Nonlinear Interpolation between Slices

Gill Barequet[*]
Dept. of Computer Science
The Technion—IIT
Haifa 32000

Amir Vaxman[*]
Dept. of Computer Science
The Technion—IIT
Haifa 32000

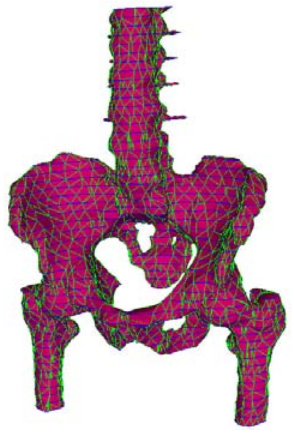


Figure 1: A reconstructed pelvis

Abstract

The topic of interpolation between slices has been an intriguing problem for many years, as it offers means to visualize and investigate a three-dimensional object given only by its level sets. A slice consists of multiple non-intersecting simple contours, each defined by a cyclic list of vertices. An interpolation solution matches between a number of such slices (two or more at a time), providing means to create a closed surface connecting these slices, or the equivalent morph from one slice to another.

We offer a method to incorporate the influence of more than two slices at each point in the reconstructed surface. We investigate the flow of the surface from one slice to the next by matching vertices and extracting differential geometric quantities from that matching. Interpolating these quantities with surface patches then allows a nonlinear reconstruction which produces a free-form, non-intersecting surface. No assumptions are made about the input, such as on the number of contours in each slice, their geometric similarity, their nesting hierarchy, etc., and the proposed algorithm handles automatically all branching and hierarchical structures. The resulting surface is smooth and does not require further subdivision measures.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems;

Keywords: Interpolation between slices, surface reconstruction, parallel cross-sections, G^1-continuity, vertex matching, Gregory patches

[*]e-mail: [barequet,avaxman]@cs.technion.ac.il

1 Introduction

1.1 Surface Reconstruction

Reconstruction of a surface, either polyhedral or free-form, from a set of consecutive parallel cross-sections, has many applications and, therefore, attracted a substantial amount of work since the 70's. The input to such a problem is usually a set of slices containing contours that represent some type of data: A general volumetric object scanned at its level sets, geographic height lines, MRI slices, CT scans, and more. The problem can be formally defined thus: Given a set of n consecutive slices $\{L_0, L_1, \ldots, L_{n-1}\}$, each composed of closed simple contours (that can be nested but otherwise not intersecting), and that are located in the (x, y, z_i) plane for a constant z_i, $0 \le i \le n-1$, construct the boundary surface S of an object O, such that $S(x, y, z = z_i) = L_i$. This problem is ill-posed and underconstrained; therefore, infinite solutions exist (unless we impose some restrictions on the output surface), and one must set up a heuristic in order to constrain the problem to a unique solution. This reconstruction poses some characteristic issues to be dealt with:

- **Correspondence**: How to match vertices of different slices in order to create a consistent non-self-intersecting mesh. This issue also involves branching ("one-to-many" correspondence) and dealing with geometric dissimilarities.
- **Optimality**: Creating the best surface, in terms that are subjective, but well-defined for each solution.

- **Robustness**: Can a method be fully automatic, or should it make prior assumptions on the input slices. In addition, should an algorithm rely on the user for tuning of parameters.

2 Previous Work

Most of the work done so far on this subject offered solutions for piecewise-linear reconstruction between two adjacent slices. Only a few early works [Bajaj et al. 1996; Barequet and Sharir 1996; Boissonnat 1988] offer a solution to the problem in full generality. Recent works [Barequet et al. 2004; Oliva et al. 1996] suggest using a novel type of a polygonal skeleton, called the *straight-skeleton* (introduced in [Aichholzer et al. 1995]), in order to solve the problem of reconstructing a surface with multiple contours of arbitrary geometry. The works [Barequet et al. 2004; Yakersberg 2004] base their solutions on this skeleton to create a general algorithm for reconstruction, imposing no constraints on the input. They utilize the straight-skeleton computation algorithm offered by [Felkel and Obdržálek 1998]. In [Ju et al. 2005], A method is devised for the interpolation of curves using a global volume graph, For which user-tuning can produce better results.
A few works deal with the problem of a general reconstruction, incorporating more than two slices at a time. In [Barequet et al. 2000], a piecewise-linear solution is offered, considering the slopes of triangles from neighboring layers. The focus of most recent works is on PDE's or level-set solving methods for creating a smooth surface [Chai et al. 1998; Cong and Parvin 2001; Hormann et al. 2003; Nilsson et al. 2005; Wang et al. 2006], providing means to match contours, or to morph between adjacent curves. In order to create a smooth surface from a given reconstruction, some works utilize mesh smoothing techniques [Wang et al. 2006; Yakersberg 2004], some works offer to build iso-surfaces, a volumetric representation directly [Cong and Parvin 2001; Nilsson et al. 2005], or an implicit surface [Akkouche and Galin 2004]. Other works resort to general free-form interpolation techniques, a great deal of which is summarized in the survey [Mann et al. 1992]. Some of these methods (e.g., [Chai et al. 1998; Hormann et al. 2003]) offer to output free-form surfaces, yet they use intermediate contours and address a terrain reconstruction, which is a special case of the slice-interpolation problem. We are not aware of any general method that creates an explicit continuous surface locally (e.g. without have to solve a set of global constraints) such as the one presented in this paper.

3 Overview of Our Method

3.1 Our Contribution

We use a robust matching technique between vertices of the original contours and intersection vertices of their overlays, based on the computation of straight-skeletons of the cells of the symmetric difference of two consecutive slices at a time (as classified by [Barequet et al. 2004]). Thus, we inherit the full generality of this work, without intermediate contour interpolation, which introduces linearity, and extra complexity to the sought surface.

This matching yields a graph representing the flow of the surface, from which we extract first-order data and to which we fit a cubic curve network, and then G^1-continuous surface Gregory patches, comprising the desired surface. We prove that the straight-skeleton-based matching works for any instance of the problem, and thus the algorithm works for any set-up of contours. The acquiring of first-order data requires the incorporation of more than two slices at each vertex, and, thus ,this algorithm produces a smooth surface even when there are abrupt changes between slices.

3.2 Properties of The Output Surface

Our algorithm yields a three-dimensional surface which satisfies the following conditions:

- The surface interpolates the vertices of the cross-sections. Therefore, a cross-section of the surface in the appropriate plane is a planar smooth interpolation of the vertices.
- The interpolated surface consists of surface patches, constrained with tangent-plane continuities between neighboring patches, rendering the surface to be (at least) G^1-continuous, closed non-self-intersecting, 2-manifold (with a proper orientation).

3.3 Outline of the Algorithm

The algorithm proceeds with the following steps (exemplified in Figure 2) :

1. Reading the sequence of input slices.
2. Computing the overlay of each pair of consecutive slices. The cells of the symmetric difference are identified and their straight-skeletons are computed (Using the algorithm of [Felkel and Obdržálek 1998]).
3. Based on the structure of the straight skeleton of each cell, the vertices comprising it are matched. This step creates a matching graph (the so-called *flow graph*), in which all connections (arcs) are between a slice to itself or to its adjacent slices. The matching algorithm is described in Section 4.
4. An orientation is imposed on the surface, considering the flow graph as an underlying mesh to the complete surface (in which faces are not necessarily planar). This procedure is explained in Section 5.
5. First-order quantities (tangent planes, used later for G^1 constraints) of the sought surface are computed from the flow graph at the vertices. This step is described in Section 6.
6. A network of cubic Bézier curves is constructed from the arcs of the flow graph, as shown in Section 7.
7. Gregory surface patches are constructed from the faces of the flow graph, interpolating the curve network. This step completes the algorithm and produces the final surface. The construction is detailed in Section 8.

4 Creating the Flow Graph

We reduce the problem of constructing a flow graph FL, which is a structure describing a matching between all the vertices in the scene, to creating local two-dimensional matching graphs on the overlay of a pair of consecutive slices, and then lifting these two-dimensional graphs to the three-dimensional space, so that the flow graph comprises their union. We now show how to match vertices in an overlay.

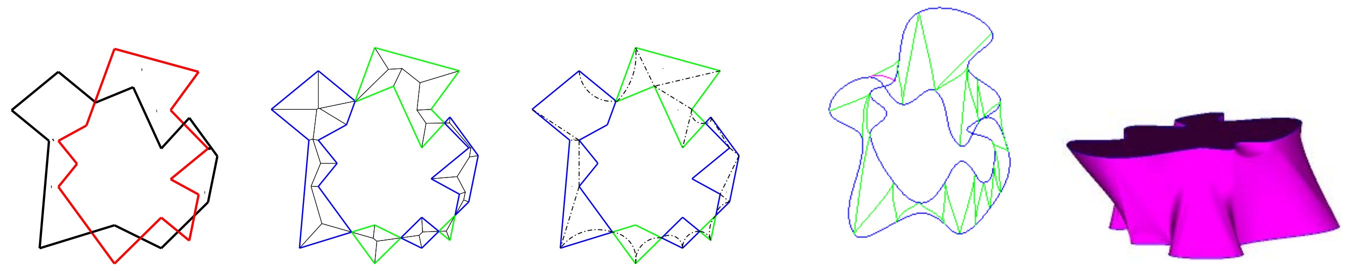

(a) Overlay of two slices. (b) Active cells and straight skeletons. (c) The matching of vertices (shown by dashed-dotted arcs). (d) The curve network (e) The reconstructed surface

Figure 2: An example of surface reconstruction from two slices.

4.1 Straight Skeletons

The straight-skeleton is a partition of a simple polygon (Which may contain simple holes), defined by propagating the edges of the polygon inwards at a fixed rate. The vertices slide along the angular bisectors of the edges sharing the vertices, and the skeleton is comprised of the union of their paths. Two major events[1] may occur during the propagation:

- **Edge Event.** An edge vanishes, and the two bisectors of its endpoints meet.
- **Split Event.** A reflex vertex runs into an opposite edge. At that point, the polygon splits as well.

The propagation stops when all intermediate polygons vanish (If the polygon is convex, there are no split events, and there is only one intermediate polygon). The straight-skeleton has a few important properties:

- Since every face of the skeleton is swept by a single edge, every face is monotone with respect to its defining edge.
- The complexity of the skeleton is linear in the number of original polygon edges.
- Theoretically, the computation of the skeleton can be done in subquadratic time [Eppstein and Erickson 1998]. In this work, we utilized the intuitive algorithm of [Felkel and Obdržálek 1998], whose running time is $O(N(r+\log N))$, Where N is the number of edges and r is the number of reflex (concave) vertices, which is $O(N)$.

4.2 Matching Vertices in Active Cells

We consider a pair of slices $\{L_i, L_{i+1}\}$, for which the overlay is computed. The active cells of the overlay are the set $\mathrm{SD}(i) = (L_i \setminus L_{i+1}) \cup (L_{i+1} \setminus L_i) = \{A_{i,1}, \ldots, A_{i,m}\}$, where m is the number of cells in the symmetric difference of L_i and L_{i+1} (see Figure 2). In relation to any overlay, and without loss of generality, we denote by L_i (resp., L_{i+1}) the "lower" (resp., "upper") slice. With that denomination, we distinguish between $U \setminus L$ cells (cells that are in the "material" zone of the upper slice alone) and similarly $L \setminus U$ cells.[2]

[1]Any other type of events can be seen as a degenerate sequence of these two events.

[2]This distinction is important when dealing with intersection vertices of the overlay (as is seen in this section), and surface orientation (as is seen in Section 5).

The algorithm proceeds with computing the straight skeletons of all the non-empty active cells (empty active cells are the result of overlapping portions of edges in the overlay), by using the algorithm proposed in [Felkel and Obdržálek 1998].[3] We assume that the active cells are in general position (i.e., there are no degeneracies in the skeleton, which can be regarded as multiple regular events by creating different skeletal nodes with zero-length edges connecting between them, and so no special treatment is needed for them). The matching is then performed on each active cell of the overlay, following the steps of the shrinking process creating the skeleton. The motivation for such a matching is that the straight skeleton serves as an indication of a vertex's proper mates (in the sense of closest neighbors when offsetting inwards) in an active cell, and by following the events of its creation the algorithm matches them naturally. The matching algorithm is as follows:

Input: An active cell $A_{i,j}$ in $SD(i)$, for which the skeleton $S(A_{i,j})$ is computed.

Output: A local matching represented by an graph M_{ij} defined on the vertices of $A_{i,j}$.

The Algorithm: A matching is created with every event of the skeleton creation, identified in [Felkel and Obdržálek 1998], following these rules:

1. An edge event makes a portion of an active cell edge e vanish. An edge of M_{ij} is created between the two original vertices comprising the portion of e at the point of vanishing. Should an original edge vanish without splitting, the matching edge would be created between the two original endpoints of e. Should several edges vanish at the same point, we match all the vertices of these edges in a Counterclockwise order. Figure 3 exemplifies this case.
2. A split event occurs between a reflex vertex (so denoted in [Felkel and Obdržálek 1998]) and a portion of an edge e. An edge of M_{ij} is created between the reflex vertex and the closest neighbor (by Euclidean distance) out of the two concurrent endpoints of that portion of e. Figure 3 exemplifies this event as well.

[3]We used this algorithm because of its straightforward approach. However, because of the unique shape of the skeleton faces (see [Yakersberg 2004]), simple adjustments can make this algorithm work on a precomputed straight skeleton, that can be computed by a faster algorithm. (such as the algorithm of [Eppstein and Erickson 1998]).

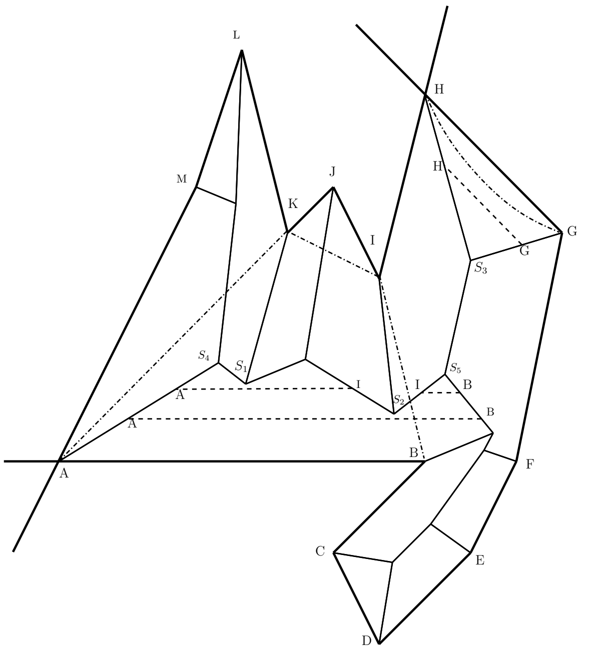


Figure 3: An example of some matchings following events in the active cell. The skeleton is shown by solid lines, the progression of cell edges by dashed lines, and the matching by dashed-dotted lines. The Matching GH is due to an edge event at S_3, while IB is actually two matching edges: a split event at S_2 (I splitting AB, B being the closest to I), followed by an edge event at S_5 of the edge part IB. KI is due to the event of K splitting AI at S_1, and KA is due to an edge event at S_4 of the edge part AS_1 (which is a split part of the original AB).

4.3 Properties of the Matching

Lemma 4.1. *M_{ij} is a connected planar graph.*

Proof. This claim is a direct consequence of the fact that the matching algorithm matches only pairs of vertices that meet during the offset process, or vertices and the concurrent endpoints of edges they split. Since the shrinking algorithm always creates non-intersecting intermediate polygons [Felkel and Obdržálek 1998], the traces of the movement of vertices along the process do not cross, considering that reflex vertices that split an edge can only be matched to other vertices that meet the same edge in the order of splitting (including original endpoints). Then, since a matching edge in M_{ij} is the union of two traces, all edges are non-crossing, and, therefore, M_{ij} is a planar graph.
Assume for contradiction that M_{ij} is not connected, i.e., there is at least one group of polygonal holes which vertices are only matched to other vertices among that group. Since every skeletal node, which is created by an event of the process, creates a matching, the straight skeleton created by the propagation of vertices of this group has to be disconnected from the outer boundary of the active cell, or from the other inner groups. This is impossible since the straight skeleton of any polygon is a single connected graph. Therefore, M_{ij} is connected. □

Lemma 4.2. *The straight segment that connects two matched vertices p,q does not intersect the boundary of their active cell.*

Proof. Assume for contradiction that the segment $pt+q(1-t), t \in [0,1]$ intersects with (at least) two edges e_1, e_2 of the boundary of the active cell, coinciding at a concave vertex r.[4] The construction of M_{ij} indicates that two vertices are matched if and only if they share the same intermediate polygon in the shrinking process until the event that matches them. Assume N_{pq} is the skeletal node and event that created m_{pq}, i.e., without loss of generality, the meeting point of bisector b_q and the edge e_p. By definition, this point is equidistant from the two edges e_p and e_q which initially coincide at p and q, respectively.[5] In any such configuration, the distances d_{e_1}, d_{e_2} are shorter than at least on of the distances d_{e_p}, d_{e_q}, and the propagation of r would eventually occlude e_p from e_q. Since the faces of the straight skeleton are monotone, that means that the match m_{pq} is not possible (See Figure 4(a) for an example). □

Theorem 4.1. *For any i, j, M_{ij} is non-intersecting (thus, M_{ij} is a planar geometric graph).*

Proof. If $A_{i,j}$ is a simple polygon (with no holes), then every matching edge partitions the polygon into two parts. M_{ij} is planar according to Lemma 4.1, which also implies that the graph is planar with all matching edges inside the polygon. Thus, when one matching edge, drawn as a straight-line, partitions the polygon (and does not intersect it, according to Lemma 4.2), there can be no other matching edges from a vertex on one side of the polygon to another in such a partition, since it contradicts the planarity. Therefore, all other matching edges are completely contained in either partition, and can be drawn as straight segments without any intersections of neither matching edges, nor the polygon boundary, and M_{ij} of a simple active cell is a planar geometric graph (Here, all the faces are simple by definition).

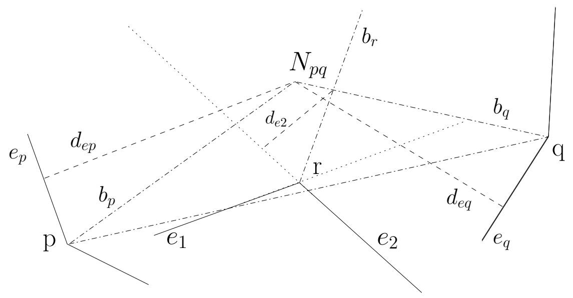
If $A_{i,j}$ contains holes, the proof follows a similar pattern. The case of a matching between two vertices of the outer boundary, is identical to the one shown above. We assume for contradiction that two matching edges m_{pq}, m_{rs} cross when they are drawn as straight segments. Without loss of generality, we assume that m_{pq} connects two holes h_1, h_2. We extend the line pq on both sides to infinity. r and s are separated by this line (see Figure 4(b)). By Lemmas 4.1 and 4.2, it is clear that the trace of m_{rs}, constructed by the propagation, should intersect the line pq outside the segment pq, resulting in m_{rs} occluding, without loss of generality, the hole h_1 and the part of then outer boundary directly to its right, as ordered on the line pq, from each other. However, the propagation of the vertices of h_1 and those of the outer boundary in the shrinking process then intersect m_{rs}, contradicting the construction in Lemma 4.1. We conclude that there is no possible trace of matching between r and s, and so m_{rs} cannot exist. Therefore, even if $A_{i,j}$ has holes, no two matching edges drawn as straight segments intersect. Moreover, because M_{ij} is connected, all faces of the straight-segment representation of M_{ij} are simple, since they cannot contain disconnected holes.

Naturally, since active cells are disjoint, and all M_{ij} are contained in their respective active cell, the general two-dimensional matching $\mathrm{MG}_i = \bigcup_{1 \le j \le m} M_{ij}$ of the overlay $\mathrm{SD}(i)$ has no crossings (and no intersections as a geometric graph) as well. We will refer to this general graph of the overlay from now on. □

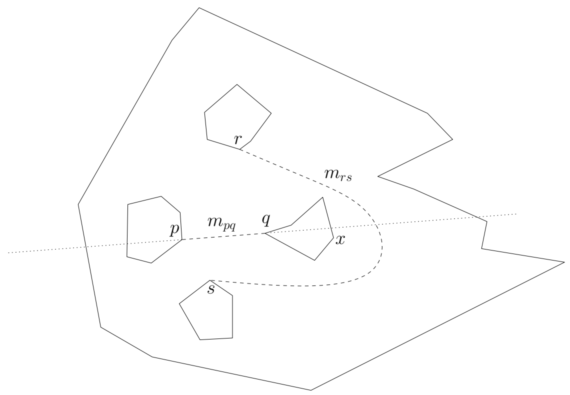
Theorem 4.2. *For any i, all the faces of MG_i are star-shaped.*

[4] If all vertices are convex, then the containment of the segment connecting p and q is assured by definition.

[5] We assume, without loss of generality, that N_{pq} is the first meeting point of these edges, since otherwise they would be neighboring edges in the same intermediate polygon until it vanishes or splits between them, and such matching would not exist. Thus, e_p,e_q are never within the triangle ΔpqN_{pq}.



(a) An example of lemma 4.2. The active-cell edges are in solid lines, the bisectors (and the line of sight between p and q) are in dashed-dotted lines, and the distances to the edges are marked with dashed lines. Here, b_r forms an event with b_q, because $d_{e_2} < d_{e_p}, d_{e_q}$, and so it prevents any event of p, q that would have created the matching.



(b) An example of a polygon with holes. Here, every path taken by the matching m_{rs}, as a trace of the movement of its vertices, is impossible. The exemplified path crosses the matching of the vertex x with any vertex of the outer boundary from its right, as the propagation of x and the outer boundary would cause.

Figure 4: Exemplifying the proof of theorem 4.1

Proof. In an inward propagation of the edges of such a face F, there are no split events, otherwise the hitting vertex would cause a split event in the original polygon as well, and F would not be a face of MG_i.[6] Hence, F shrinks to a single point, which is by definition in the halfplanes defined by all its edges (directed inwards). Therefore, this point, and a small neighborhood around it, are part of the kernel of F, which is thus non-empty. ☐

4.4 Lifting Up in Space

We show how to create the complete flow graph $\mathrm{FL}(V,E)$ in three dimensions, using all of the graphs MG_i:

1. The vertices of FL are created from the original vertices of the contours of all slices, and two new vertices are added per each edge intersection in the overlay, in the respective upper and lower slices. In addition, all original edges of the contours are added as well (parted by intersection nodes).
2. An edge of FL is created per each edge of all MG_i. Its connectivity is as follows:

   • If an endpoint of an edge is an original vertex in a slice L_i, which is not also an intersection vertex, it remains so in FL.

   • When an edge of MG_i is incident to an intersection vertex in the overlay, corresponding to two vertices (upper and lower), a decision is made as to which of these two it connects. If the edge of MG_i lies in a $U \setminus L$ cell (resp., a $L \setminus U$ cell) of the overlay, we connect it to the lower (resp., upper) FL vertex. Intersection vertices represent places where the surface changes orientation from the viewing direction (the Z axis) (see Figure 5). Therefore, this approach allows a gradual change of the surface around these vertices, and prevents foldovers. An intersection point can serve as a meeting point for, the lower slice with the contained upper slice, or vice versa, in which case this matching is cogent as well (see Figures 5(d) and 5(e)).

   • A new edge is created between two vertices of FL which correspond to the same intersection vertex in the overlay. Only these types of vertices exist in an empty active cell, making up a trivial match between overlapping segments of the symmetric difference.

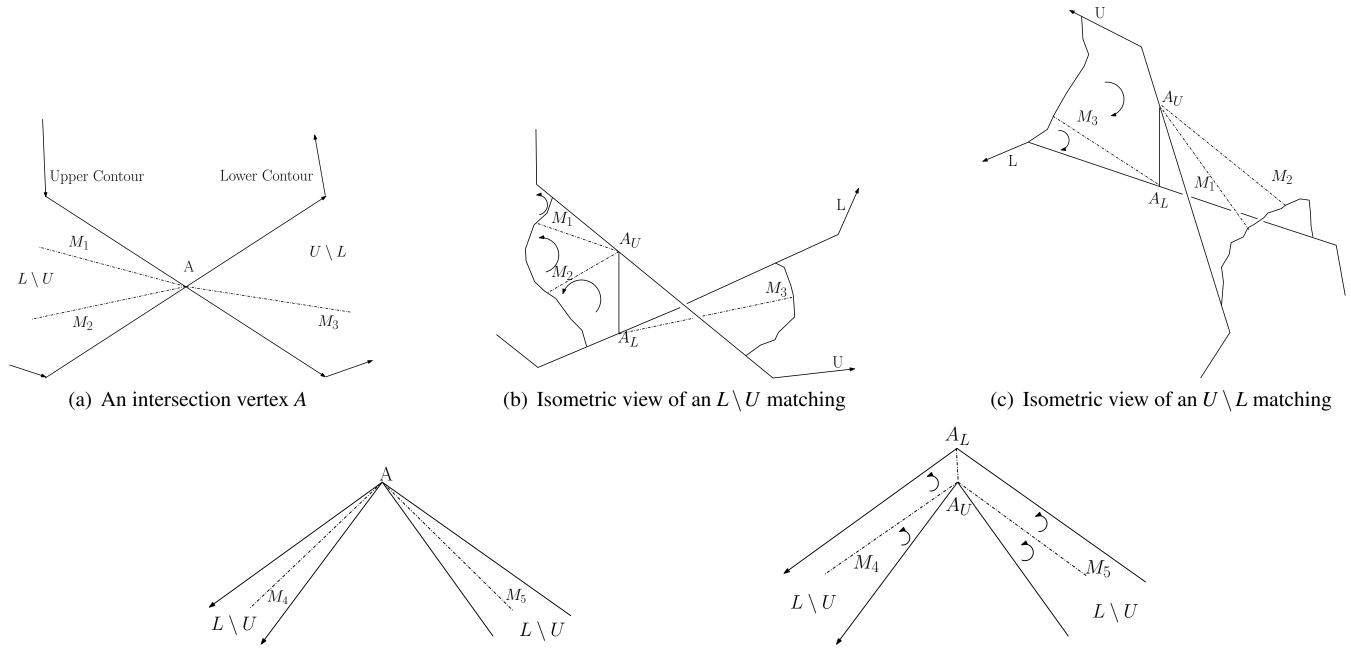
   • After all of the previous steps, edges that are parallel to other edges in FL (including original contour edges) are cast out.

[6]In most cases the faces of MG_i are convex. It is not always the case, since not all reflex vertices cause split events.

The resultant flow graph contains three types of edges: Original contour edges, edges between two adjacent slices (which we denote as *across-edges*), and edges from a slice to itself (denoted as *self-edges*). The self-edges have two subcategories: *Down-self-edges* (resp., *up-self-edges*) are self edges created from a matching in which the endpoints belong to the upper (resp., lower) slice in that overlay. This distinction is important when treating the edges as boundary curves, for then two self-edges between the same endpoints would diverse in this subcategory, and create two different curves. Self edges are common in a substantial area of a slice not shared by its adjacent slice, and so they mark a place where a feature of the surface vanishes in that direction. FL can be oriented to be a non-intersecting mesh, simply because of the non-intersection property of MG_i, $0 \le i \le n-1$, the matching graphs that comprise it.

5 Surface Orientation

We establish an orientation on the flow graph to form the faces of the surface, that will pose as the surface patches in the interpolation. The orientation is fairly simple, resulting from the layered construction of FL from every MG_i. We orient the sets of edges (and faces) between each two slices independently. The final stitching is trivial since the disjoint sets of faces join only along original contours. For such an orientation to take place, we treat each edge as two



(a) An intersection vertex A (b) Isometric view of an $L \setminus U$ matching (c) Isometric view of an $U \setminus L$ matching

(d) A meeting point (e) Isometric view of meeting point(s)

Figure 5: Matching decisions in intersection vertices. Figures 5(a–c) exemplify a pure intersection and the twist in the surface, and 5(d,e) exemplify intersection vertices which are meeting points. In 5(a–c), A is the intersection vertex in the overlay, and A_U, A_L are the corresponding vertices in the lower and upper slices, respectively. Since M_1, M_2 are in a $L \setminus U$ cell, they are connected to A_U, and M_3, in the $U \setminus L$ cell, is connected to A_L. Both M_4 and M_5 are connected to A_U since the surface does not twist at the meeting point A in 5(d,e).

twin directed half-edges, each of opposite direction and with the same endpoints. Then, we proceed as follows:

1. All faces that correspond to faces of MG_i are oriented CCW (resp., CW) if they reside in an $L \setminus U$ (resp., a $U \setminus L$) cell in the corresponding graph MG_i.
2. Vertical faces (orthogonal to the XY plane), which translate to segments in the corresponding graph MG_i, are oriented according to the affiliation of the segment in two dimensions ($U \setminus L$ or $L \setminus U$).

Note that when orienting an overlay, only one half-edge of each original contour edge is oriented, and the other half is oriented in the adjacent overlay (see Figure 6). Therefore, the orientation of the different layers results in a complete orientation of the flow graph. The oriented graph is now referred to as the *structural mesh*. This cannot, however, be treated as a regular mesh, since its faces are not necessarily planar, and furthermore, they can be overlapping (because of the inability to distinguish yet between two self-edges of different subcategories and with the same endpoints). A DCEL data structure [de Berg et al. 1997] is used in order to represent the orientation.

6 Extracting First-Order Data

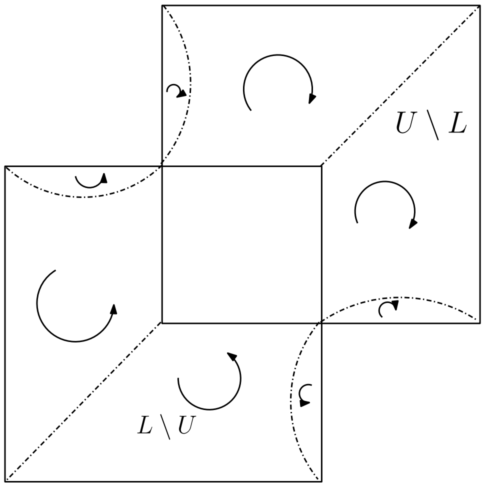
Having the structural mesh of the sought surface, we proceed by extracting first-order data (normal directions) from it, in order to create the appropriate curve network. The unique structure of the input and the flow graph allows the extraction of two linear-independent tangent directions: the flow of the surface between adjacent slices (the "vertical" direction), and the original contours describing the flow in the input slice (the "horizontal" direction). We establish first-order data at the vertices of the mesh in the following manner:

6.1 Horizontal Direction

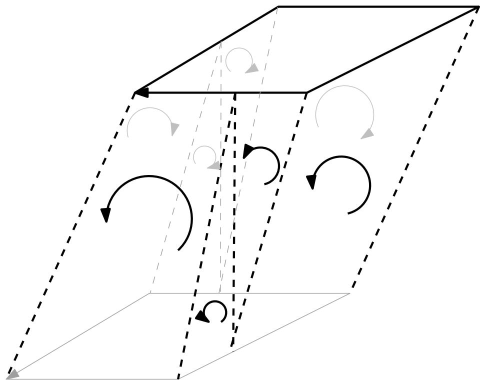
We interpolate all original contours using periodic cubic splines. For this matter, we consider the original vertices as fixed and the created intersection vertices as floating. The spline is created for the fixed points, and then we position the floating vertices on the spline, proportionally to their location on the original edge. Namely, if the original edge AB can be described as $At + B(1-t)$, $t \in [0,1]$, the cubic spline computed for the edge is $C_{AB}(t)$, and the intersection vertex lies in position t_0 on this edge, its final position would be $C_{AB}(t_0)$. This is not always a good practice, since cubic splines are not arclength parameterized, yet complex contours with many vertices produce shorter spline segments to which this positioning applies well. Then, we compute the horizontal direction $\vec{H}$ for each vertex (fixed or former floating) from the the derivation of $C_{AB}(t)$. Since cubic splines ensure (at least) G^1 continuity, both splines coinciding at each vertex agree with that quantity.

6.2 Vertical Direction

We find, by a least-square method, the vertical direction that, along with the horizontal direction, creates the tangent plane that fits best the directions acquired from across-edges and self-edges. For an



(a) Overlay with orientation in two dimensions



(b) Complete orientation in three dimensions

Figure 6: Orientation of an overlay of two squares. The faces that are visible are oriented CCW, and vice versa. In addition, the original contours are oriented with only one (opposite) half-edge each, and, so, their orientation matches that of their adjacent slices.

edge between vertices A and B, where A is the vertex in mentioning, the derived direction is $\vec{B}-\vec{A}$. If the edge is a self-edge on slice L_i, we add an element to the Z component which is $z^u_{\text{diff}} = z_{i+1} - z_i$ (resp., $z^d_{\text{diff}} = z_{i-1} - z_i$) for an up-self-edge (resp., down-self-edge). This addition differentiates between the two types of self-edges connecting identical vertices, and so it defeats the foldovers in the surface introduced by self-edges in their straight-line form. Self-edges mark places where the object vanishes, so that its boundary is not connected to the previous or next slice, and therefore, this step ensures the creation of an arc that allows its neighboring patch to vanish smoothly. Given the tangent plane $\text{TP}(u,v) = \vec{H}u + \vec{V}v$ for the sought variable V, we compute the deviation error of every direction $\vec{D}$ from that plane as $e(\vec{D}) =< \vec{D}, \vec{N} >^2$, where $\vec{N}$ is the unit normal to the plane. We seek the minimum the quantity $\sum_i e(D_i)$,[7] thus finding the tangent plane from which the matching edges deviate the least.[8]

[7]This known problem is reduced to finding the smallest eigenvalue, and the associated eigenvector of a $3x3$ matrix.

[8]Naturally, when there is only one across- or self-edge, the solution is unique, as it happens to be in many cases.

7 Creating the Curve Network

Having tangent planes set at each vertex, the underlying curve network for the surface is created. The network is comprised of cubic Bézier curves, which interpolate the vertices and conform to the tangent plane constraints. The original contours have already been set with the cubic splines, and for the other curves we use the projection method detailed in [Shirman and Séquin 1987] to establish the tangent vector of a curve $C(t)$ from endpoint $\vec{A}$ (at which the tangent vector is sought) with normal $\hat{N_A}$ to $\vec{B}$ with normal $\hat{N_B}$: The chord $\vec{AB}$ (if $\vec{AB}$ is a self-edge, then with the addition of z_{diff}) is projected onto the appropriate tangent plane, and the tangent is scaled to the chord's length:

$$\vec{T_A} = \left\|\vec{AB}\right\| \cdot \frac{\vec{AB} - < \vec{AB}, \hat{N_A} > \hat{N_A}}{\left\|\vec{AB} - < \vec{AB}, \hat{N_A} > \hat{N_A}\right\|} \tag{1}$$

$\vec{T_B}$ is computed in the same manner. Two endpoints and two tangents suffice to establish a cubic bézier curve with $\{P_0, P_1, P_2, P_3\}$ as a control polygon, since: $P_0 = \vec{A}, P_1 = \vec{A} + \frac{1}{3}\vec{T_A}, P_2 = \vec{B} + \frac{1}{3}\vec{T_B}, P_3 = \vec{B}$.

8 Building Surface Patches

Having the structural mesh and the appropriate curve network, we interpolate them by building surface patches on the faces of the mesh, using the curves as their boundaries. In order for the patches to be G^1-continuous, along their common boundaries, we have to define their control points accordingly. We treat three types of faces: quadrilaterals, triangular, and k-sided.

8.1 Quadrilaterals and Triangular patches

In order to avoid the vertex consistency problem [Mann et al. 1992] we use the interpolation technique presented in [Chiyokura and Kimura 1983] for quadrilateral patches, to create patches with G^1 continuity between them. The triangular Gregory scheme we use is due to [Longhi 1985]. The essence of this method is to establish the tangent plane along every boundary curve $C(t)$, using the derivative of the curve and a vector field D, such that $u, v \in \Re, \text{TP}(t) = uC'(t) + vD(t)$. $D(t)$ is a linear blend of two vectors D_A, D_B at the endpoints A, B of the curve, which are uniquely established by the tangent planes at these points.[9] This method sets all the internal points of an either a quadrilateral (cubic) Gregory patch (8 internal points) or a triangular (quadratic) patch (6 internal points).

8.2 k-Sided Patches

Some methods, described in the literature to deal with k-sided patches, suggest either to cover the surface with a single patch [Loop and DeRose 1989], or offer subdivision schemes [Chiyokura and Kimura 1983; Levin 1999; Murotani and Sugihara 2002; Overveld and Wyvill 1997]. Since the faces of the

[9]Actually, D_A, D_B can be chosen to be any vectors on the tangent planes of their respective endpoints, in a matter that will produces these vectors in the opposite direction when interpolating on both sides of the boundary curve. We chose them to be $D_A = \hat{N_A}xC'(A), D_B = \hat{N_B}xC'(B)$.

underlying mesh have a simple star-shaped embedding (see Section 4), we employ a simple subdivision scheme: we subdivide the patch into convex faces, and then recursively subdivide subpatches by forming a new boundary curve between two vertices of the patch, until all subpatches are either triangular or quadrilateral. The two endpoints chosen in every subdivision are the two non-adjacent closest vertices of that subpatch. In case the two endpoints belong to the same slice, we add the z_{diff} component as explained in Section 6. This subdivision method does not create extra vertices, and so it is appealing. Then, having created quadrilateral and triangular patches, we proceed with the regular interpolation.

9 Experimental Results

We present results obtained by applying our algorithm to several MRI or CT scanned instances: the steps of the reconstruction of a heart in Figure 8, and other medical input examples in Figures 9 to 11. The algorithm was also applied to a topographic map (of the Zikhron-Yaakov area in Israel), shown in Figure 12.

9.1 Complexity of the Algorithm

Calculating the symmetric difference of two slices is done by a simple line-sweep algorithm which time complexity is $O(n \log n + k)$, where k is the number of intersection points and n is the number of original contour vertices. The value of k can be up to $O(n^2)$ in the worst case, but for most practical cases it is $O(n)$. The bottleneck of the algorithm is the straight-skeleton computation. Although theoretically it can be done in subquadratic time [Eppstein and Erickson 1998], we utilized the algorithm of [Felkel and Obdržálek 1998], whose running time is $O(N(r + \log N))$ ($N = \theta(n+k)$ being the size of the input to the algorithm and r being the number of reflex (concave) vertices, which is $O(N)$). The matching algorithm requires $O(1)$ steps per edge or split event, and, thus, does not add to the total complexity. Subdividing and calculating the network and control points of patches require additional $O(N)$ time. Therefore, the total worst-case time complexity is $O(N^2) = O(n^4)$, yet most cases produce subquadratic behavior. The space complexity of the algorithm is $O(N)$, since there are $O(N)$ matching edges (every skeletal edge creates $O(1)$ matches and the total number of faces is $O(N)$ as well).

9.2 Timing Measurements

Table 1 shows timing results for the different stages of the algorithm working on different inputs. The algorithm was implemented in Visual C++.NET and run on a 3GHz Athlon 64 processor PC with 1Gb of RAM.

Input					
Input	**Heart**	**Hip**	**Pelvis**	**Lungs**	**Map**
Size of Input					
Slices	30	34	50	34	17
Contours	65	38	108	88	111
Vertices	1285	1706	2277	3121	4546
Running Times (Seconds)					
Symmetric difference	0.079	0.062	0.190	0.202	0.237
Straight-skeleton & matching	0.874	1.118	1.81	4.235	17.857
Orientation	0.031	0.032	0.047	0.062	0.065
Cure network	0.156	0.218	0.312	0.453	0.813
Patches interpolation	0.094	0.108	0.126	0.222	0.230
Total time	1.234	1.538	2.485	5.174	19.202
Size of Output					
Curves	4630	5386	8805	11479	14533
Patches	2982	3485	5662	7362	8971

Table 1: Time measurements for several inputs. Excluded are irrelevant visualization methods (such as patch tessellation). Curve network time includes first-order data extraction. The overall measured time complexity is about $O(n^{2.12})$.

10 Conclusions and Future Work

We have shown a method to reconstruct an explicit free-form smooth surface directly from a set of parallel contours, which is robust and creates a visually pleasing result, and is without any need for intermediate contour interpolation. In addition, since the interpolation is independent of the matching, different matching heuristics could benefit from employing this algorithm as well. However, the given G^1-continuity does not always satisfy needs of smoothness (especially with geometric dissimilarities which create high curvature areas, such as in Figure 7). By using higher-degree curves and surfaces, one can reconstruct a surface with G^2-continuity for a better result, albeit further complications. The structure of the flow graph allows further extraction of geometric quantities (e.g., curvature) in a similar manner, avoiding some of the complications introduced in general mesh surface fitting. Also, although this algorithm handles dissimilarity well, it is at its best when the active cells of the overlay are small in area with respect to the original contours, and so it might benefit greatly from preprocessing the input contours with proper transformations.

11 Acknowledgment

Work on this paper by both authors has been supported in part by the European FP6 Network of Excellence Grant 506766 (AIM@SHAPE). We wish to thank Petr Felkel for providing us with the full source code of his implementation of the straight-skeleton computation algorithm.

References

AICHHOLZER, O., AURENHAMMER, F., ALBERTS, D., AND GÄRTNER, B. 1995. A novel type of skeleton for polygons. *J. of Universal Computer Science 1*, 12, 752–761.

AKKOUCHE, S., AND GALIN, E. 2004. Implicit surface reconstruction from contours. *The Visual Computer 20*, 6 (August), 392–401.

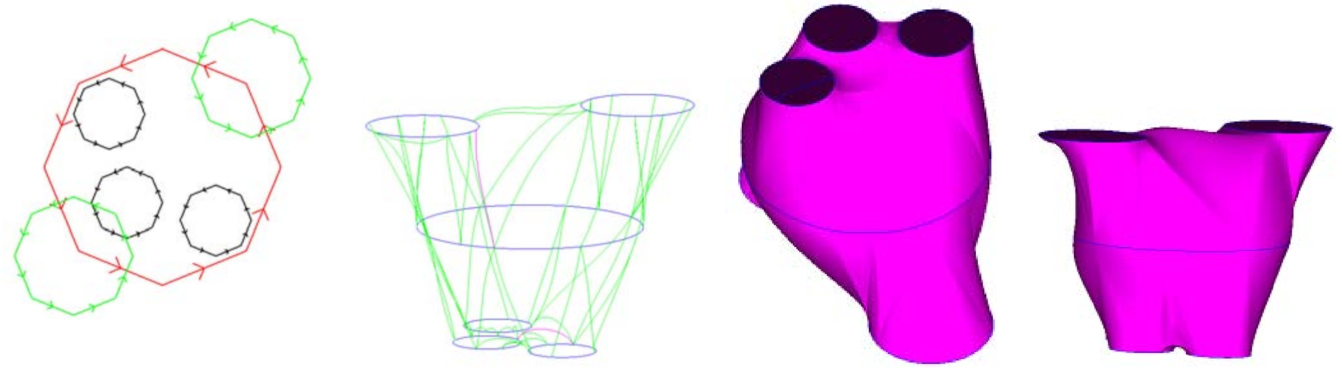
BAJAJ, L., COYLE, E. J., AND LIN, K. N. 1996. Arbitrary topology shape reconstruction from planar cross sections. *Graphical Model and Image Processing 58*, 6, 524–543.

BAREQUET, G., AND SHARIR, M. 1996. Piecewise-linear interpolation between polygonal slices. *Computer Vision and Image Understanding 63*, 2, 251–272.

BAREQUET, G., SHAPIRO, D., AND TAL, A. 2000. Multilevel sensitive reconstruction of polyhedral surfaces from parallel slices. *The Visual Computer 16*, 2 (March), 116–133.

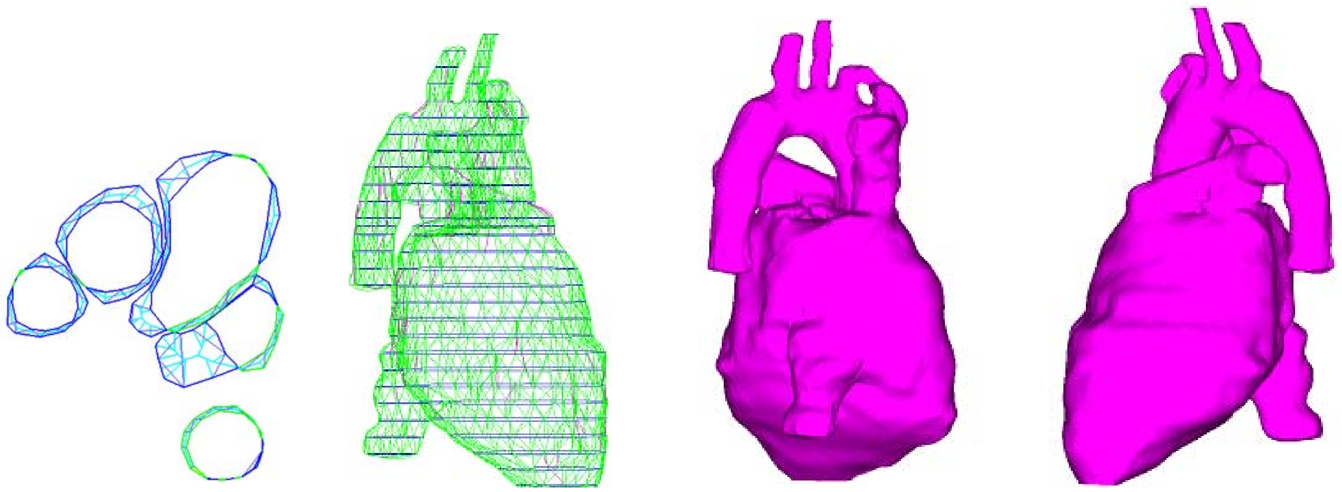
BAREQUET, G., GOODRICH, M. T., LEVI-STEINER, A., AND STEINER, D. 2004. Contour interpolation by straight skeletons. *Graphical Models 66*, 4, 245–260.

BOISSONNAT, J.-D. 1988. Shape reconstruction from planar cross sections. *Computer Vision Graphing, and Image Processing 44*, 1, 1–29.



(a) The three level-set slices (b) Curve network (c) Full view (d) Full view from a different angle

Figure 7: A reconstruction of a simple example, featuring a smooth solution to the branching and correspondence issues. The output surface exemplifies clearly a reconstruction of a "many-to-many" case.



(a) Overlay (b) Curve network (c) Full view (d) View from a different angle

Figure 8: A reconstructed heart. 8(a) Shows an overlay of two adjacent slices from the input, 8(b) shows the curve network, showing splines of original contours (dark) and matching curves (bright). 8(c) and 8(d) show the final results with shading, drawing the original contours as level sets on the surface.

CHAI, J., MIYOSHI, T., AND NAKAMAE, E. 1998. Contour interpolation and surface reconstruction of smooth terrain models. *Proc. IEEE VIS,* 27–33 (October).

CHIYOKURA, H., AND KIMURA, F. 1983. Design of solids with free-form surfaces. *Proc. SIGGRAPH, Computer Graphics 17*, 3, 289–298.

CONG, G., AND PARVIN, B. 2001. Robust and efficient surface reconstruction from contours. *The Visual Computer 17*, 4 (June), 199–208.

DE BERG, M., VAN KREVELD, M., OVERMARS, M., AND SCHWARZKOPF, O. 1997. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Germany.

EPPSTEIN, D., AND ERICKSON, J. 1998. Raising roofs, crashing cycles, and playing pool: applications of a data structure for finding pairwise interactions. *Proc. 14th Ann. Symp. on Computational Geometry,* 58–67.

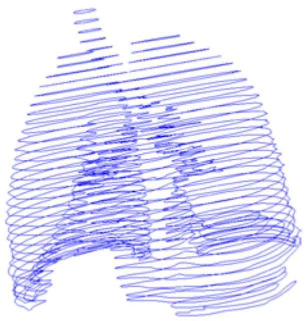
FELKEL, P., AND OBDRŽÁLEK, Š. 1998. Straight skeleton implementation. *Proc. Spring Conf. on Computer Graphics, L. S. Kalos, ed.* 210–218.

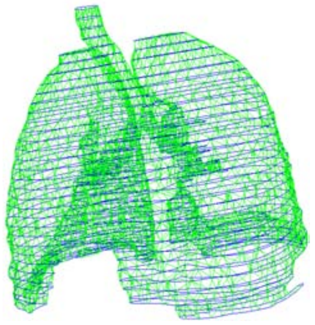
GREGORY, J. A. 1974. Smooth interpolation without twist constraints. *Computer Aided Geometric Design, R. E. Barnhill and R. F. Riesenfeld, eds.,* 71–88.

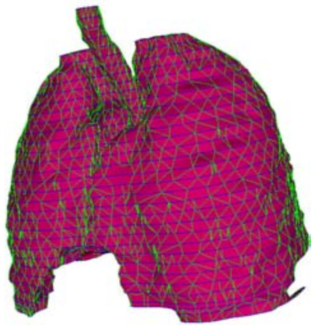
HORMANN, K., SPINELLO, S., AND SCHRÖDER, P. 2003. C^1-continuous terrain reconstruction from sparse contours. *Proc. Vision, Modeling, and Visualization, T. Ertl, B. Girod, G. Greiner, H. Niemann, H.-P. Seidel, E. Steinbach, and R. Westermann, eds.,* 289–297 (November).

JU T., WARREN J., CARSON J., EICHELE G., THALLER C., CHIU W., BELLO M., AND KAKADIARIS I. 2005. Building 3D surface networks from 2D curve networks with application to anatomical modeling. *The Visual Computer, Special Issue for Pacific Graphics, 21(8–10):764–773.*

LEVIN, A. 1999. Interpolating nets of curves by smooth subdivision surfaces. *Proc. 26th Ann. Conf. on Computer Graphics and*


(a) Slices


(b) Curve network


(c) Full view

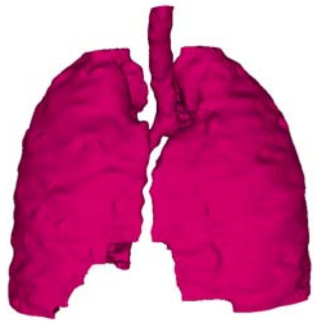
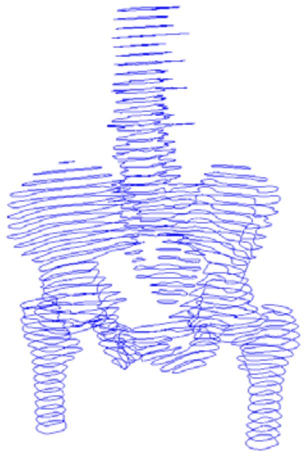
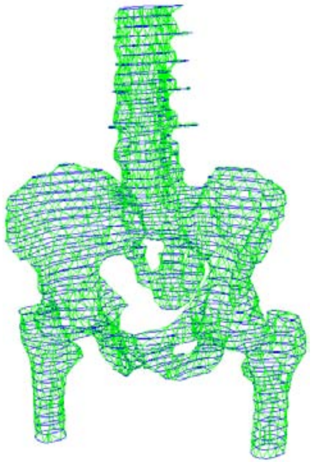
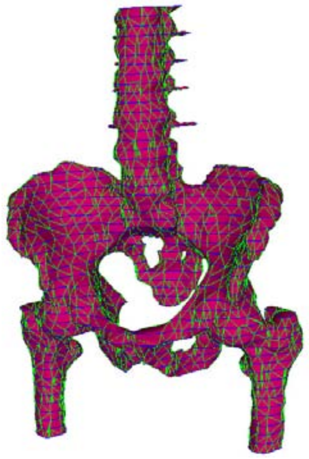

(d) Shaded view

Figure 9: A pair of lungs


(a) Slices


(b) Curve network


(c) Full view

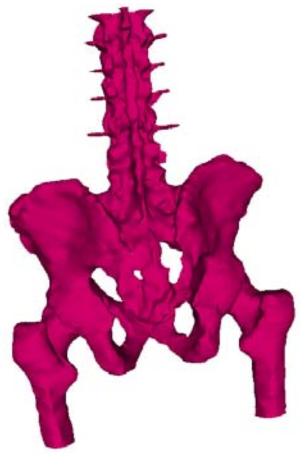

(d) Shaded view

Figure 10: A Pelvis

Interactive Techniques, 57–64.

LONGHI, L. 1985. Interpolating patches between cubic boundaries. Technical report, Univ. of California at Berkeley, Berkeley, CA.

LOOP, C. T., AND DEROSE, T. D. 1989. A multisided generalization of Bézier surfaces. *ACM Transactions on Graphics 8*, 3 (July), 204–234.

MANN, S., LOOP, C., LOUNSBERY, M., MEYERS, D., PAINTER, J., DEROSE, T., AND SLOAN, K. 1992. A survey of parametric scattered data fitting using triangular interpolants. *Curve and Surface Design, H. Hagen, ed., SIAM,* 145–172.

MUROTANI, K., AND SUGIHARA, K. 2002. G^1 surface interpolation for irregularly located data. *Proc. Geometric Modeling and Processing—Theory and Applications,* 187.

NILSSON, O., BREEN, D., AND MUSETH, K. 2005. Surface reconstruction via contour metamorphosis: An eulerian approach with lagrangian particle tracking. *Proc. IEEE VIS,* 407–414 (October).

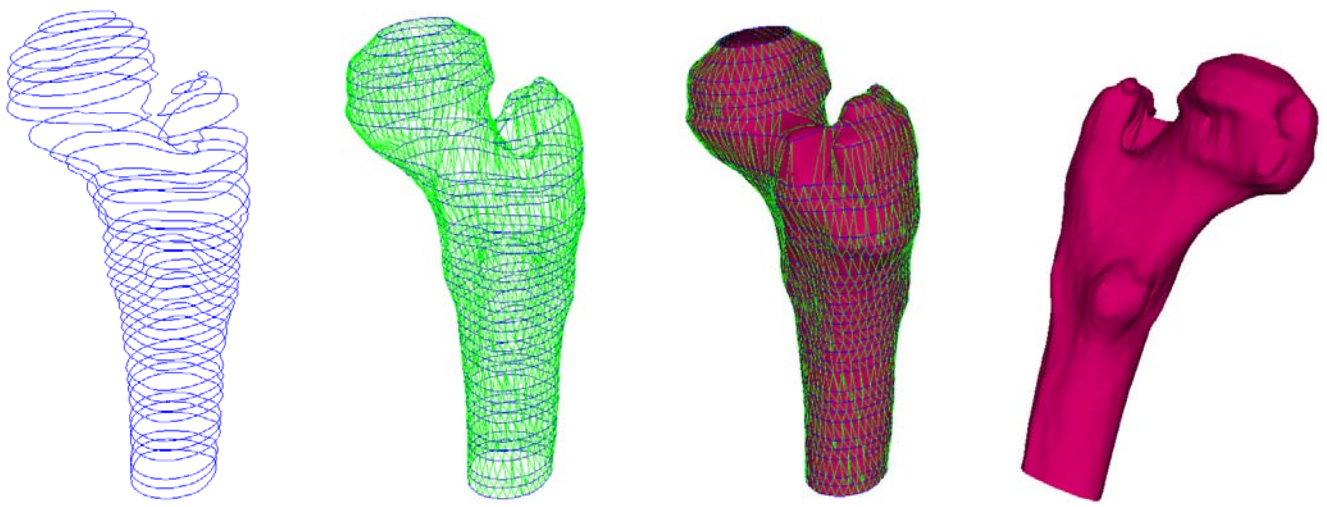
OLIVA, J.-M., PERRIN, M., AND COQUILLART, S. 1996. 3D reconstruction of complex polyhedral shapes from contours using a simplified generalized voronoi diagram. *Computer Graphics Forum 15*, 3, 397–408.

OVERVELD, C. W. A. M. V., AND WYVILL, B. 1997. An algorithm for polygon subdivision based on vertex normals. *Proc. Computer Graphics International,* 3–12.

SHIRMAN, L. A., AND SÉQUIN, C. H. 1987. Local surface interpolation with Bézier patches. *Computer Aided Geometric Design 4*, 4 (December), 279–295.

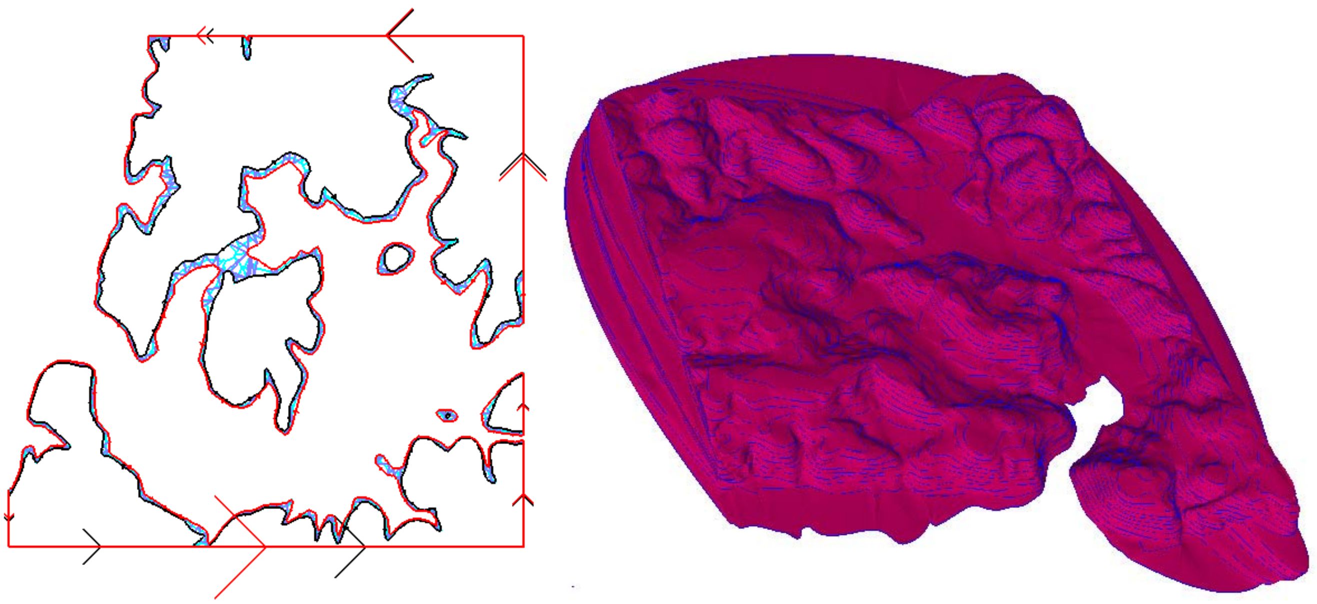
WANG, D., HASSAN, O., K.MORGAN, AND WEATHERILL, N. 2006. Efficient surface reconstruction from contours based on two dimensional delaunay triangulation. *Int. J. for Numerical Methods in Engineering 65*, 5 (January), 734–751.

YAKERSBERG, E. 2004. *Morphing between geometric shapes using straight-skeleton-based interpolation*. M.Sc. Thesis, Dept. of Computer Science, Technion, Haifa, Israel.



(a) Slices (b) Curve network (c) Full view (d) Shaded view

Figure 11: A hip bone



(a) Overlay of two slices (b) Full View

Figure 12: An example of a reconstructed topographic map. This input is unique in the sense that there are no intersections in any overlay. The edges of the map are also interpolated smoothly, and ,therefore, there is no straight “wall” alongside the map.