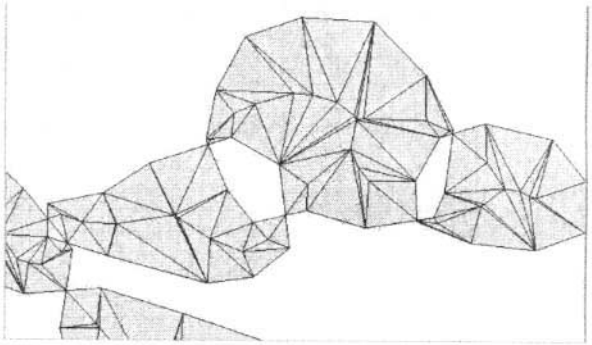
(c) Straight skeletons of active cells



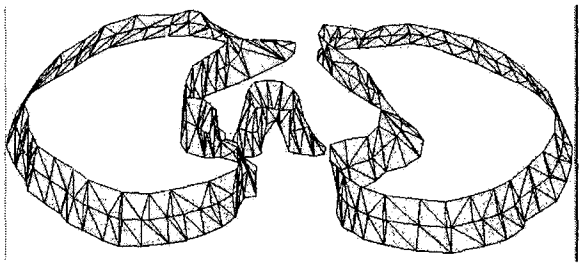
(d) Close-up of (c)



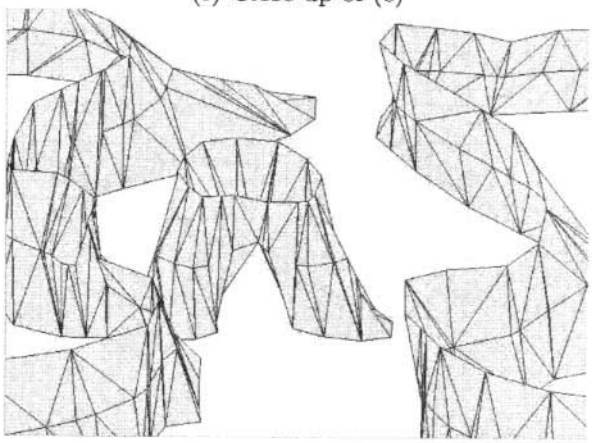
(e) Triangulated skeletons



(f) Close-up of (e)



(g) A perspective view of the interpolation



(h) Close-up of (g)

Figure 7: A complex branching example

Slice Number	Numbers of Contour Edges	Skeleton Edges	Surface Triangles	Time (Seconds) Line Sweep	Symm. Diff.	Skeleton	Height Setting	Triang.	Total
1	34	80	110			0.05			0.05
2	30	72	98			0.06			0.06
3	26	68	96			0.05			0.05
4	58	64	86			0.06			0.06
5	101	252	337			0.22		0.05	0.27
6	128	318	401			0.44		0.06	0.50
7	149	386	481			0.49			0.49
8	149	378	471	0.05		0.55			0.60
9	162	414	523	0.05		0.66		0.05	0.76
10	164	430	532			0.77		0.06	0.83
11	162	413	512			0.66		0.05	0.71
12	174	430	577			0.77		0.05	0.82
13	189	466	632		0.06	0.77			0.83
14	229	579	769			1.27		0.05	1.32
15	238	588	850		0.06	1.15		0.05	1.26
16	233	588	826		0.06	1.32		0.05	1.43
17	257	628	940		0.06	1.43	0.05	0.06	1.60
18	263	681	1,028		0.05	1.76		0.05	1.86
19	265	681	963		0.06	1.43		0.05	1.54
20	259	664	961	0.06		1.48		0.05	1.59
21	228	560	815		0.06	1.21	0.05	0.06	1.38
22	210	522	776			0.88	0.05		0.93
23	212	540	784			0.93		0.05	0.98
24	230	578	835		0.06	1.10		0.05	1.21
25	245	608	956		0.06	1.37		0.06	1.49
26	242	608	889	0.05		1.32		0.05	1.42
27	234	594	864	0.06		1.15		0.06	1.27
28	230	586	801		0.05	1.16		0.05	1.26
29	249	615	899	0.06		1.26	0.05	0.06	1.43
Total	5,350	13,391	18,812	0.33	0.58	25.77	0.20	1.12	28.00

Table 1: Performance of the algorithm (empty entries are practically 0)

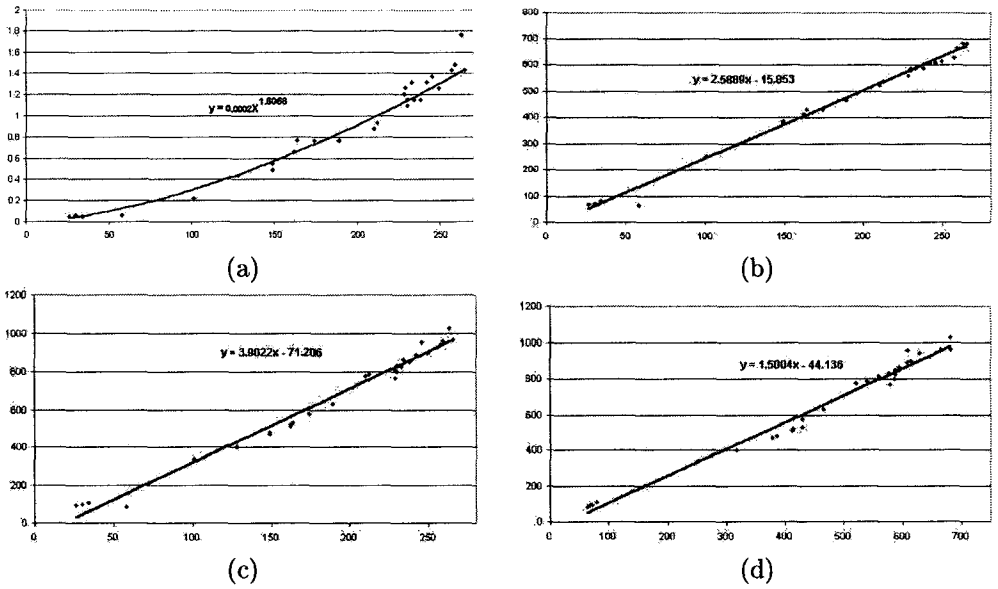


Figure 9: (a,b,c) Skeleton creation time, number of skeleton edges, and number of output triangles, respectively (in the lungs model), as functions of the number of input contour edges; (d) Number of output triangles as a function of intermediate skeleton edges.

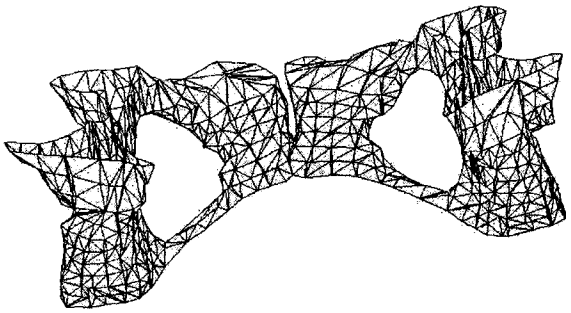


Figure 10: A reconstructed pelvis

[Bo88] J.D. BOISSONNAT, Shape reconstruction from planar cross sections, *Computer Vision, Graphics and Image Processing*, 44 (1988), 1–29.

[BG92] J.D. BOISSONNAT AND B. GEIGER, Three dimensional reconstruction of complex shapes based on the Delaunay triangulation, Technical Report 1697, Inria-Sophia Antipolis, 1992.

[CD99] S.W. CHENG AND T.K. DEY, Improved construction of Delaunay based contour surfaces, *Proc. ACM Symp. on Solid Modeling and Applications*, 322–323, 1999.

[CV02] S.-W. CHENG AND A. VIGNERON, Motorcycle graphs and straight skeletons, *Proc. 13th ACM/SIAM Symp. on Discrete Algorithms*, 156–165, 2002.

[CP94] Y.K. CHOI AND K.H. PARK, A heuristic triangulation algorithm for multiple planar contours using an extended double branching procedure, *The Visual Computer*, 10 (1994), 372–387.

[CS78] H.N. CHRISTIANSEN AND T.W. SEDERBERG, Conversion of complex contour line definitions into polygonal element mosaics, *Computer Graphics*, 13 (1978), 187–192.

[FO98] P. FELKEL AND Š. OBDRŽÁLEK, Straight skeleton computation, *Spring Conf. on Computer Graphics*, Budmerice, Slovakia, 210–218, 1998.

[FO99] P. FELKEL AND Š. OBDRŽÁLEK, Improvement of Oliva's algorithm for surface reconstruction from contours, *Spring Conf. on Computer Graphics*, Budmerice, Slovakia, 254–263, 1999.

[EE99] D. EPPSTEIN AND J. ERICKSON, Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions, *Disc. & Comp. Geometry*, 22 (1999), 569–592.

[FKU77] H. FUCHS, Z.M. KEDEM, AND S.P. USELTON, Optimal surface reconstruction from planar contours, *Communications of the ACM*, 20 (1977), 693–702.

[GD82] S. GANAPATHY AND T.G. DENNEHY, A new general triangulation method for planar contours, *ACM Trans. on Computer Graphics*, 16 (1982), 69–75.

[KF88] N. KEHTARNAVAZ AND R.J.P. DE FIGUEIREDO, A framework for surface reconstruction from 3D contours, *Computer Vision, Graphics and Image Processing*, 42 (1988), 32–47.

[KSF88] N. KEHTARNAVAZ, L.R. SIMAR, AND R.J.P. DE FIGUEIREDO, A syntactic/semantic technique for surface reconstruction from cross-sectional contours, *Computer Vision, Graphics and Image Processing*, 42 (1988), 399–409.

[Ke75] E. KEPPEL, Approximating complex surfaces by triangulation of contour lines, *IBM Journal of Research and Development*, 19 (1975), 2–11.

[MSS91] D. MEYERS, S. SKINNER, AND K. SLOAN, Surfaces from contours: The correspondence and branching problems, *Proc. Graphics Interface*, 246–254, 1991.

[OPC96] J.-M. OLIVA, M. PERRIN, AND S. COQUILLART, 3D reconstruction of complex polyhedral shapes from contours using a simplified generalized Voronoi diagram, *Computer Graphics Forum*, 15 (1996), C397–408.

[Sh81] M. SHANTZ, Surface definition for branching contour-defined objects, *Computer Graphics*, 15 (1981), 242–270.

[SP88] K.R. SLOAN AND J. PAINTER, Pessimal guesses may be optimal: A counterintuitive search result, *IEEE T-PAMI*, 10 (1988), 949–955.

[WA86] Y.F. WANG AND J.K. AGGARWAL, Surface reconstruction and representation of 3-D scenes, *Pattern Recognition*, 19 (1986), 197–207.

[WW94] E. WELZL AND B. WOLFERS, Surface reconstruction between simple polygons via angle criteria, *J. of Symbolic Computation*, 17 (1994), 351–369.

[ZJH87] M.J. ZYDA, A.R. JONES, AND P.G. HOGAN, Surface construction from planar contours, *Computers and Graphics*, 11 (1987), 393–408.