# Topologically correct reconstruction of tortuous contour forests

John Edwards
Department of Computer Science
University of Texas at Austin
edwardsj@cs.utexas.edu

Chandrajit Bajaj
Department of Computer Science
University of Texas at Austin
bajaj@cs.utexas.edu

## ABSTRACT

Motivated by the need for correct and robust 3D models of neuronal processes, we present a method for reconstruction of spatially realistic and topologically correct models from planar cross sections of multiple objects. Previous work in 3D reconstruction from serial contours has focused on reconstructing one object at a time, potentially producing inter-object intersections between slices. We have developed a robust algorithm that removes these intersections using a geometric approach. Our method not only removes intersections but can guarantee a given minimum separation of objects. This paper describes the algorithm for geometric adjustment, proves correctness, and presents several results of our high-fidelity modeling.

## Categories and Subject Descriptors

I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling; I.3.7 [**Computer Graphics**]: Three Dimensional Graphics and Realism

## 1. INTRODUCTION

Much work on 3D reconstruction from planar cross-sectional data has been done in recent years to the benefit of many applications. These methods generate a 3D model of an object from given contours, and many of them are extremely efficient and accurate. These single-component reconstruction methods are not sufficient, however, when faced with reconstructing models involving multiple components. This type of reconstruction is necessary in many different fields, including neuronal modeling, surgical planning, and composite materials simulation. The problem is that reconstructing components one by one can yield intersections between components after compositing them into the same model, regardless of the guarantees made by the algorithm. The intersections occur frequently in data that is highly tortuous and densely packed, and is exacerbated further by highly anisotropic data, where the spacing of slices is large compared to the geometric behavior of the objects. The methods presented in this paper aim to fill the gap in generating topologically correct multi-component models while maintaining the accuracy of existing single-component methods.

Our ultimate goal is to better understand the brain through accurate modeling and simulation of neuronal processes (e.g. axons, dendrites, glial cells). Previously this modeling has been done using the simplified cases of treating the dendritic arbor as a series of cylinders. However, serial section transmission electron microscopy (ssTEM) reveals highly complex geometries among neuronal processes, including high tortuosity, varying caliber, spiny protrusions and extremely tight packing. Our work of simulating neuronal electrophysiological function at high resolution requires very accurate and topologically correct 3D reconstructions of neuronal processes. These reconstructions can be used in a variety of modeling experiments using high-fidelity versions of the traditional cable model approach [11] or, more recently, an approach based on meshless Weighted Extended B-spline-based finite element methods [10].

The source data for generating these neuronal reconstructions is generally ssTEM imagery. Neuronal contours are generated from these images by neurobiology experts who trace out the contours using a variety of tools [24]. Pixel spacing in the xy plane of neuronal ssTEM images is roughly 2-5 nm, while spacing between slices is closer to 45 nm. Extracellular spacing (spacing between neuronal processes) is on the order tens of nanometers [25, 19]. This close spacing, combined with the comparatively large distance between slices can cause inter-object intersections between slices when using single-component reconstruction techniques. This topological incorrectness prohibits any type of multi-component analysis based on finite element methods.

We have developed an intersection removal method that acts in concert with any surface reconstruction method, provided it conforms to several criteria. Our approach is to remove intersections by moving triangle vertices and induced points along the axis orthogonal to the slice plane. This approach can remove intersections efficiently and without causing additional intersections. Using an approach that moves vertices in the slice plane is possibly more intuitive, but yields considerably greater computational complexity [3].

We first give a brief outline of work that has been done both in single component and multi-component surface reconstruction from cross-sectional contour data (Section 2). We then build up a set of rules and theorems to prove correctness and robustness of our algorithm (Section 3), followed by a discussion of the surface removal algorithm in-


*2010 ACM Symposium of Solid and Physical Modeling (SPM '10), Haifa, Israel*

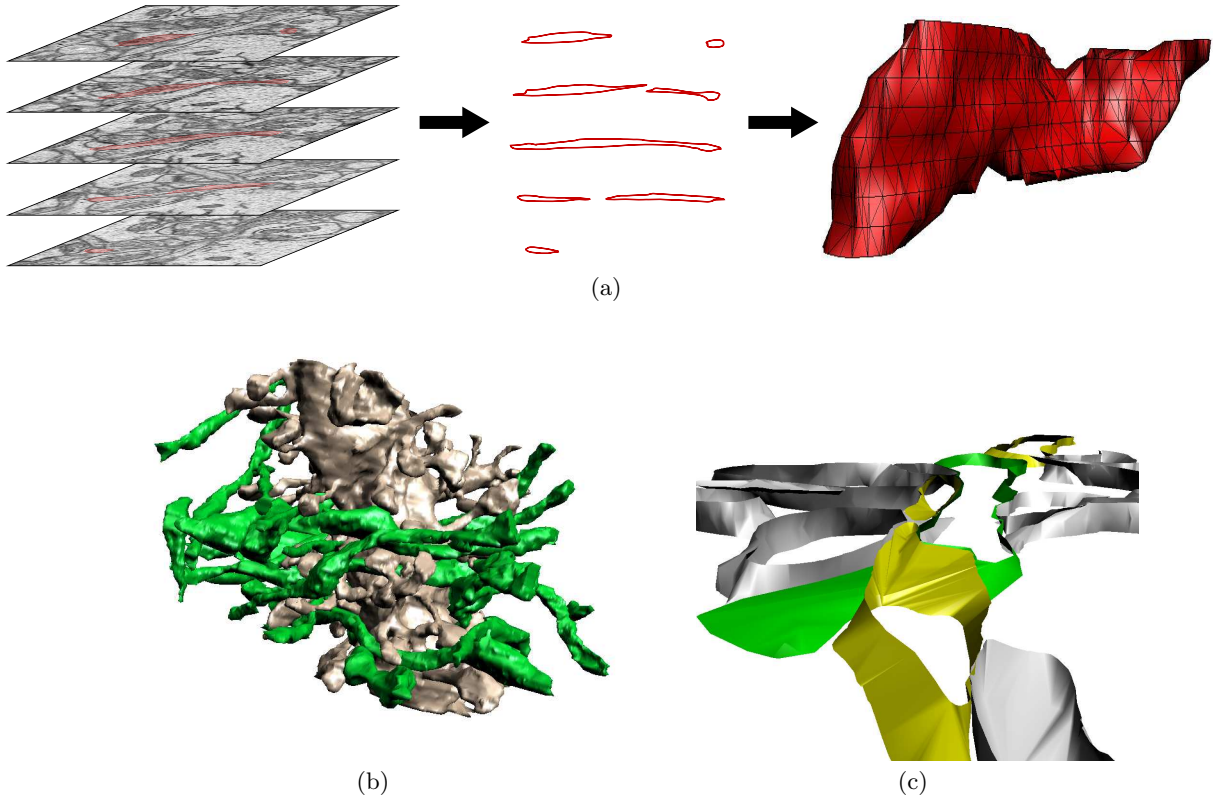


**Figure 1: Overview of single component reconstruction. (a) Single component reconstruction pipeline. Contours are first traced onto images, then a component is reconstructed from the contours. (b) A multi-component model generated by multiple calls to a single component reconstruction algorithm. (c) A subset of the reconstruction in (b) that shows the surfaces of multiple components between two slices after intersection removal. See also figure 8(a).**

cluding complexity bounds (Section 4). We then present results of our implementation (Section 5).

## 2. RELATED WORK

A large amount of work has been done to solve the single-component reconstruction problem. Fuchs et al. [15] presented the problem and proposed a solution based on triangulations guided by a toroidal graph. Barequet and Sharir [6] introduced a method using linear interpolations between slices of medical images. Bajaj et al. [2] expanded on their work by using medial axes to tile regions with no legal slice-to-slice tiling. Oliva et al. [22] specifically targeted difficult objects (objects with multiple branches, holes, and other irregularities) and used Voronoi diagrams to construct topologically correct surfaces.

Somewhat more recent works interpolate contours using 2D skeletons in valid tile regions [5], Delaunay triangulation [26] and contour morphing [21]. Most works use some type of linear interpolation and thus require a smoothing post-processing step. One approach that performs non-linear smoothing during tiling is found in [7]. Other recent approaches reconstruct surfaces from non-parallel contours [20, 9, 8].

A different class of approaches that work directly from image data rather than contours exist, such as energy-minimizing 3D snakes [18] and a 3D extension to path cost-minimizing LiveWire [14]. These approaches require a level of user interactivity, however, and often don't scale well to massive datasets with large numbers of components.

Surprisingly little work has dealt with the issue of intersections between multiple components. Bajaj and Gillette [3] produced a method for removing intersections by removing contour overlaps in intermediate planes. This algorithm uses an inflated medial axis surface to separate mid-slice contours, but there is no guaranteed bound on the number of mid-slice contours to be separated. In addition, branching of components (where a single contour in one slice correlates to multiple contours in an adjacent slice) is treated in a pre-processing step, where branch points are moved as needed to enable intersection removal. Our algorithm has provably finite computational bounds and it also handles branching of components without treating them as a special case.

Of course, there are many algorithms that perform general boolean set operations on surfaces [17]. Our algorithm, however, is able to perform intersection removal inline and thus require far less computation than the comparatively heavy-weight methods. It is restricted to intersections between slices and can make correctness guarantees based on knowledge of the source contours. Intersection discovery and most other elements of the algorithm are performed primarily using inexpensive 2D geometry calculations. So by using this method over more general methods we get large computational savings as well as surfaces conforming to constraints

imposed in the initial single component reconstruction.

## 3. RULES

Our method is correct and robust and, in addition, it has bounded computational complexity and requires neither parameter optimization nor refinements. The only modifications made to the original surface are the addition of induced points and moving of points parallel to the axis orthogonal to the image plane, which we call the z axis. We prove that moving these points in z in a constrained manner can remove intersections without causing additional intersections among other components. In addition, our method can guarantee a minimum separation distance of $\delta$. This is an important guarantee for applications that are sensitive to inter-component spacings.

We state here criteria on which our method relies, as well as various theorems which prove correctness and completeness. We define the xy plane to be the plane of the original images (and slice contours), and the z axis to be perpendicular to the xy plane. For simplicity, we assume that each component has a unique color and each element belonging to that component (*e.g.* contour, boundary) inherits the same color. So if two contours have different colors, they belong to different components.

CRITERION 1. *The reconstructed surface is a piecewise closed surface of polyhedra.*

CRITERION 2. *Any vertical (parallel to the z axis) line segment between two adjacent slices intersects a single component exactly 0 or 1 times, or along exactly one line segment.*

CRITERION 3. *Slicing the reconstructed surface on any of the original slices produces exactly the input contours.*

CRITERION 4. *All contours on the same slice have a minimum separation distance of $\delta$.*

CRITERION 5. *A contour cannot be nested inside a different colored contour.*

Criteria 1-3 are borrowed from [2] and are required to ensure high quality and topologically correct surfaces. Criterion 1 ensures that each component surface given to our algorithm are topologically correct and water-tight. Note that this does not guarantee that intersections between components won't exist. Criterion 2 also applies only to single components and helps to avoid topologies that are unlikely. Without it, a host of additional correspondences are possible, most of which are incorrect. In order to avoid additional complexity and a large number of false positive correspondences, this criterion is carried through into our work. It assumes, however, that the slice spacing is close enough to enable reasonable reconstructions even with the criterion in place. Criterion 3 ensures that interpolation is bounded by only that required to generate a likely topology and avoids adding information to the original contours.

We enforce criterion 4 by adding a pre-processing step to our algorithm – that of separating contours by $\delta$. Of course, if the application prohibits this then either $\delta$ can be decreased or application-specific contour separation methods can be used.

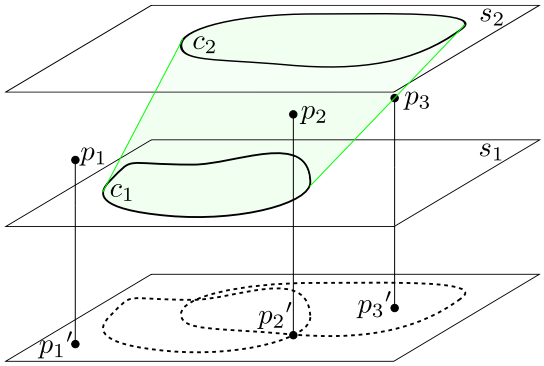



**Figure 2: $s_1$ and $s_2$ are two adjacent slices and the lower plane is an arbitrary xy plane showing containing contours of various points. $p_2$ and $p_3$ are on the green component, while $p_1$ cannot be according to lemma 1. $\mathscr{C}(p_1) = \emptyset$, $\mathscr{C}(p_2) = \{c_1, c_2\}$, $\mathscr{C}(p_3) = \{c_2\}$.**

Criterion 5 is currently required as nested contours of different colors are not currently supported. In other words, components are treated as solids without any nested components. Note that this does not preclude nested contours of the same color, which can occur when there are concavities in the surface.

Our algorithm works on slice pairs. That is, all components on a pair of slices are tiled, and then intersections occuring between slices are removed. The algorithm then proceeds to the next pair of slices. In the following discussion, it is assumed that the data we are dealing with lies on and between a pair of adjacent slices. We also deal with only two components at a time, $C^g$ and $C^y$, the green and yellow components, respectively. We will see that working with only two components at a time is justified.

Proofs of the following lemmas and theorems can be found in the appendix.

DEFINITION 1. *Suppose $p'$ is the projection of point $p$ onto the xy plane. We say that contour $c$ is a* containing contour *of $p$ if $p'$ is inside or on the boundary of $c'$ (see figure 2). We define $\mathscr{C}(p)$ to be the set of all containing contours of point $p$ and $\mathscr{C}^g(p)$ to be $\mathscr{C}(p)$ restricted to all green contours $\{c_i^g\}$, i.e., $\mathscr{C}^g(p) = \mathscr{C}(p) \cap \{c_i^g\}$.*

LEMMA 1. *Suppose $p^g$ is a point on the surface $\partial C^g$ of component $C^g$. Then $1 \leq |\mathscr{C}(p^g)| \leq 2$.*

This lemma states that any point on a surface must have at least one containing contour (of any color) but no more than two. In figure 2, $p_1$ has no containing contour and so cannot exist on the surface of the reconstructed component.

LEMMA 2. *If $|\mathscr{C}(p^g)| = 2$ then the two contours in $\mathscr{C}(p^g)$ lie on separate slices.*

DEFINITION 2. *$S^g$ is the set of all points $p^g \in \partial C^g$ such that $|\mathscr{C}^g(p^g)| = 2$. These points lie "sandwiched" between two green contours. Similarly, $U^g$ is the set of all points $p^g \in \partial C^g$ such that $|\mathscr{C}^g(p^g)| = 1$. These points have no containing contour on one of the slices. For a point $p^g \in U^g$, we call its single containing green contour the* penumbral contour *$\mathscr{P}(p^g)$.*

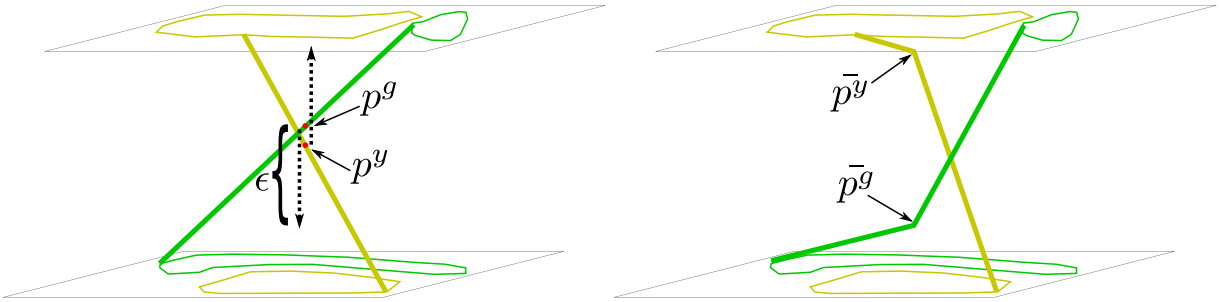



**Figure 3: Conflict points are detected and removed by moving the points along the z axis. $p^g$ and $p^y$ are corresponding conflict points. The conflict is removed by moving $p^g$ and $p^y$ in the z direction by $s^g\epsilon$ and $s^y\epsilon$, respectively (equation 2) to produce new points $\bar{p^g}$ and $\bar{p^y}$.**

In figure 2, $\mathscr{P}(p_3) = c_2$ and the penumbral contour for the other two points is undefined.

As it happens, points in $S^g$ cannot lie on the inside of both contours, but must lie on the boundary of at least one, but this distinction is not necessary for our analysis here.

DEFINITION 3. *Suppose $p^g \in U^g$. Then $\mathscr{Z}(p^g)$ is the z coordinate value for $\mathscr{P}(p^g)$. We call this the* z-home *value for $p^g$. $\mathscr{Z}(q^g)$ is undefined for all $q^g \notin U^g$.*

DEFINITION 4. *Consider a sphere of radius $\delta$ centered at a point $p$. Consider also the cylinder of radius $\delta$ about the vertical line segment from $p$ to $p'$ where $p'$ is the projection of $p$ onto the slice at $\mathscr{Z}(p)$. The union of the open sphere and open cylinder is called the buffer region $\mathscr{B}(p)$ of $p$.*

DEFINITION 5. *Point $p^g \in U^g$ is called a* conflict point *if there is some point $p^y \in U^y$ such that $p^y \in \mathscr{B}(p^g)$.*

Conflict points are points at which two components are closer than $\delta$ or at which some point $p^y$ is inside or on the surface of two components.

LEMMA 3. *No point $p^g \in S^g$ can be a conflict point.*

LEMMA 4. *Let $p^g \in \partial C^g$ be a conflict point. Then there is no other point $q^g \in \partial C^g$ such that $p^{g\prime} = q^{g\prime}$.*

This lemma shows that a point $p$ can conflict with points of only one other component. This is important because it frees us to deal with only component pairs. That is, we are guaranteed that if component A and component B intersect, there is no other component C that intersects in the same region. This naturally leads to the algorithm described in section 4, in which conflict points are found and dealt with between pairs of components.

THEOREM 1. *Two components $C^g$ and $C^y$ are within $\delta$ distance of each other if and only if there is at least one conflict point on the surface of either component.*

COROLLARY 1. *If two components $C^g$ and $C^y$ intersect then there is at least one conflict point on the surface of either component.*

By this theorem and corollary we see that removing all conflict points between two components is necessary and sufficient to guarantee that the components are not intersecting and are separated at every point by at least $\delta$.

THEOREM 2. *Moving any point $p^g \in U^g$ in the direction of $\mathscr{Z}(p^g)$ will not generate any additional conflict points among any pair of components.*

This theorem is important to justify removing conflict points between pairs of components. If it wasn't so then the algorithm would be computationally far more complex as we would have to continuously check previously resolved components for additional conflict points after any modification is made in the region.

THEOREM 3. *Conflict points exist on the triangulated surfaces of two components if and only if at least one conflict point exists either at a triangle vertex or along a triangle edge of either component.*

## 4. IMPLEMENTATION

Our algorithm removes conflict points in a manner conforming to theorem 2, i.e., no additional conflict points are introduced at any step. And since removing all conflict points at locations defined in theorem 3 effectively removes all conflict points, our algorithm is bounded by the number of those locations.

The algorithm removes conflict points between pairs of components. We rely on theorem 2 to justify working with only two components at a time: resolving conflict points between two components will not increase the number of conflict points among any other pair of components.

In addition to working with only two components at a time, we also deal only with reconstructions between two given slices at a time. There are no component intersections on the slices themselves (by criteria 3 and 4), and the algorithm does not modify any points on the slices. So the contours on the slices act as natural boundary points between intra-slice reconstructions, even after intersection removal.

The algorithm is as follows:

1. Separate contours in slices $z_i$ and $z_{i+1}$
2. Run single component tiling on each component in slices
3. Determine conflict points
4. Trace cut from conflict point to its exit in a triangle
5. Triangulate (planar) polygons
6. Adjust z values

We now describe each step in detail.

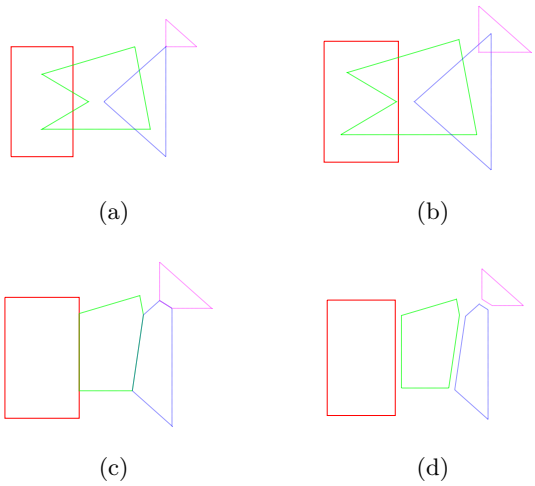



**Figure 4: Contour intersection removal. (a) shows the original contours and (b) shows the contours after dilation by $\delta/2$. In (c) the dilated contours have been clipped and (d) shows the final result after erosion.**

## Slice contour separation

Our 2D contour intersection removal algorithm is a simple and efficient algorithm aimed at separating contours by a value $\delta$ (see figure 4). All contours in a given slice are first dilated by $\delta/2$ after which proper intersections between contours (proper, in this case, meaning intersections such that a segment from one contour touches both the interior and exterior of another contour) are found using a sweep line and marked. We are guaranteed that there are an even number of intersections as only proper intersections are marked. Intersection points are paired up such that the mid-point of the line segment defined between two intersections is in the interior of two contours. For each of these pairs, the intersecting contours are clipped along this line segment. The contours are then eroded back to roughly their original shape minus the clipped areas.

This approach is simple and fast, but it does have its failings. For one, it doesn't support intersections of more than two contours. This is, of course, possible, but generally doesn't happen often in the data we have dealt with. Another side effect is a smoothing of the contours, which is dependent on $\delta$. In our case this is actually desirable, as the contours we generally deal with are rather noisy, which is why we have to perform the intersection removal in the first place.

## Single component reconstruction

The single component reconstruction algorithm we use is found in [2]. It takes planar contours in adjacent slices as input and outputs a series of triangles, or "tiles", forming a surface between the contours. It supports branching and conforms to all of the criteria as they apply to single component reconstructions.

## Determine conflict points

Once all components between two slices are found, we use a modified sweep line to find all tile edges of different colors

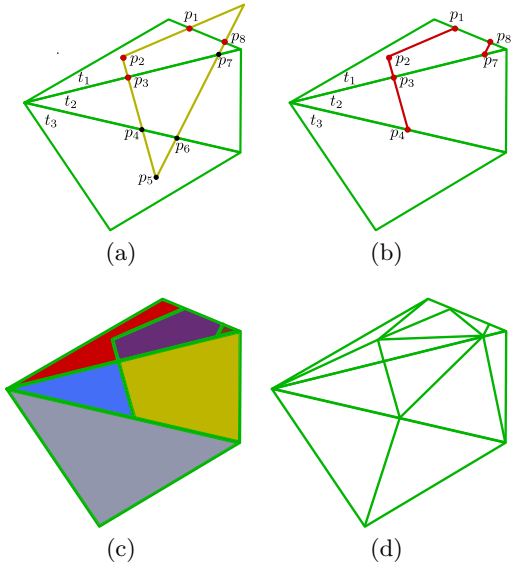



**Figure 5: Steps of intersection removal algorithm. (a) Conflict points on the green tile are detected. (b) Cut paths are traced. (c) New polygons are induced by cut paths. The polygons are colored for clarity. (d) After triangulation of the polygons**

that are closer than $\delta$ in the xy plane. In addition, all tile vertices that are inside of a tile of another component (still in the xy projection of the tiles) are considered. We call these potential conflict points "approach" points. They are marked and stored in a data structure that maps the approach point to the two tiles and every edge passing through the point. There will usually be exactly two edges unless the approach point is at a tile vertex, in which case the number of edges is greater since every tile vertex touches at least two tiles.

We consider conflict points at only these approach points. We determine whether a point is conflicting or not by examining the minimum distance between the different colored edges on which the approach points lie. Then, if the minimum distance is less than $\delta$, we mark each approach point as conflicting.

Figure 5(a) shows a yellow tile and three green tiles. The algorithm finds all approach points (black dots) and then determines which of these are conflict points (red dots).

## Trace tile cuts

The algorithm for tracing tile cuts is as follows:

For each conflict point, we must trace out cut polylines that we will use to induce new polygons that will then be triangulated. The way this is done is to start at a conflict point $p$ and find its projection onto the yellow component to get $p^y$ (line 3). $p^y$ will be on a yellow tile's boundary. Now follow the points on the boundary of the tile that have been marked as intersections from $p^y$ to the exit point where the yellow tile exits the green tile (lines 8-10). Each point encountered in this trace are added to the polyline (line 14).

This can be seen visually in figure 5(a). Point $p_1$ is a conflict point. The algorithm builds an ordered array of points following the yellow tile from $p_1$ all the way to point

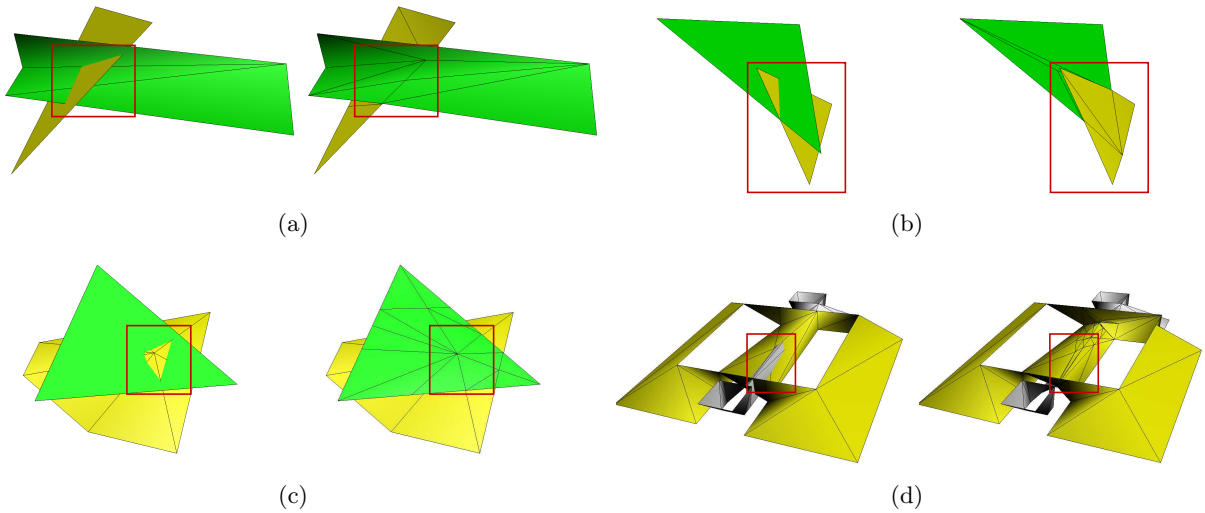



**Figure 6: Examples showing various interesting cases of intersection. (a) A classic intersection except that the back green tile is vertical. (b) A case where conflict points occur at a tile vertex rather than intersections between tile edges. (c) A slightly more complicated case containing conflict points both at tile edges and at vertices. (d) Intersection removal of a branching topology. The yellow component branches from the top slice to the bottom slice and the gray component is the opposite, resulting in a complex intersection that cannot be removed by simply shifting in the xy plane.**

**Algorithm 1** TRACE_CUTS

```
1:  cuts := empty array of polylines
2:  for all points p in conflict points do
3:      p^y := projection of p onto yellow component
4:      t^y := yellow tile containing p^y
5:      t^g := green tile containing p^g
6:      polyline := empty array of points
7:      push p onto polyline
8:      dir := direction to travel on ∂t^y to go to interior of t^g
9:      boundary := ordered intersections on ∂t^y
10:     for all approach points q^y in boundary do
11:         if q^y' ∈ t^g' ∪ ∂t^g' then
12:             q^g := projection of q^y onto green component
13:             q := q^g'
14:             push q onto polyline
15:         end if
16:     end for
17:     push polyline onto cuts
18: end for
19: return cuts
```

$p_8$. Then it loops over each point in the array checking to see if the current point $p$ is inside of the projection of the green tile ${t_1}'$. If it is, then the point is added to the cut. So the cut from point $p_1 = [p_1, p_2, p_3]$. All cuts can be shown in red in figure 5(b).

Cut polylines are specific to a tile. So cuts for the green component's tiles are first found, and then cuts for the yellow component. These polylines are superimposed onto the tiles to generate a set of induced polygons (figure 5(c)). Even though the cuts are different for green tiles vs. yellow tiles, the projection of green tiles and cuts will be identical to that of the yellow.

Figure 5 shows an interesting case in that if the induced polygons are triangulated naively, an illegal triangulation can result. Consider point $p_4$ in the figure. There will be a triangle vertex at this point due to the triangulation of tile $t_2$. If, then, it is not a vertex in the triangulation of tile $t_3$, then the triangulation will not be legal. Because of this, the algorithm checks for any point on a tile edge that is involved in the triangulation of any adjacent tile, and induces that point onto all adjacent tiles. This ensures legal triangulations.

## Triangulate polygons

At this point we have polygons that are ready to be triangulated into a new induced surface. An intuitive approach would be to simply run a constrained Delaunay 2D triangulation on the xy projection of all induced points on the tile. This has two problems: first, the tiles can be vertical and thus the projections of the triangles are degenerate and second, inducing points on edges of triangles causes a large number of collinear points.

The first problem can be addressed by checking to see if the tile is vertical before triangulation. If it is, then rotate by 90 degrees and then triangulate. Happily, the tiles are still coplanar and so we can safely rotate the tile with induced points without worry of causing additional degeneracies.

The second problem is more troublesome. The combined problem of collinear points coupled with numerical error causing possible triangle edges outside of the original tile requires something more than a naive approach. Our solution is to maintain a data structure mapping points to the edges of the original tile from which they were induced. Now a numerically error-prone check for collinearity is perfectly safe by simply doing a hash lookup for each point and comparing the original edges. If all three edges are the same then the points are collinear. If not, then they are not collinear, provided the original tiling algorithm returns non-degenerate

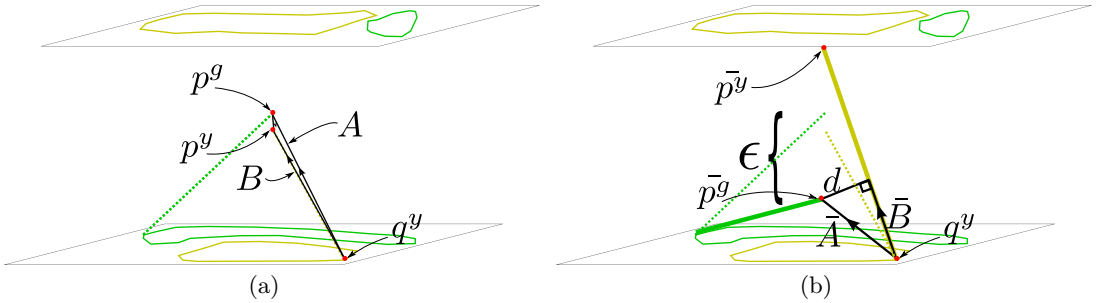



**Figure 7: Calculation of $\epsilon$. A and B are vectors from $q^y$ to the original conflict points. $\bar{\mathbf{A}}$ and $\bar{\mathbf{B}}$ are vectors from $q^y$ to the resolved conflict points. $\epsilon$ is calculated using these vectors and input minimum separation distance parameter $\delta$.**

tiles (which it does in our case).

We used a simple ear-cutting algorithm [23] using this data structure to ensure legal triangulations. Even with the check, an additional modification is required to ensure numerical stability. That is to first generate triangles involving at least one unused collinear point. Without this, the result often includes very long thin triangles if at least one induced point is very close to an existing vertex.

## Adjust z values

The result of triangulating induced polygons is an induced triangulated surface, which still has conflict points between components. At this point we can adjust z values of all conflict points in the new triangulation to separate component surfaces. Each conflict point $p$ is checked and its two associated component points $p^g$ and $p^y$ are given new z values as follows:

$$p_z^g = \frac{p_z^g + p_z^y}{2} + s^g\epsilon \tag{1}$$

where

$$s^g = \begin{cases} -1 & z^g > z^y \\ 1 & z^g < z^y \end{cases} \tag{2}$$

$p_z^y$ is calculated similarly.

$\epsilon$ and $\delta$ are related but distinct. $\delta$ is the input parameter of minimum separation distance between components. $\epsilon$ is the distance along the z axis to move conflict points such that the new surfaces will be separated by $\delta$. Determining $\epsilon$ to achieve the desired minimum distance $\delta$ between components is done as follows. A conflict point $p^y$ is either an induced point on an edge or a vertex. Let $\mathbf{B}$ be the vector $p^y - q^y$ where $q^y$ is either an induced point or vertex such that $p_z^y$ is between $q_z^y$ and $\mathscr{Z}(p^y)$. See figure 7. Further, let $\mathbf{A}$ be the vector $p^g - q^y$. Now let $p^{\bar{y}} = \{p_x^y, p_y^y, p_z^y + s^y\epsilon\}$ and $p^{\bar{g}} = \{p_x^g, p_y^g, p_z^g + s^g\epsilon\}$. And lastly, $\bar{\mathbf{B}} = p^{\bar{y}} - q^y$ and $\bar{\mathbf{A}} = p^{\bar{g}} - q^y$. Distance $d$ is

$$d = \frac{|\bar{\mathbf{A}} \times \bar{\mathbf{B}}|}{|\bar{\mathbf{B}}|} \tag{3}$$

Substituting for $\bar{\mathbf{A}}$ and $\bar{\mathbf{B}}$ and assuming that $s^g = 1$ we get

$$\begin{aligned} d^2 = \quad & ((A_y(B_z - \epsilon) - (A_z + \epsilon)B_y)^2 \\ + \quad & ((A_z + \epsilon)B_x - A_x(B_z - \epsilon))^2 \\ + \quad & (A_xB_y - A_yB_x)^2)/(B_x^2 + B_y^2 + (B_z + \epsilon)^2) \end{aligned} \tag{4}$$

Factoring $\epsilon$ yields a quadratic:

$$\begin{aligned} 0 = \quad & \epsilon^2((A_y + B_y)^2 + (A_x + B_x)^2 - d^2) \\ + \quad & \epsilon(2)((A_x + B_x)(A_zB_x - A_xB_z) \\ & -(A_y + B_y)(A_yB_z - A_zB_y) - d^2A_z) \\ + \quad & (A_yB_z - A_zB_y)^2 + (A_zB_x - A_xB_z)^2 \\ & +(A_xB_y - A_yB_x)^2 - d^2(B_x^2 + B_y^2 + B_z^2) \end{aligned} \tag{5}$$

We then substitute $\delta$ in for $d$ and find the two roots for $\epsilon$. It is possible that both roots yield shifts in the direction of $\mathscr{Z}(p^g)$. This occurs when the surface intersections are gross enough to cause conflict points that are more than $\delta$ apart. So we choose the solution for $\epsilon$ with the greatest value and then multiply by $s^g$ (since we assumed that $s^g = 1$).

There is a denominator on the right hand side of (5) which we have omitted for brevity. But it should be clear from (4) that the denominator is zero only when $\bar{\mathbf{B}}$ is degenerate which cannot happen since $p^y$ is moved in the direction of $\mathscr{Z}(p^y)$ for $\epsilon > 0$.

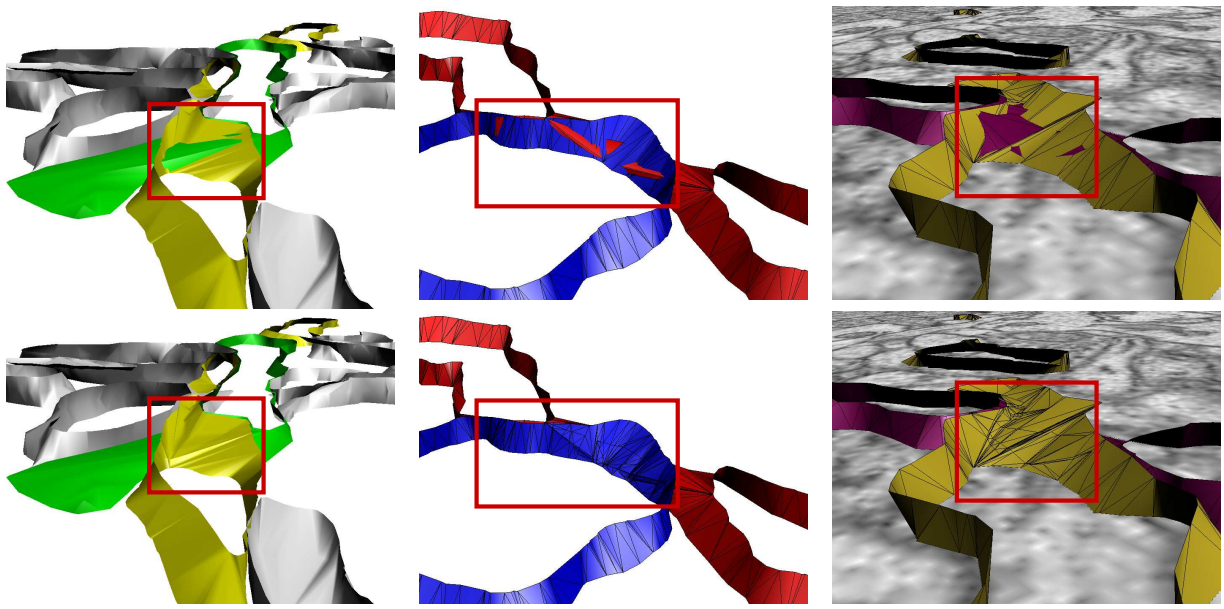
THEOREM 4. *$\epsilon < |p^g - \mathscr{Z}(p^g)|$ and $\epsilon < |p^y - \mathscr{Z}(p^y)|$.*

This theorem bounds $\epsilon$ by the distance from the conflict points to the slices. In other words, no value of $\epsilon$ can cause a point shift in z such that the point crosses a slice boundary. This is important because any such shift would cause the reconstruction to violate criterion 3, not to mention all the criterion's dependent guarantees.

## Computational complexity

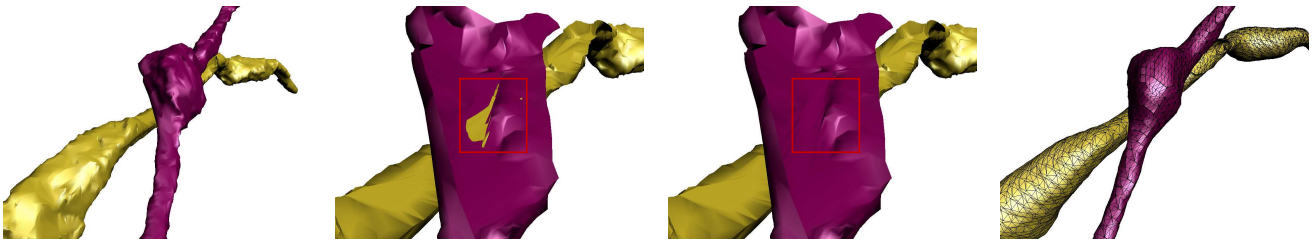
The computational complexity of our algorithm is bounded by the number of conflict points found. The maximum number of conflict points between the boundaries of two triangles is 12: 6 for the xy-plane intersections and 6 for the vertices. Thus, if a reconstruction generates $m$ components, each with $n_i$ triangles, then the maximum number of conflict points is $12n_in_j$ where $n_i$ and $n_j$ are the two largest numbers of triangles in a single component. We state the computational complexity, then, as being $O(n^2)$ where $n$ is the largest number of triangles in a single component.

In practice we have found that there are far fewer conflict points, even in highly tortuous datasets. The reconstruction shown in figure 9 used 21 slices. Between slices 4-5 there were 1352 total triangles before intersection removal. Among these triangles there were 1532 detected conflict points. Tiling these two slices without intersection



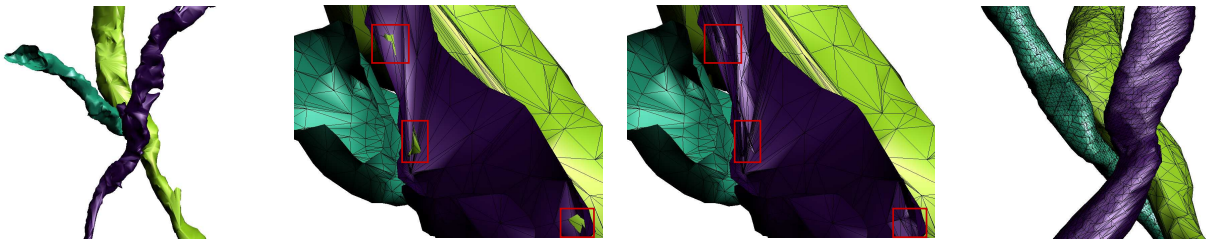
(a) (b) (c)

**Figure 8: Results of running intersection removal on various portions of neuronal contour data. (a) Intersection removal of intersections shown in figure 1(c). (b) Before and after intersection removal with triangle edges visible. (c) Result of intersection removal is shown on top of the original ssTEM data.**



(a) (b) (c) (d)

**Figure 9: Results of running intersection removal on two axons that intersect. (a) Two axons whose reconstructions intersect between slices. (b) Zoomed in with part of the top axon cut away to reveal the intersection. (c) Result of intersection removal. (d) After smoothing.**



(a) (b) (c) (d)

**Figure 10: Results of running intersection removal on two axons that intersect with a third axon in close proximity. (a) Three axons coming very close to each other in the same region. (b) With part of the axons cut away to reveal the intersection between two of the axons. Wireframe is shown superimposed on the surface. (c) Result of intersection removal. Notice the bump in the upper-left box left over from removing intersections. (d) After smoothing.**

removal took 1.07 seconds. Adding intersection removal increased the time to 1.59 seconds.

## Smoothing

One additional step in our pipeline is that of smoothing the surfaces. Initial surface reconstruction can introduce numerically-troublesome thin triangles, and the intersection removal adds $O(n^2)$ triangles. So at completion of intersection removal we run the surfaces through our quality improvement pipeline, which includes edge contraction using the QSlim software package [16] after which we decimate [27] and improve triangles [4]. These tasks are done using a library version of our Level Set Boundary Interior and Exterior Mesher [12] which is also embedded in our Volume Rover software package [13].

# 5. RESULTS AND DISCUSSION

We have implemented a fixed-$\epsilon$ version of the intersection removal algorithm along with a slightly modified version of the single component reconstruction algorithm found in [2].

Figure 6 shows some interesting cases of intersection between tiles (though it is by no means exhaustive). Figure 6(a) shows intersection of a vertical tile. This requires the tile to be rotated by 90 degrees before being re-triangulated after new points are induced. 6(b) and (c) show conflict points at only tile vertices and not xy-plane intersections. As can be seen this case is handled since conflict points are checked at tile vertices in addition to xy-plane intersections. 6(d) shows a branching case. The yellow tile is branching from one contour above to two contours below, while the gray tile is branching opposite. The intersection is removed without treating the branching structure as a special case. In fact, most cases of egregious intersections in real neuronal data were caused at locations where branching occurred. All three examples in figure 8 involve branching of at least one of the components.

Figure 8 shows results of intersection removal from reconstructions of the hippocampal region of the brain. The contours used were hand-traced from 4K x 4K pixel resolution ssTEM images. Image pixels are approximately 2 nm square and inter-slice spacing is 45 nm. We show results from various regions of the dataset at different slices and using different components. The intersection removal algorithm is run immediately after all components between two slices are reconstructed. Only one pass over conflict points is made and, as can be seen, no additional conflict points are generated. The actual number of conflict points was significantly fewer than the upper bound in each case, roughly 1/10th of the $O(n^2)$ upper bound in cases where the components intersect.

Without correct topologies, multiple component modeling is severely limited, and this work provides a solution to this important problem. Using single component reconstructions that adhere to certain guidelines, the method presented in this paper can remove intersections between components correctly and robustly. It also guarantees a minimum separation distance between all components. The algorithm is efficient, performing the intersection removal in $O(n^2)$ time and using highly efficient 2D geometric calculations.

One improvement that needs to be made is the removal of criterion 5. This criterion prohibits intersection removal between nested components. In the case of neuronal modeling, so-called intracellular components such as endoplasmic reticulum and mitochondria are treated separately. But to get a truly accurate model, these will need to be included.

## Acknowledgements

We thank Dr. Kristen Harris from the Center for Learning and Memory and Institute for Neuroscience at UT Austin for use of the high resolution data. Thanks also to Dr. Justin Kinney and Dr. Tom Bartol at the Salk Institute and Andrew Gillette at UT Austin. We used the excellent CGAL [1] library in our implementation of both reconstruction and intersection removal software. This research was supported in part by NIH contracts R01-EB00487, R01-GM074258, and a grant from the UT-Portugal colab project.

# 6. REFERENCES

[1] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.

[2] C. L. Bajaj, E. J. Coyle, and K. Lin. Arbitrary topology shape reconstruction from planar cross sections. *Graph. Models Image Process.*, 58(6):524–543, 1996.

[3] C. L. Bajaj and A. Gillette. Quality meshing of a forest of branching structures. In *Proceedings of the 17th International Meshing Roundtable*, pages 433–449. Springer-Verlag, October 2008.

[4] C. L. Bajaj, G. Xu, R. J. Holt, and A. N. Netravali. Hierarchical multiresolution reconstruction of shell surfaces. *Computer Aided Geometric Design*, 19(2):89 – 112, 2002.

[5] G. Barequet, M. T. Goodrich, A. Levi-steiner, and D. Steiner. Contour interpolation by straight skeletons, graphical models. In *Graphical Models, v.66 n.4*, pages 245–260, 2004.

[6] G. Barequet and M. Sharir. Piecewise-linear interpolation between polygonal slices. In *Computer Vision and Image Understanding*, pages 93–102, 1994.

[7] G. Barequet and A. Vaxman. Nonlinear interpolation between slices. In *SPM '07: Proceedings of the 2007 ACM symposium on Solid and physical modeling*, pages 97–107, New York, NY, USA, 2007. ACM.

[8] G. Barequet and A. Vaxman. Reconstruction of multi-label domains from partial planar cross-sections. In *SGP '09: Proceedings of the Symposium on Geometry Processing*, pages 1327–1337, Aire-la-Ville, Switzerland, Switzerland, 2009. Eurographics Association.

[9] J-D. Boissonnat and P. Memari. Shape reconstruction from unorganized cross-sections. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 89–98, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[10] S. Brenner and L. Scott. *The mathematical theory of finite element methods*. Springer Verlag, 2002.

[11] N. T. Carnevale and M. L. Hines. *The NEURON Book*. Cambridge, UK: Cambridge University, 2006.

[12] CVC. Lbie: Level set boundary interior and exterior mesher. `http://cvcweb.ices.utexas.edu/cvc/projects/project.php?proID=10`.

[13] CVC. Volume rover. `http://cvcweb.ices.utexas.edu/cvc/projects/project.php?proID=9`.

[14] J. Edwards. Livemesh: An interactive 3d image segmentation tool. Master's thesis, Brigham Young University, Provo, Utah, May 2004.

[15] H. Fuchs, Z. M. Kedem, and S. P. Uselton. Optimal surface reconstruction from planar contours. *Commun. ACM*, 20(10):693–702, 1977.

[16] M. Garland. Qslim. `http://mgarland.org/software/qslim.html`.

[17] C. M. Hoffmann. *Geometric and Solid Modeling: An Introduction*. Morgan Kaufmann Pub, 1989.

[18] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *INTERNATIONAL JOURNAL OF COMPUTER VISION*, 1(4):321–331, 1988.

[19] J. P. Kinney, J. Spacek, T. M. Bartol, C. L. Bajaj, K. M. Harris, and T. J. Sejnowski. Extracellular space in hippocampus is wider than previously thought. *Manuscript submitted for publication*, 2010.

[20] L. Liu, C. Bajaj, L. O. Deasy, D. A. Low, and T. Ju. Surface reconstruction from non-parallel curve networks. *Computer Graphics Forum*, 27(2):155–163, 2008.

[21] O. Nilsson, D. Breen, and K. Museth. Surface reconstruction via contour metamorphosis: An eulerian approach with lagrangian particle tracking. In *In Proc. IEEE Visualization*, pages 407–414, 2005.

[22] J-M. Oliva, M. Perrin, and S. Coquillart. 3d reconstruction of complex polyhedral shapes from contours using a simplified generalized voronoi diagram. *Computer Graphics Forum*, 15(3):397–408, 1996.

[23] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1994.

[24] K. E. Sorra and K. M. Harris. Overview on the structure, composition, function, development, and plasticity of hippocampal dendritic spines. *Hippocampus*, 10(5):501–511, 2000.

[25] R. G. Thorne and C. Nicholson. In vivo diffusion analysis with quantum dots and dextrans predicts the width of brain extracellular space. *PNAS*, 103(14):5567 – 5572, 2006.

[26] D. Wang, O. Hassan, K. Morgan, and N. Weatherill. Efficient surface reconstruction from contours based on two-dimensional delaunay triangulation. In *In Proc. Int. J. Numer. Meth. Engng*, pages 734–751, 2006.

[27] Y. Zhang, C. L. Bajaj, and G. Xu. Surface smoothing and quality improvement of quadrilateral/hexahedral meshes with geometric flow. In *In Proceedings, 14th International Meshing Roundtable*, pages 449 – 468. John Wiley & Sons, 2005.

# APPENDIX

Our proofs rely on an additional theorem from [2], which we restate here in a slightly weaker formulation, but sufficient for our purposes:

THEOREM 5. *([2], Theorem 2) For every point $p^g$ on the surface $\partial C^g$ of the green component,* $1 \leq |\mathscr{C}^g(p^g)| \leq 2$.

PROOF OF LEMMA 1. Since $|\mathscr{C}(p^g)| \supset |\mathscr{C}^g(p^g)|$, then $1 \leq |\mathscr{C}^g(p^g)|$ by theorem 5. We prove that $|\mathscr{C}^g(p^g)| \leq 2$ by contradiction. Suppose $|\mathscr{C}^g(p^g)| > 2$. Then by the pigeonhole principle, at least 2 contours must exist on some adjacent slice. These contours share the projection $p_s^g$ onto the slice of point $p_s^g$. This violates criterion 4. □

PROOF OF LEMMA 2. If the 2 containing contours exist on the same slice, then they share the projection $p_s^g$ onto the slice of point $p_s^g$. This violates criterion 4. □

PROOF OF LEMMA 3. This is a proof by contradiction. Consider a point $p^g \in S^g$. Suppose that $p^g$ is a conflict point. Then there exists some point $p^y$ such that $p^y \in \mathscr{B}(p^g)$ (for the moment, set aside the fact that $\mathscr{Z}(p^g)$ is undefined). By lemma 1 we know that the projection of point $p^g$ can have no more than 2 containing contours. By virtue of being in $S^g$, $p^g$ indeed has 2 containing contours, both of which belong to the green component by definition. Since $p^y \in \mathscr{B}(p^g)$, the projections of $p^y$ onto each adjacent slice are within $\delta$ of the projections of $p^g$. Then the projection of $p^y$ onto $\mathscr{C}(p^y)$ (by theorem 5 there must be at least one) is within $\delta$ of a green contour $\mathscr{C}(p^g)$ which contradicts criterion 4. Thus no point $p^y$ exists and $p^g$ is not a conflict point. □

PROOF OF LEMMA 4. By lemma 3, $p^g \in U^g$ and thus cannot be on a vertical tile edge. Then, by criterion 2, a vertical line passing through $p^g$ passes through no other point. □

PROOF OF THEOREM 1. We first prove that if a conflict point $p^g \in \partial C^g$ exists, then the two components are, at some point, within $\delta$ of each other. Let $p^g \in \partial C^g$ be a conflict point. Let $\Gamma$ be the vertical path from $p^g$ to the slice at $\mathscr{Z}(p^g)$. By definition of a conflict point, $\Gamma$ will pass within $\delta$ of some point $p^y \in \partial C^y$ before reaching the slice. By criterion 2, $\Gamma$ intersects $\partial C^g$ exactly once, at $p^g$, and since every point inside the planar contour $\mathscr{C}(p^g)$ is inside the green component, every point on $\Gamma$ between $p^g$ and the projection of $p^g$ onto $\mathscr{Z}(p^g)$ is inside the green component. Therefore, some point $q^g \in (\Gamma \cup C^g)$ is within $\delta$ of $\partial C^y$.

If two components are within $\delta$ of each other, then by definition there exists a conflict point. □

PROOF OF COROLLARY 1. If $p^g$ is the intersection point between two components it is by definition a conflict point. □

PROOF OF THEOREM 2. Let $p^g \in U^g$ and let $\bar{p^g}$ be $p^g$ shifted by $\epsilon$ in the direction of $\mathscr{Z}(p^g)$. $\mathscr{B}(\bar{p^g}) \subset \mathscr{B}(p^g)$ and therefore $\{p^y | p^y \in \mathscr{B}(\bar{p^g})\} \subset \{q^y | q^y \in \mathscr{B}(p^g)\}$. Therefore the number of conflict points will only decrease. □

PROOF OF THEOREM 3. We prove this by contradiction. Suppose there exists a single pair of conflict points $p^g \in \partial C^g$ and $p^y \in \partial C^y$ on the interiors of triangles $t^g \in \partial C^g$ and $t^y \in \partial C^y$, respectively. Further, suppose that every point $p \in (\partial C^g \cup \partial C^y)$ is not a conflict point. Then $|p^g - p^y| < \delta$ while the minimum separation distance between $\partial C^g$ and $\partial C^y$ is at least $\delta$. This violates planarity of the triangles, and therefore there must be some conflict point $p \in (\partial C^g \cup \partial C^y)$. □

PROOF OF THEOREM 4. By definition, the projection $p^{g\prime}$ of $p^g$ onto the slice at $z = \mathscr{Z}(p^g)$ lies inside or on the boundary of a green contour. It can lie on a boundary only if $p^g \in S^g$, so by lemma 3 we know that $p^{g\prime}$ lies on the interior of a green contour. Since the contours are separated by $\delta$, and $q^y$ lies on a yellow contour, then $|p^{g\prime} - q^y| > \delta$. Thus there is some point $\bar{p^g}$ on the vertical line between $p^g$ and $p^{g\prime}$ such that $\frac{|\bar{\mathbf{A}} \times \bar{\mathbf{B}}|}{|\bar{\mathbf{B}}|} = \delta$ proving the first statement. The second follows with a similar argument. □