

Implementation of multi-partitioning surface construction from 2D curves

Eric Shobe · Tao Ju

Department of Computer Science and Engineering
Washington University in St. Louis
One Brookings Drive
St. Louis, Mo 63130
United States

ebs1@wustl.edu · taoju@cs.wustl.edu

Abstract

Presented is an implementation that automatically constructs a 3D surface network from 2D curve networks with arbitrary topology. The surface network generated exactly interpolates the curve network presented on each plane and is guaranteed to be free of gaps and self-intersections. The motivation behind this implementation was to build a high-resolution 3D model of the mouse brain from 2D anatomical boundaries defined on 350 tissue sections. We will be exploring just two sections of the brain at a time, a top and bottom.

When generating the surface networks we wish to exactly interpolate the 2D curve network on both planes, the surface must be a closed mesh without self-intersections or gaps, and the topology of the surface network should agree with the topology of the 2D curve networks.

For example, two neighboring sections of tissue are presented in Figure 1. Figure 1(a,b) show the top and bottom planes of the 2D curve networks from a mouse brain partitioned into anatomical regions which differ in color along boundaries. Our application will automatically generate the surface network, Figure 1(c), free of invalid geometry and also satisfying our requirements.

1 Introduction

Many applications arise for the need to reconstruct a 3D surface model defined from 2D materials on parallel planes. One application in particular is building a 3D model of an anatomical structure. Often when building such a structure, we start from a stack of 2D tissue sections.

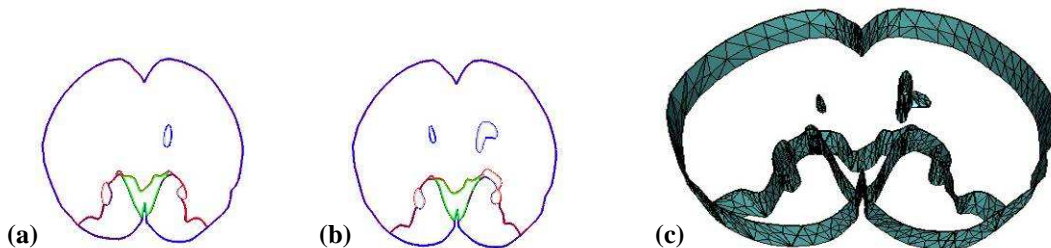


Fig. 1 Constructing a 3D surface network (c) from two parallel sections of the mouse brain partitioned into regions by curve networks (Top plane(a) , Bottom plane (b)).

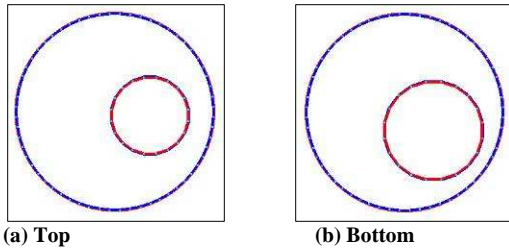


Fig. 2 Two neighboring planes (a,b) of a general topology.

2 Algorithm

Our algorithm computes a layer of surface network between two neighboring planes. Our method proceeds in the following steps:

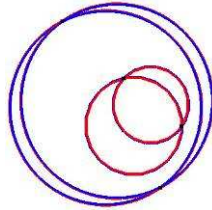


Fig. 3 Projection of two neighboring planes orthogonally onto a common plane, followed by intersection of the top and bottom, then combined to form one mesh.

2.1 Projection and Intersection

2.1.1 Projection: Project the 2D curve networks from each plane orthogonally onto a common plane. For the purpose of our application, each plane will be considered a top or bottom plane. For the purpose of illustration, let Figure 2 (a,b) be considered the general topology of such planes. Figure 3 shows the orthogonal projection of Figure 2(a,b). Projection of both planes is needed in order to compute the intersections.

2.1.2 Intersection and Merging: Each curve network is defined by its geometry and topology based on an input file. The topology of these planes consists of a list of edges. After the projection of the top and bottom planes, edge-edge intersections of the 2D curve networks are computed. New vertices at the intersection points are then added back into each of the

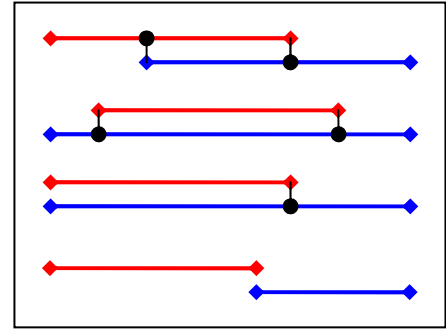


Fig.4 Shown are special cases of collinear edge intersection. Edges are represented by red and blue line segments. Black circles represent each edge split.

planes. Doing this will allow polygons generated between the planes to share common vertices.

Intersections are found and edges correspondingly split with this simple algorithm:

- i. For all edges in top plane, compare with all edges in bottom plane.
- ii. If edges are collinear, find special cases (Figure 4) and split edges accordingly, add edges back to mesh.
- iii. If edges are parallel, do nothing
- iv. If edges are skewed, find intersection, split both edges accordingly, add edges back to mesh.

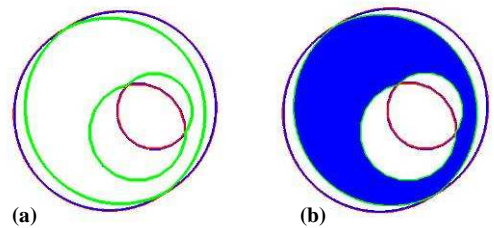


Fig. 5 After intersections and merging, loops are found in the mesh. Once loops are found, regions are established. Outlined in (a) is two loops defined with green borders. A region (b) is found consisting of the two loops in (a).

2.2 Loop and Region Detection

2.2.1 **Loop Detection:** Loops are closed, non-intersecting boundary cycles in the combined 2D curve network. Figure 5(a) shows two loops outlined in green. Loops are found from this simple algorithm:

- i. Find all neighbors for each vertex
- ii. Sort neighbors in a clockwise fashion.
- iii. For each vertex, proceed to the next clockwise neighbor vertex until the starting vertex is reached, then define the loop as that traversed edge list.

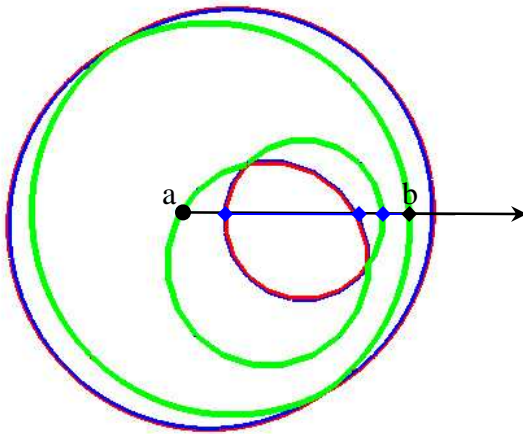


Fig. 6 Starting from a point on an outside loop (a), ray casting is performed until an inside loop is found (b).

2.2.2 **Region Detection:** Regions are then identified on the common plane by using a scan-line algorithm. Regions may be bounded by multiple loop cycles as shown in Figure 5(b). Regions correspond to space between the two planes that project onto the region. Since each region contains an inside loop (counterclockwise orientation), there are inside loops + 1 regions. Regions are found from this simple algorithm.

- i. Let each inside loop correspond to its own region
- ii. For each outside loop (clockwise orientation), ray-cast to determine which inside loop it encloses (Figure. 6). Special cases worth noting are in Figure 7.

- iii. Add outside loop to region containing first inside loop found by ray-casting

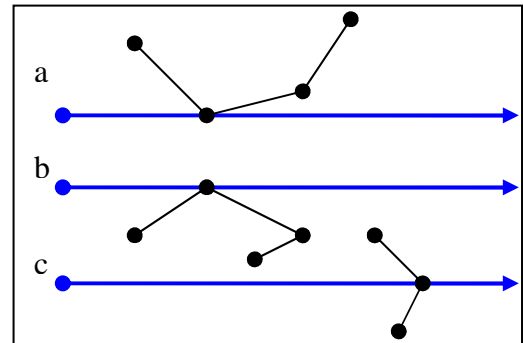


Fig. 7 Special cases for region detection: (a,b) ray-casting intersects with an edge on part of a loop defined by black, this intersection should be discarded. (c) Shows a correct intersection type via ray-casting.

2.2.3 **Color Detection:** Each top and bottom plane consists of 2D curve sections. These curve

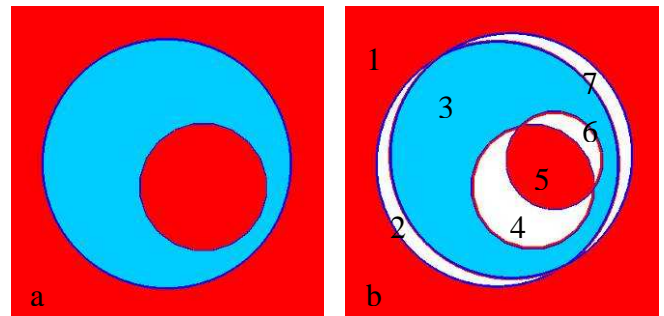


Table 1 – Color Region Index – Before color finding

Before	R1	R2	R3	R4	R5	R6	R7
Bottom Plane Color	Red	Blue	Blue	Red	Red	?	?
Top Plane Color	Red	?	Blue	?	Red	Red	Blue

Table 2– Color Region Index – After color finding

After	R1	R2	R3	R4	R5	R6	R7
Bottom Plane Color	Red	Blue	Blue	Red	Red	Blue	Red
Top Plane Color	Red	Red	Blue	Blue	Red	Red	Blue

Fig. 8 Each plane consists of sections with color defined from the input(a). After region finding it is necessary to find the top and bottom colors(b) of each region. White colors are regions that differ in top and bottom color.

sections are labeled with a color provided from the input. Figure 8(a) shows a sample input with each section having a color from the input. Figure 8(b) show the projection of two planes (top and bottom) of type Figure 8(a) after region finding. After finding each region, it is necessary to compute the top and bottom colors. Figure 8 (Table 1) show the resulting color region index before color finding. Figure 8(Table 2) show the color region index table after color finding. Color finding is simply projecting each region onto the top and bottom planes and finding their respective color.

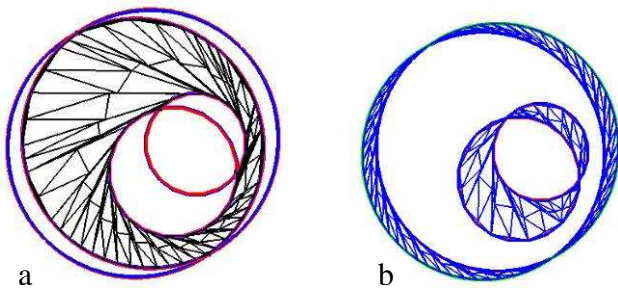


Fig. 9 Triangulation: (a) Shows correct handling of triangulation with holes. (b) Triangulation over each region that has different top and bottom colors.

2.3 Triangulation and Surface Construction

2.3.1 **Triangulation:** Our triangulation must be robust enough to handle holes. The algorithm is as follows for each region:

- i. For each edge, create triangle T with any remaining vertex's for the loops in region, if
 - a. All angles of T < 180°

- b. No vertices of region are inside T
 - c. Edges of T do not intersect any other edge.
- ii. If T is valid, split triangles, as shown in Fig. 9(a), add T to our Triangle list and connect the vertex list accordingly.

2.3.2 **Constructing Surface** There are two major types of polygons when constructing the surface, triangles and quads. The algorithm can be described in two steps.

- i. For regions with different top and bottom plane colors, for every triangle T that exists in the Triangulation list.
 - a. If the vertex is on the top or bottom plane, its z value is 1 or 0 respectively.
 - b. If the vertex is not on the top or bottom plane, its z value is .5
- ii. For regions with the same top and bottom plane colors, for every edge E that exists on the loop.
 - a. If both vertices of the edge exists on top and bottom plane, form a quadrilateral made of triangles.
 - b. If just one vertex exists on both top and bottom, form a triangle.

Figure 10 shows the resulting surface construction.

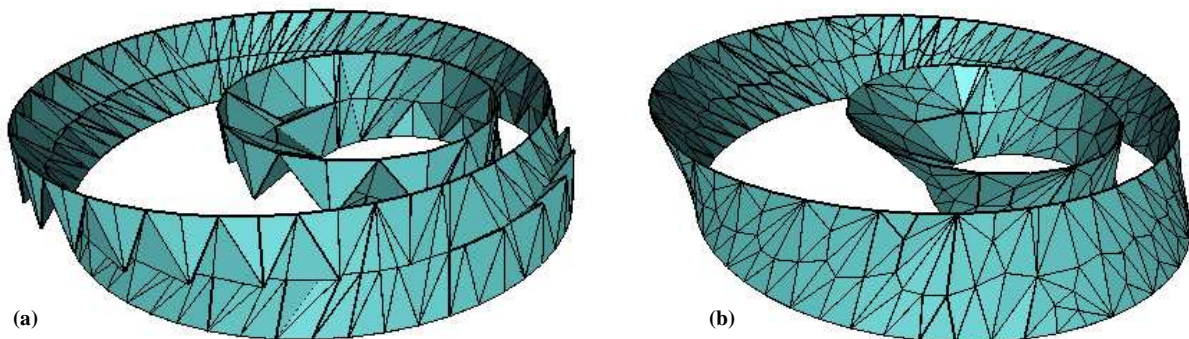


Fig 10 Surface Construction: (a) Shows the top and bottom mesh defined in Figure 3 after Surface Construction without any Laplacian smoothing. (b) Shows the surface construction after 5 iterations of Laplacian smoothing

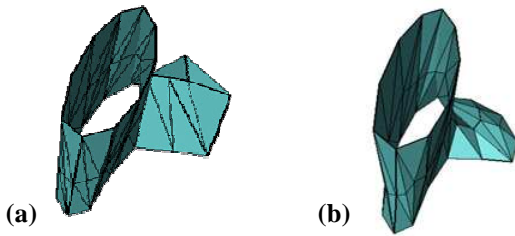


Fig 11 Mouse brain 3D surface before Laplacian (a) and after 5 iterations of Laplacian smoothing (b).

2.4 Laplacian Smoothing: The algorithm for constructing the surface produces jagged appearances where polygons meet at right angles. A Laplacian smoothing operator is computed for every vertex not on original top or bottom plane using the following:

$$v^* = \frac{v_i + \sum \frac{v_i}{n}}{2}$$

Figure 11 shows the results with and without Laplacian smoothing.

3 Results

Here we present the results of our method for building 3D surface networks from 2D curve networks. One section of the mouse brain is used for the results. The algorithm was programmed in C++ and is currently running using a Windows platform running an Intel Pentium® 4 2.53Ghz processor.

The input curve network is the model shown in Figure 1 (a,b) consisting of a top (Figure 1(a)) with 247 vertices and a bottom (Figure 1(b)) consisting of 248 vertices. The following is the time for each step in seconds:

1. **Intersections:** .062 seconds for the Intersection Routine. 38 intersections were found.
2. **Merging:** .062 seconds for the merging of the top and bottom 2D curve networks. The combined 2D curve network consists of 419 vertices.
3. **Loop and Region Finding:** 1.0 seconds for finding loops and regions. 64 regions were found and 66 loops were found.

4. **Triangulation:** .25 seconds for triangulation.
5. **Constructing Surface:** .218 seconds
6. **Laplacian Smoothing:** .469 seconds

Our motivation for this application was to build a high-resolution 3D model of the mouse brain. This implementation reconstructs each neighbor as a 3d model. Future work will progress on connecting each one of these partitions to form a full 3D model. However the total construction of a stack of 350 tissue sections of the mouse brain has been achieved. These tissue sections contained 200404 vertices on 350 sections. The total time for construction was 90 minutes, 41 seconds, averaging 15 seconds between each neighbor planes.

Reference

1. Ju, T., Warren, J., Carson, J., et al: Building 3D surface networks from 2D curve networks with application to anatomical modeling.
2. Antonio, Franklin: Faster Line Segment Intersection, Graphic Gems III.