Available online at www.sciencedirect.com
SCIENCE DIRECT®
Graphical Models 66 (2004) 245–260
Graphical Models
www.elsevier.com/locate/gmod

Contour interpolation by straight skeletons☆

Gill Barequet,[a,*,1] Michael T. Goodrich,[b,2] Aya Levi-Steiner,[c,1] and Dvir Steiner[d]

[a] *Department of Computer Science, The Technion—Israel Institute of Technology, Haifa 32000, Israel*
[b] *Department of Information and Computer Science, University of California, Irvine, CA 92697, USA*
[c] *Department of Mathematics, The Technion—Israel Institute of Technology, Haifa 32000, Israel*
[d] *Department of Mechanical Engineering, The Technion—Israel Institute of Technology, Haifa 32000, Israel*

Received 21 October 2003; received in revised form 3 March 2004; accepted 4 May 2004
Available online 7 June 2004

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 'non-material' 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 each face of the skeletons. 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

☆ A preliminary version of this paper appeared in [5].

* Corresponding author. Fax: +972-4-8295538.
E-mail addresses: barequet@cs.technion.ac.il (G. Barequet), goodrich@ics.uci.edu (M.T. Goodrich), ayas@tx.technion.ac.il (A. Levi-Steiner), ovir@tx.technion.ac.il (D. Steiner).

[1] Work on this paper by the first and third authors has been supported in part by Israel's Ministry of Science Infrastructure Grant 01-01-01509 and through a donation from IBM Shared University Research program to the Technion in 2003.

[2] 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.

1524-0703/$ - see front matter
doi:10.1016/j.gmod.2004.05.001

1. Introduction

The reconstruction of a polyhedral surface from a sequence of parallel polygonal slices has been an intriguing problem during the last 30 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 as 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. The goal is to compute a surface that tiles between these contours and forms a solid volume whose cross-sections at the given heights match the input slices.

Various algorithms for two-dimensional based polyhedral surface reconstruction have been suggested in the literature (e.g. [8,20–22,24–26,30–32]). 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 [14–16,27,29,33] reduce the more involved cases to the simple case where each slice contains only one contour. There have been only a few attempts [2,7,10–12,28] 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 each instance, 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 [6].

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 *in practice* 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 [1]) of each one of these cells is computed and used to guide a Steiner triangulation of each face of the skeletalized cells. Finally, the topology of the skeleton is used again for lifting the triangulation up to three dimensions. 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. [28]. This latter algorithm also computes the symmetric difference of the slices as ours does, and com-

putes 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 than we do;
(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 [19]); 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 a 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. We end in Section 8 with some concluding remarks.

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 non-material regions of each slice. Discard all the cells that either belong to the material or to the non-material 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, each slice is a forest of closed simple polygons with non-intersecting 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 "non-material." We orient the contours consistently, for example, so that for each contour, when viewed from above, the material lies to its right. Consequently, 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 [7]).

The internal representation of the contours that our system uses is the *quad-edge* data structure described by Guibas and Stolfi [23]. This is done to efficiently maintain 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 ones (the computation of contour hierarchy and orientation) can be performed simultaneously. As part of sweeping the plane, each cell in the arrangement is given attributes that indicate whether it lies in the material or non-material regions of each of the slices.

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

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

Theorem 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 the 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 non-material 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

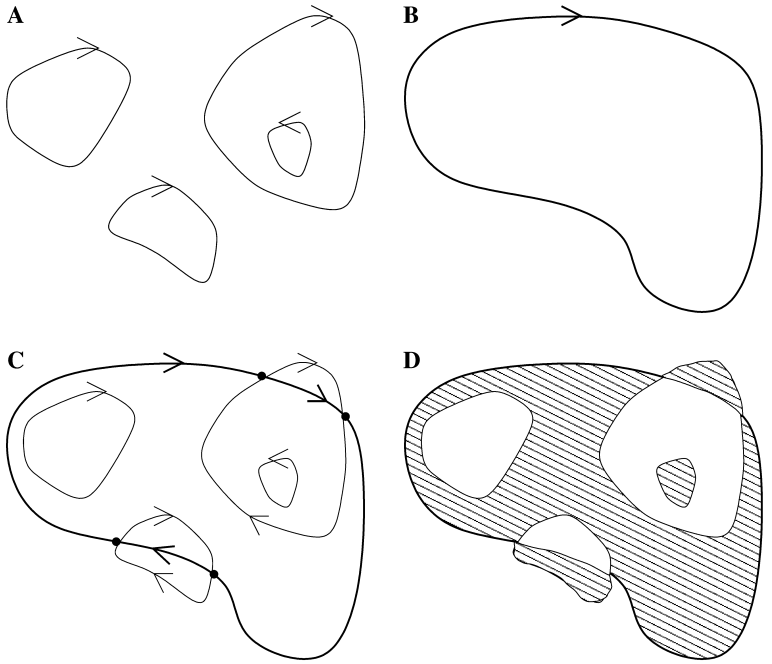

Fig. 1. Active cells in the overlay of two slices: (A) upper slice, (B) lower slice, (C) slices overlay, and (D) active cells.

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 Fig. 1D. 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 by 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

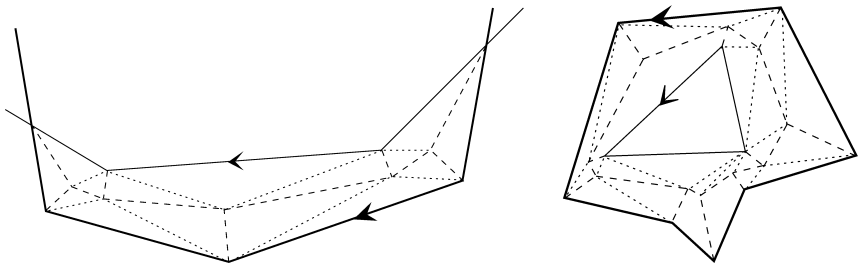

Fig. 2. Active cells: their straight skeletons and triangulations.

contains exactly one original polygon edge, and the face is monotone with respect to that edge [1, Lemma 3]. Then, among all possible triangulations of the face, we choose a triangulation that is monotone with respect to the polygon edge (see [9, pp. 55–58]).[3] Fig. 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 skeletal faces of 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 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, which refers to active cells bounded by portions of contours of both slices, will help us in doing so. The term "offset distance" stands here for the amount a polygon needs to be offset inward in order to hit a point.

Theorem 2. *Let H denote the vertices of the straight skeleton (of some active cell) that are offset-equidistant from contours of both slices. Then the straight skeleton of the cell is the union of*:
(1) *edge-disjoint trees whose roots are points in H and*
(2) *skeletal edges connecting pairs of points in H.*

[3] Although Lemma 3 of [1] guarantees that the face is monotone with respect to its defining edge, the endpoints of the edge are not necessarily visible by all vertices of the face. (That is, both chords connecting a vertex of the face to the two edge endpoints penetrate to the outside of the face.) Thus, we either compromise the monotonicity of the triangulation (where this is unavoidable), or enforce it by introducing Steiner points along the edge. The latter operation at most doubles the number of vertices of the face, so asymptotically it does not affect the complexity of the algorithm.

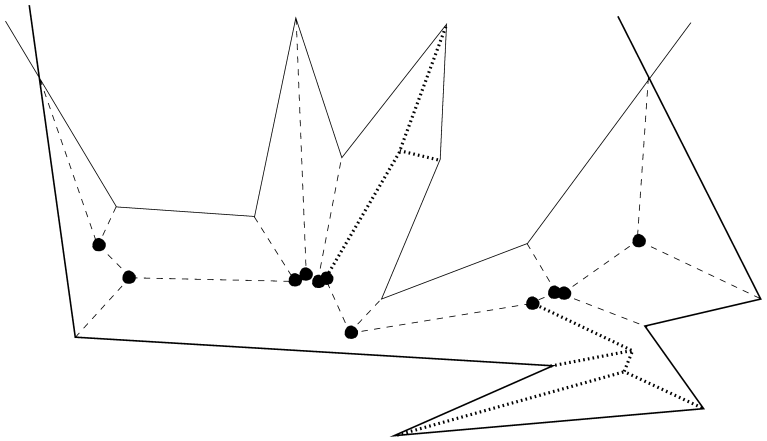

Fig. 3. Trees in the skeleton, rooted at points (shown as black disks) equidistant from contours of the two slices.

This claim is shown in Fig. 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 non-trivial trees are highlighted. Segments connecting points in H, and the trivial trees (line segments) are shown with dashed lines.

Proof. The claim follows from the properties of the straight skeleton of a simple polygon. The skeleton itself is a tree. Removing Q, the connected set of all edges connecting skeleton vertices that are offset-equidistant from the polygon, cuts the skeleton into edge-disjoint disconnected subtrees, and each such subtree touches Q at a single vertex—the subtree's root.

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 that are equidistant from points on contours of both slices. We set the height of these vertices to 0.5.
   (b) Skeleton vertices that are not equidistant from points on contours of the two slices. According to Theorem 2, 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 [4]) 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. Fig. 4 shows a close-up of the skeleton tree at the bottom-right corner of Fig. 3. The height of the vertices u, v, and w is 0 since they belong to the lower slice.

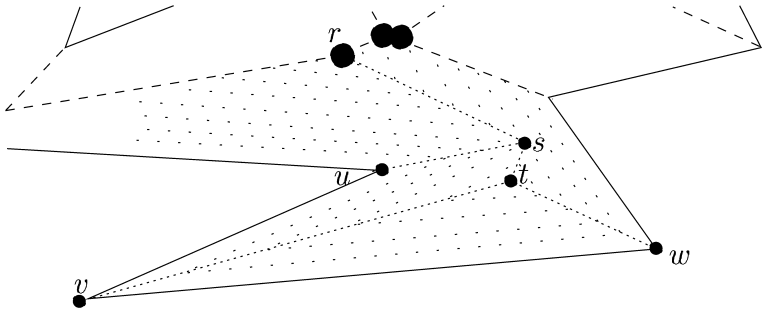

Fig. 4. Setting vertex heights according to their offsets.

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.[4] 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 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 match 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 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. Fig. 5A shows the xy projection of two slices, with a contour-intersection point v. Fig. 5B shows the same scene in a perspective view. The point v is actually the xy projection of three

[4] The values 3/9 and 5/9 follow from the fact that the skeleton vertices s and t are located 3 and 5, respectively, inward-offset units away from the polygon, while 9 units are needed to reach the root r. 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 Fig. 4 were 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.

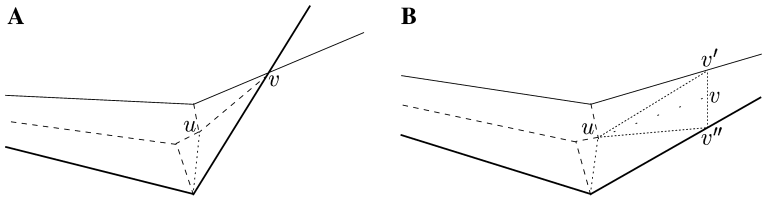

Fig. 5. Triangles sharing a contour-intersection point: (A) top view and (B) isometric view.

points v, v', and v'' (at heights 0.5, 1, and 0, respectively). The two supposed triangles in Fig. 5A that share the vertex u are actually quadrangles, as is seen in Fig. 5B. Therefore, we triangulate them by adding two more 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.

5.3. Surface orientation

Orienting the triangles consistently is rather easy. We assume that by convention original material (resp., non-material) contours are oriented clockwise (resp., counterclockwise). First, we invert all the polygons of one of the slices, say, the upper slice. It is easily seen that all the cells of the symmetric difference of the slices are collected as directed polygons. Cells in the upper (resp., lower) slice but not in the lower (resp., upper) slice appear counterclockwise (resp., clockwise). Then, triangulating the skeletal cells is performed consistently with these orientations. Finally, we do not alter the orientations of the triangles when we lift them up to three dimensions. This results in an oriented triangulated manifold, in which all triangles are oriented clockwise when they are viewed from the outside of the solid bounded by the union of the original polygons and the interpolating surface.

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.

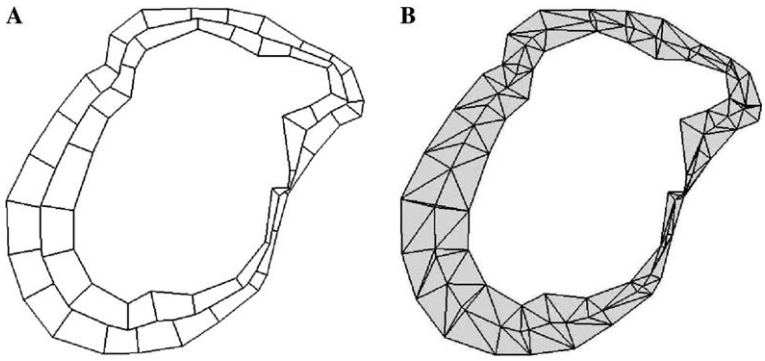

Fig. 6. A simple interpolation case: (A) contours and skeleton and (B) triangulated active cell.

Computing and analyzing the overlay of the two slices takes $O(n \log n + k)$ time [3]. 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 $\varepsilon > 0$ [17],[5] 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 [13]. However, we implemented the algorithm of [18] whose running time is $O(k^2)$ in the worst case, and in practice, much less than that. (Our experiments suggest that this step is subquadratic in n.) 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).

The space complexity of the algorithm is naturally $\Theta(n + k)$, since the number of internal skeleton vertices is linear in the number of vertices of the overlay of the polygons. In the worst case this quantity is $\Theta(n^2)$, but in practice it is proportional to n.

7. Experimental results

We implemented the entire algorithm in C++ on an HP Omnibook 6000 (a portable computer). 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

[5] 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.

fourth authors, took about 2 months, and the software consisted of about 8500 non-comment lines of code. We experimented with the algorithm on several data files obtained by medical scanners, and obtained very good results in practically all cases.

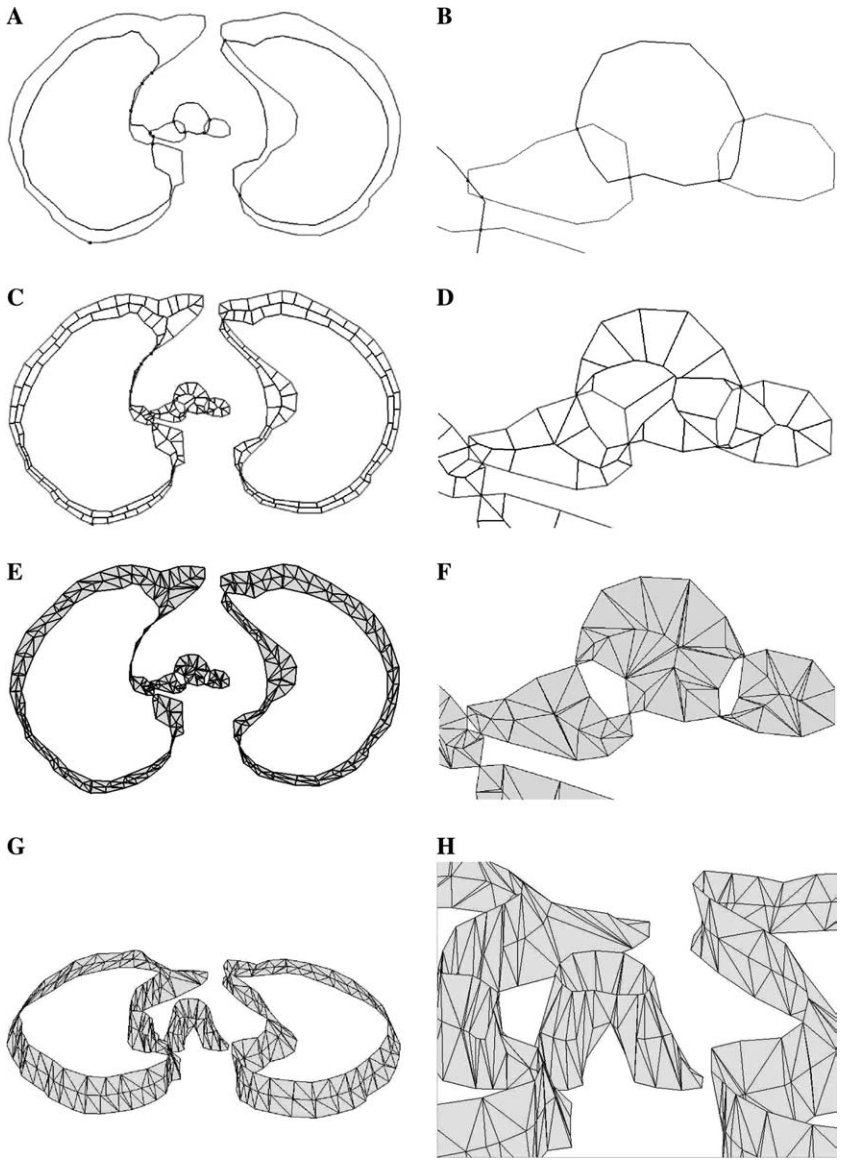

Fig. 7. A complex branching example: (A) overlay of slices, (B) close-up of (a), (C) straight skeletons of active cells, (D) close-up of (c), (E) triangulated skeletons, (F) close-up of (e), (G) a perspective view of the interpolation, and (H) close-up of (g).

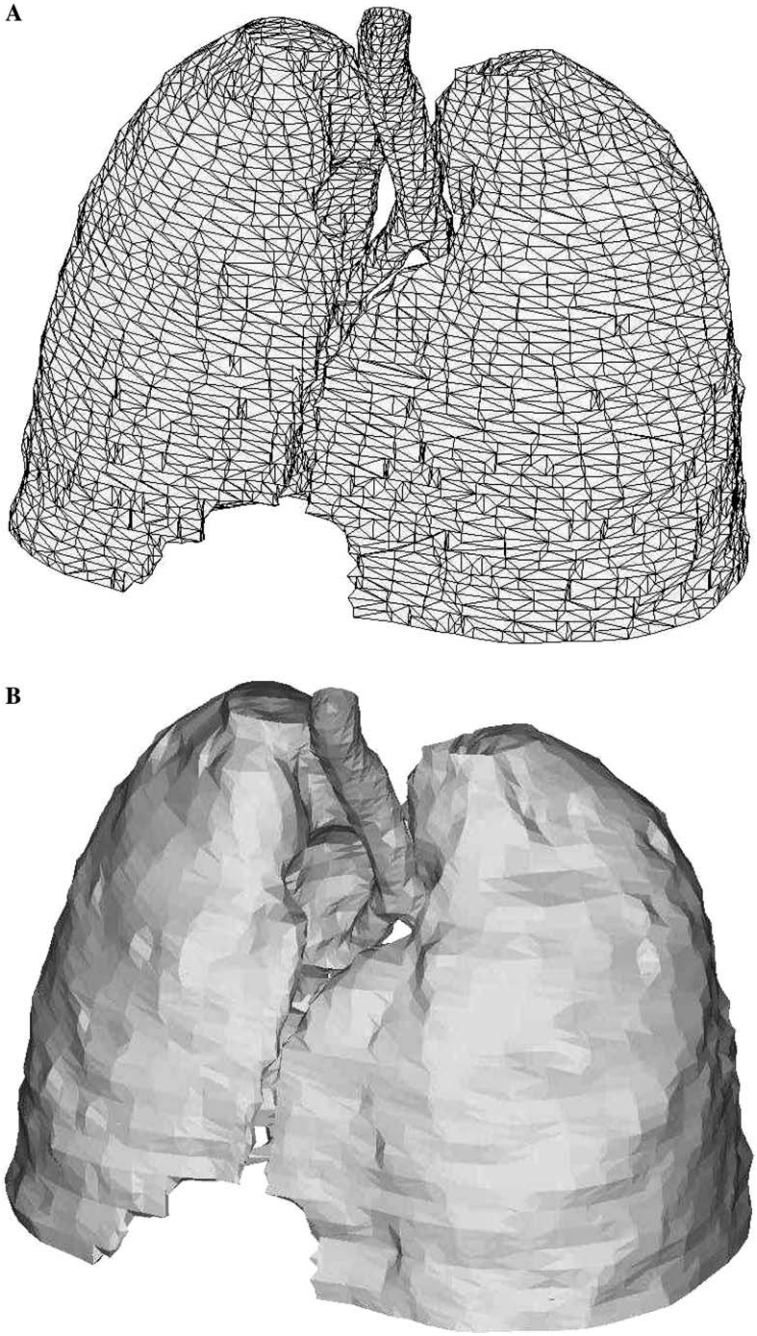

Fig. 8. A fully reconstructed pair of lungs: (A) wire-frame and (B) shaded.

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

Fig. 6A 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. Fig. 6B shows the triangulated ring.

Fig. 7A 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 Fig. 7B. Fig. 7C shows in thick black lines the straight skeletons of all the active cells in the arrangements of contours of the two slices. Similarly, Fig. 7D is a close-up of Fig. 7C. Fig. 7E shows a top-down view of the triangulated skeletons. A close-up of their middle area is shown in Fig. 7F. Figs. 7G and H show a perspective view and a close-up of the surface interpolated between the two slices.

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

Slice No.	Numbers of			Time (s)					
	Contour edges	Skeleton edges	Surface triangles	Line sweep	Sym. 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	5350	13,391	18,812	0.33	0.58	25.77	0.20	1.12	28.00

Figs. 8A and B show a wire-frame and a shaded display of the fully reconstructed pair of lungs. These data contained 30 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 Fig. 9, which shows a few relations between running times and output size to complexities of the input. (The displayed

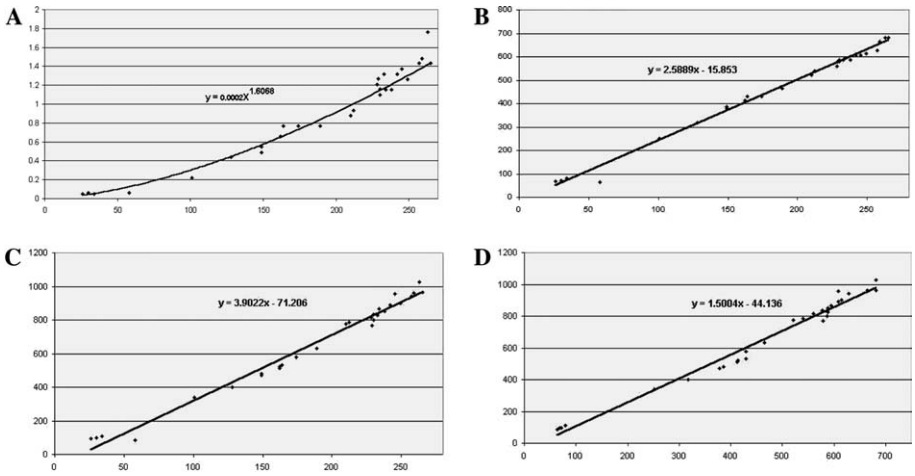

Fig. 9. (A–C) Skeleton creation time, number of skeleton edges, and number of output triangles, respectively (in the lungs model), as functions of the number of input contour edges; (D) Number of output triangles as a function of intermediate skeleton edges.

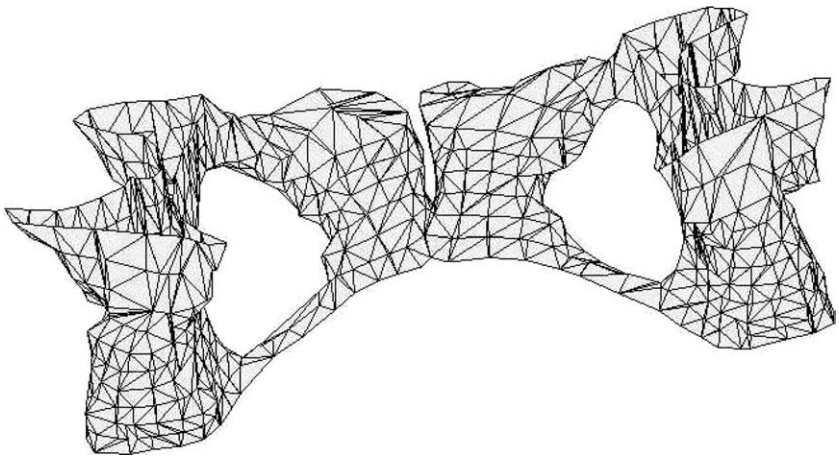

Fig. 10. A reconstructed pelvis.

functions were approximated by the curve-fitting tool of Microsoft Excel.) Overall, every layer was interpolated on average in less than one second.

Fig. 10 shows a reconstruction of part of a human pelvis.

8. Conclusion

In this paper we have proposed an algorithm for solving the practical problem of polyhedral interpolation between parallel polygonal slices, a problem that has many applications in medical imaging.

Our solution is based on computing the symmetric difference of the two slices, then computing the straight skeleton of each cell of the difference, triangulating each face in the skeletal map, and finally lifting the triangles up to three dimensions to heights guided by the skeletons.

We feel that our technique reconstructed the boundary of various organs in an intuitively appealing manner. The results were more than adequate even in extreme cases of tiling between two seemingly totally different slices. The use of straight skeletons may create undesired long peaks at the vicinity of sharp corners of the input polygons. To remedy this we can use, instead of the straight skeleton, a linear approximation of the medial axis, or even the chordal axis, of the cells of the symmetric difference. We leave this as future work.

Acknowledgments

We wish to thank Evgeny Yakersberg who pointed out the fact that it is not always possible to triangulate a face of the straight skeleton of a polygon such that all triangles share the polygon's edge that defines the skeletal face.

References

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

[2] C.L. Bajaj, E.J. Coyle, K.N. Lin, Arbitrary topology shape reconstruction from planar cross sections, Graph. Models Image Process. 58 (1996) 524–543.

[3] I.J. Balaban, An optimal algorithm for finding segment intersections, in: Proceedings of the 11th Annual ACM Symposium on Computational Geometry, Vancouver, British Columbia, Canada, 1995, pp. 211–219.

[4] G. Barequet, M.T. Dickerson, M.T. Goodrich, Voronoi diagrams for polygon-offset distance functions, Discrete Comput. Geom. 25 (2001) 271–291.

[5] G. Barequet, M.T. Goodrich, A. Levi-Steiner, D. Steiner, Straight-skeleton based contour interpolation, in: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, 2003, pp. 119–127.

[6] G. Barequet, D. Shapiro, A. Tal, Multilevel sensitive reconstruction of polyhedral surfaces from parallel slices, Visual Comput. 16 (2000) 116–133.

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

[8] S. Batnitzky, H.I. Price, P.N. Cook, L.T. Cook, S.J. Dwyer III, Three-dimensional computer reconstruction from surface contours for head CT examinations, J. Comput. Assisted Tomography 5 (1981) 60–67.
[9] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, Computational Geometry: Algorithms and Applications, Springer-Verlag, Germany, 1997.
[10] J.D. Boissonnat, Shape reconstruction from planar cross sections, Comput. Vision Graph. Image Process. 44 (1988) 1–29.
[11] J.D. Boissonnat, B. Geiger, Three dimensional reconstruction of complex shapes based on the Delaunay triangulation, Technical Report 1697, Inria-Sophia Antipolis, 1992.
[12] S.W. Cheng, T.K. Dey, Improved construction of Delaunay based contour surfaces, in: Proceedings of the 5th ACM Symposium on Solid Modeling and Applications, Ann Arbor, MI, 1999, pp. 322–323.
[13] S.-W. Cheng, A. Vigneron, Motorcycle graphs and straight skeletons, in: Proceedings of the 13th ACM/SIAM Symposium on Discrete Algorithms, San Francisco, CA, 2002, pp. 156–165.
[14] Y.K. Choi, K.H. Park, A heuristic triangulation algorithm for multiple planar contours using an extended double branching procedure, Visual Comput. 10 (1994) 372–387.
[15] H.N. Christiansen, T.W. Sederberg, Conversion of complex contour line definitions into polygonal element mosaics, Comput. Graph. 13 (1978) 187–192.
[16] A.B. Ekoule, F.C. Peyrin, C.L. Odet, A triangulation algorithm from arbitrary shaped multiple planar contours, ACM Trans. Graph. 10 (1991) 182–199.
[17] D. Eppstein, J. Erickson, Raising roofs, crashing cycles, and playing pool: applications of a data structure for finding pairwise interactions, Discrete Comput. Geom. 22 (1999) 569–592.
[18] P. Felkel, Š. Obdržálek, Straight skeleton computation, in: L. Szirmay-Kalos (Ed.), Spring Conference on Computer Graphics, Budmerice, Slovakia, 1998, pp. 210–218.
[19] P. Felkel, Š. Obdržálek, Improvement of Oliva's algorithm for surface reconstruction from contours, in: J. Zara (Ed.), Spring Conference on Computer Graphics, Budmerice, Slovakia, 1999, pp. 254–263.
[20] J.D. Fix, R.E. Ladner, Multiresolution banded refinement to accelerate surface reconstruction from polygons, in: Proceedings of the 14th Annual ACM Symposium on Computational Geometry, Minneapolis, MN, 1998, pp. 240–248.
[21] H. Fuchs, Z.M. Kedem, S.P. Uselton, Optimal surface reconstruction from planar contours, Commun. ACM 20 (1977) 693–702.
[22] S. Ganapathy, T.G. Dennehy, A new general triangulation method for planar contours, ACM Trans. Comput. Graph. 16 (1982) 69–75.
[23] L. Guibas, J. Stolfi, Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams, ACM Trans. Graph. 4 (1985) 74–123.
[24] N. Kehtarnavaz, R.J.P. De Figueiredo, A framework for surface reconstruction from 3D contours, Comput. Vision Graph. Image Process. 42 (1988) 32–47.
[25] N. Kehtarnavaz, L.R. Simar, R.J.P. De Figueiredo, A syntactic/semantic technique for surface reconstruction from cross-sectional contours, Comput. Vision Graph. Image Process. 42 (1988) 399–409.
[26] E. Keppel, Approximating complex surfaces by triangulation of contour lines, IBM J. Res. Dev. 19 (1975) 2–11.
[27] D. Meyers, S. Skinner, K. Sloan, Surfaces from contours: the correspondence and branching problems, Proc. Graph. Interface (1991) 246–254.
[28] J.-M. Oliva, M. Perrin, S. Coquillart, 3D reconstruction of complex polyhedral shapes from contours using a simplified generalized Voronoi diagram, Comput. Graph. Forum 15 (1996) C397–408.
[29] M. Shantz, Surface definition for branching contour-defined objects, Comput. Graph. 15 (1981) 242–270.
[30] K.R. Sloan, J. Painter, Pessimal guesses may be optimal: a counterintuitive search result, IEEE Trans. Pattern Anal. Mach. Intell. 10 (1988) 949–955.
[31] Y.F. Wang, J.K. Aggarwal, Surface reconstruction and representation of 3-D scenes, Pattern Recogn. 19 (1986) 197–207.
[32] E. Welzl, B. Wolfers, Surface reconstruction between simple polygons via angle criteria, J. Symbolic Comput. 17 (1994) 351–369.
[33] M.J. Zyda, A.R. Jones, P.G. Hogan, Surface construction from planar contours, Comput. Graph. 11 (1987) 393–408.