Straight-Skeleton Based Contour Interpolation*

Gill Barequet† Michael T. Goodrich‡

Abstract

In this paper we present a novel yet simple 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 guaranteed to produce a nonselfintersecting surface in all cases regardless of the number of contours in each slice, their complexity and geometry, and the height of their hierarchy of nesting. The method is based on a simple analysis of the overlay of the two slices and on the choice of some of the cells. Then, the straight skeletons of the selected cells guide the triangulation of these cells. Finally, the resulting triangles are lifted up to space to form an interpolating surface.

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. The main motivations for this problem come from 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 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 is 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, BPCC81, GD82, WA86, KD88, KSD88,

*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 and PHY-9980044.

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

‡Center for Algorithm Engineering, Dept. of Computer Science, Johns Hopkins University, Baltimore, MD 21218. E-mail: `goodrich@cs.jhu.edu`

SP88, WW94, FL98]). 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., selfintersecting surfaces). Some algorithms [CS78, Sh81, ZJH87, EPO91, 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, 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. To the best of our knowledge, this is the first contour-interpolation algorithm that is based on the straight skeletons of cells of the overlay of slices.

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. The union of the lifted-up triangulations of all the chosen cells is the output surface.

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.

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 contours 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, triangulate separately 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

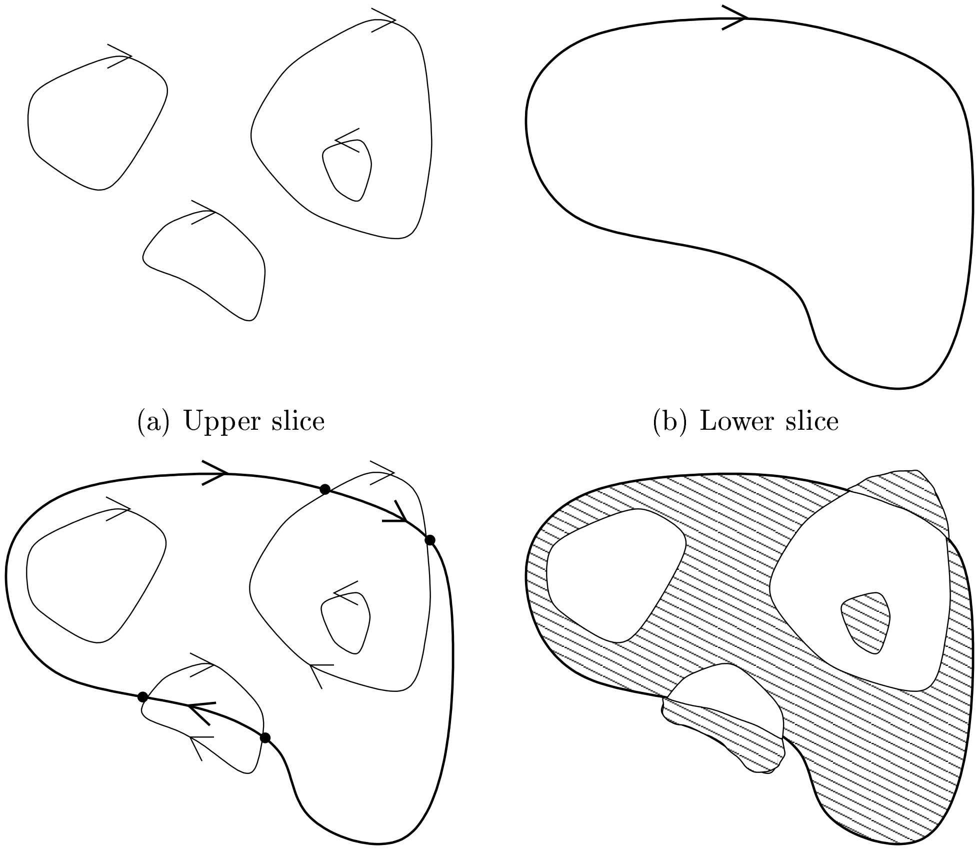
The data consists of a sequence of slices, all in the same file. Each slice consists of a hierarchy of contours, that is, a forest of closed simple polygons with nonintersecting boundaries, where a parent polygon fully encloses all its children, and no other contour is enclosed in the parent polygon 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 each contour, when it is viewed above, the material lies to the right of it. Thus all even-level contours are oriented in the clockwise direction, when viewed from above, and all odd-level contours are oriented in the 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 described by Guibas and Stolfi [GS85]. This is done for maintaining efficiently the constructed polyhedral boundary of the interpolating solid object.

4 Analyzing the Contours 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.



(a) Upper slice (b) Lower slice

(c) Slices overlay (d) Active cells

Figure 1: Active cells in the overlay of two 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 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 are vertices of the arrangement and whose interior is free of such vertices.

Theorem 1 *Each contour portion belongs to exactly one active cell.*

Proof: Refer to Figure 2. We show the claim, for example, for the contour portion AB that belongs to the lower slice (shown with thick lines) and delimited by the points A and B. The same argument holds for every contour portion. Obviously, no other contour of any of the two slices intersects AB, for otherwise AB would not be a contour portion by definition. Thus, the material of the lower slice lies on one side of AB, whereas the nonmaterial of the lower slice lies on the other side of AB. Our goal is to show that both sides of AB (in a

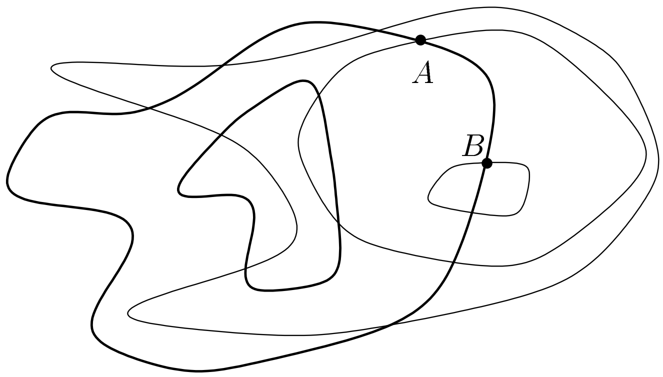


Figure 2: Every contour portion is adjacent to exactly one active cell

small-enough neighborhood) belong to either the material or the nonmaterial region of the upper slice. But this is obvious, since in the planar map induced by only the upper slice AB does not intersect (except in its endpoints) any polygon, and hence it lies completely in one cell of the map, which is either a material or a nonmaterial region of the upper slice. Thus, exactly one of the two cells in the overlay of the two slices, that share AB on their boundaries, belongs to the material of one slice and to the nonmaterial of the other slice. Hence AB is on the boundary of exactly one active cell.

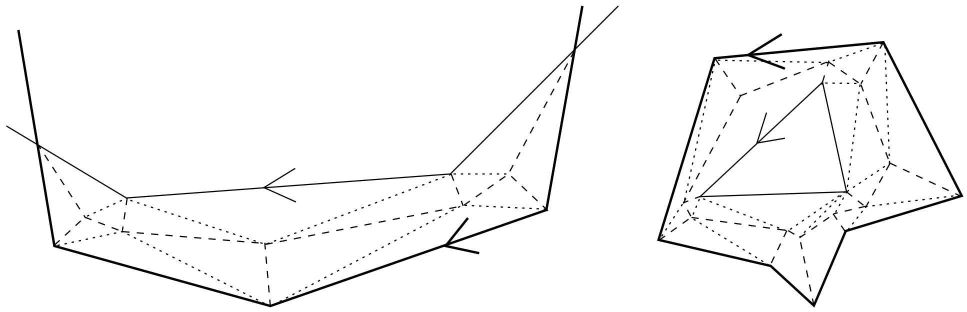
To complete the argument, we show the claim for the case in which the contour portion is an entire contour (denoted as c) that bounds a cell in the overlay of the slices. This contour is either a material or a nonmaterial contour in one of the slices, and it is fully contained in either the material or the nonmaterial region of the other slice, for otherwise a contour (or contours) of the other slice would intersect c. In all four combinations exactly one of the two cells bounded by c (either the inner or outer side of it) is active, while the other cell is inactive. □

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 nonselfintersecting 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.



(a) (b)

Figure 3: The straight skeletons and their triangulations of active cells

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 which is monotone with respect to the polygon edge (see [BKOS97, pp. 55–58]). Figure 3 shows two examples 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 motivate this choice of triangulation by its contribution to the intuitiveness of the reconstructed 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 lemma will help us in doing so.

Theorem 2 *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 4. 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.

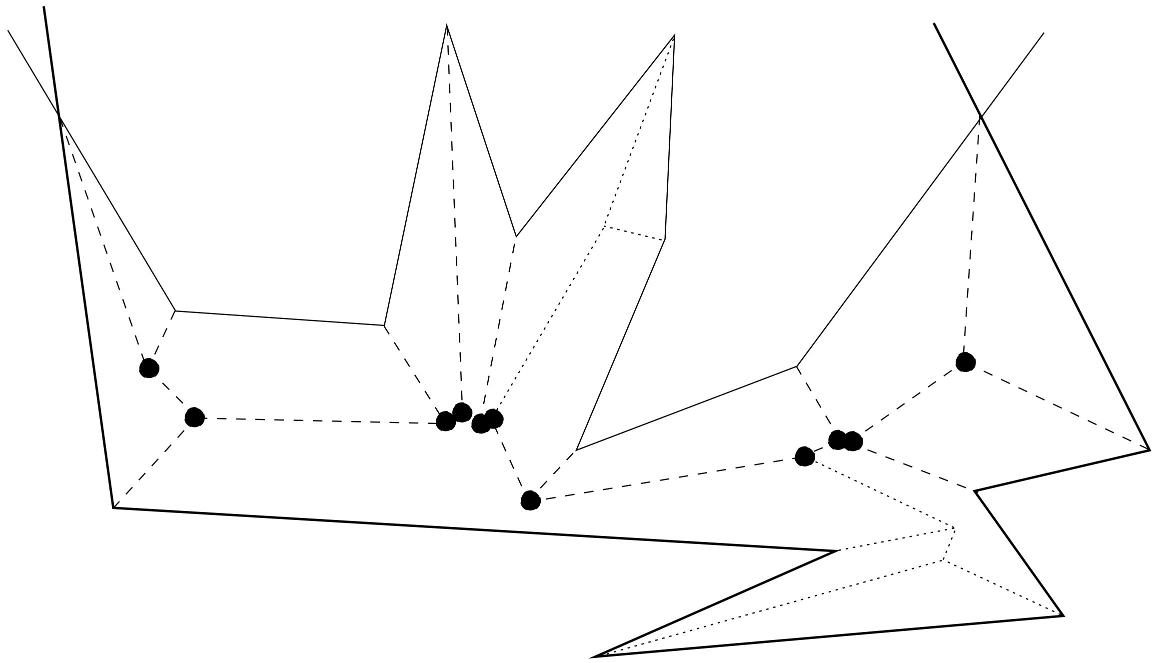


Figure 4: Trees in the skeleton, rooted at points equidistant from contours of the two slices

Proof: Omitted in this short version of the paper. □

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 two 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 only one slice. 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 [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 5 shows a close-up of the skeleton tree at the bottom-right corner of Figure 4. 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

[1] Obviously we must take care that the height of isolated branches of the tree will not exceed 1. For example, if the vertex t in Figure 4 were twice further than r (offset-wise) from the boundary of the lower

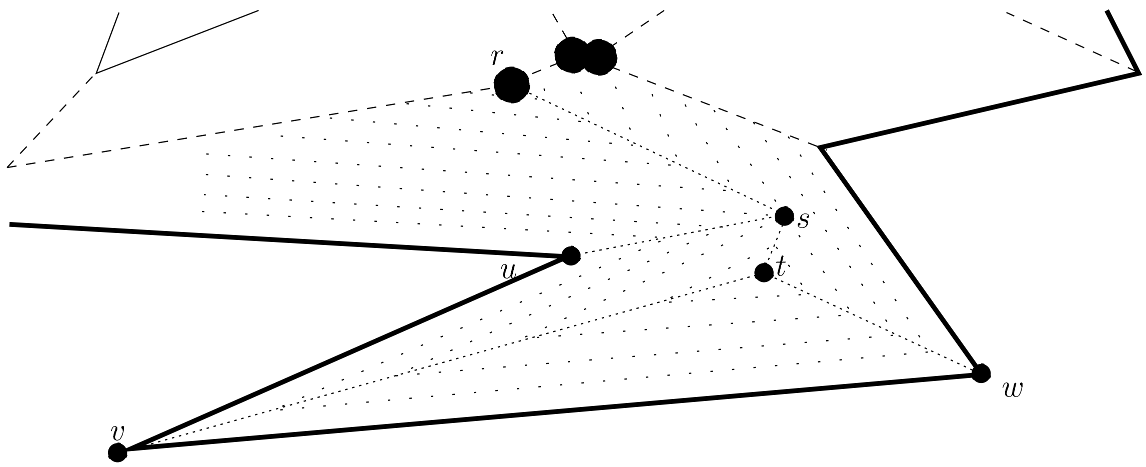
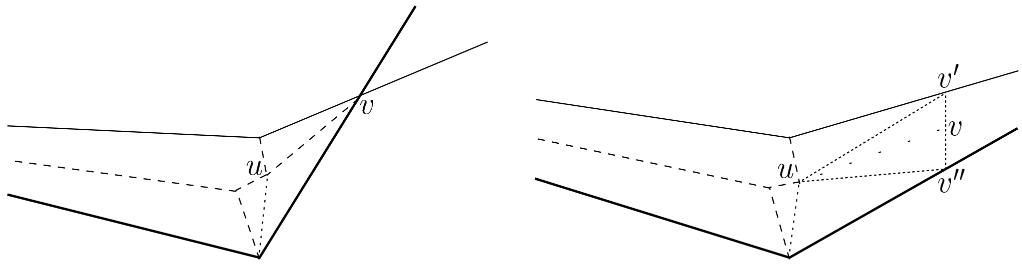


Figure 5: Setting vertex heights according to their offsets



(a) Top view

(b) Isometric view

Figure 6: Triangles sharing a contour-intersection point

fits our application due to the strong relation between the offset of a shape and its straight skeleton.

3. Intersection points of pairs of contours, one of each slice. In fact, such a point is actually a triple of 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 intersection point of their xy projections. Figure 6(a) shows the xy projection of two slices, with a contour-intersection point v. Figure 6(b) shows the same scene from a slightly 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 seemingly triangles in Figure 6(a) that share the vertex u are actually quadrangles, as is seen in Figure 6(b). We therefore 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).

slice, which is possible since the contours are not necessarily convex, then its height would be more than 1. Our solution is to apply a secondary scaling, if needed, on every isolated subtree.

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 sought 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 takes $O(n + k)$ time, as well as triangulating the monotone subcells induces by the skeletons. Lifting the triangulations up to three dimensions to form the interpolating surfaces takes also $O(n + k)$ time.

To conclude, the entire algorithm runs in $O(n \log n + k)$ time. In the worst case this is $\Theta(n^2)$, but for most practical cases the running time of the algorithm is $O(n \log n)$.

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, *Proc. 5th Workshop on Algorithms and Data Structures*, Halifax, Nova Scotia, Canada, *Lecture Notes in Computer Science*, 1272, Springer-Verlag, 200–209, August 1997; full version to appear in *Discrete & Computational Geometry*.

[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.

[BPCC81] 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. of Computer Assisted Tomography*, 5 (1981), 60–67.

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

[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.

[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.

[EPO91] A.B. Ekoule, F.C. Peyrin, and C.L. Odet, A triangulation algorithm from arbitrary shaped multiple planar contours, *ACM Transactions on Graphics*, 10 (1991), 182–199.

[FL98] J.D. Fix and R.E. Ladner, Multiresolution banded refinement to accelerate surface reconstruction from polygons, *Proc. 14th Ann. ACM Symp. on Computational Geometry*, 240–248, 1998.

[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 Transactions on Computer Graphics*, 16 (1982), 69–75.

[GS85] L. Guibas and J. Stolfi, Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams, *ACM Transactions on Graphics*, 4 (1985), 74–123.

[KD88] 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.

[KSD88] 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.

[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 Transactions on Pattern Analysis and Machine Intelligence*, 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.