Reconstruction of surfaces from planar contours through contour interpolation

Kyle Sunderland, Boyeong Woo, Csaba Pinter, and Gabor Fichtinger
Laboratory for Percutaneous Surgery, School of Computing, Queen's University, Kingston, Canada

ABSTRACT

PURPOSE: Segmented structures such as targets or organs at risk are typically stored as 2D contours contained on evenly spaced cross sectional images (slices). Contour interpolation algorithms are implemented in radiation oncology treatment planning software to turn 2D contours into a 3D surface, however the results differ between algorithms, causing discrepancies in analysis. Our goal was to create an accurate and consistent contour interpolation algorithm that can handle issues such as keyhole contours, rapid changes, and branching. This was primarily motivated by radiation therapy research using the open source SlicerRT extension for the 3D Slicer platform. METHODS: The implemented algorithm triangulates the mesh by minimizing the length of edges spanning the contours with dynamic programming. The first step in the algorithm is removing keyholes from contours. Correspondence is then found between contour layers and branching patterns are determined. The final step is triangulating the contours and sealing the external contours. RESULTS: The algorithm was tested on contours segmented on computed tomography (CT) images. Some cases such as inner contours, rapid changes in contour size, and branching were handled well by the algorithm when encountered individually. There were some special cases in which the simultaneous occurrence of several of these problems in the same location could cause the algorithm to produce suboptimal mesh. CONCLUSION: An open source contour interpolation algorithm was implemented in SlicerRT for reconstructing surfaces from planar contours. The implemented algorithm was able to generate qualitatively good 3D mesh from the set of 2D contours for most tested structures.

Keywords: surface reconstruction, contour interpolation, triangulation, radiation therapy

1. INTRODUCTION

Medical image data, such as computed tomography (CT) scans, typically provide 2D cross sectional images that are spaced evenly on individual slices. In procedures such as radiation therapy, it is often required that target structures and organs at risk be identified through the use of planar contours which circumnavigate the structure on the image slices. These structures must be converted into 3D meshes in order to be visualized or to be used for treatment planning and further analyses.

In radiation oncology treatment planning software, an algorithm is often implemented which creates a binary volume (labelmap) from a set of contours, for which an intermediate step is often reconstructing a surface model. This presents a problem however, since every software package uses a different algorithm with different results, and their methods for producing the surface are not disclosed. This results in differences for dosimetry and dose volume histograms (DVH) for the same structures, the effects of which are small in large structures, but are pronounced in small structures[1].

The 3D Slicer[2] software package is a tool that is widely used by clinicians for treatment planning and visualization. SlicerRT[3] is an open source extension that has been developed for the 3D Slicer platform which facilitates dosimetric evaluations, comparisons, dose accumulation modelling, and external beam planning among other research workflows, many of which require correct and consistent surface reconstruction from a set of contours.

The topic of converting contour layers into 3D meshes is a subject that has been studied frequently. One of the major contributions was made by Fuchs et al.[4], when they described the problem of producing an optimal triangulation as a graph problem that can be minimized. They also specified that two of the edges for each triangles in the mesh should span the gap between contours, with the third edge lying directly on the surface of one of the contours.

One issue that needs to be solved by all triangulation algorithms is determining which contours should be connected by triangles between each of the slices. This is referred to as the correspondence problem. Another obstacle to overcome is

the occurrence of branching contours. A branching contour occurs when a single contour on one of the slices must be connected to two or more contours on an adjacent slice[5].

Another issue that can occur is the existence of keyholes within the set of contours. The DICOM standard (RT structure set module) defines the storage of hollow structures as keyholes that are represented as an inner and outer contour connected by an arbitrarily small channel in order to be considered one contour. These keyhole contours can cause problems with contour triangulation algorithms and need to be handled before triangulation can take place.

Our goal is to create an accurate and consistent algorithm for interpolating contours, implemented in the open source SlicerRT extension for the 3D Slicer platform. The algorithm should be able to handle the three main issues that the old implementation of our algorithm had difficulty handling: keyhole contours, rapid changes, and branching. The mesh produced by the algorithm should be a qualitatively acceptable representation of the original structure that is defined by the original set of planar contours.

2. METHODOLOGY

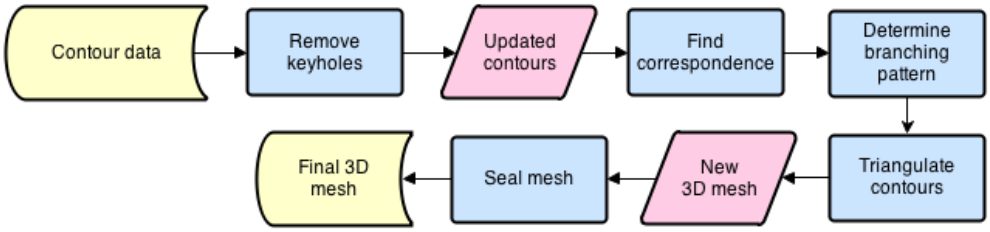

Figure 1: Data flow diagram representing the process of converting a set of 2D contours into a corresponding surface mesh.

We implemented our algorithm for creating 3D closed surface models from planar contours as a Python scripted module for the 3D Slicer platform, as part of the SlicerRT extension. Our algorithm was designed with the intent to preserve the original points and to avoid introducing new points. There is one instance in which this rule is deviated from during the keyhole handling process, since the points need to be removed from within the channel. The 3D mesh construction procedure can be seen in the data flow diagram in Figure 1. The algorithm for contour to surface conversion is to be implemented as a single step in a contour conversion pathway for SlicerRT (see Figure 2).

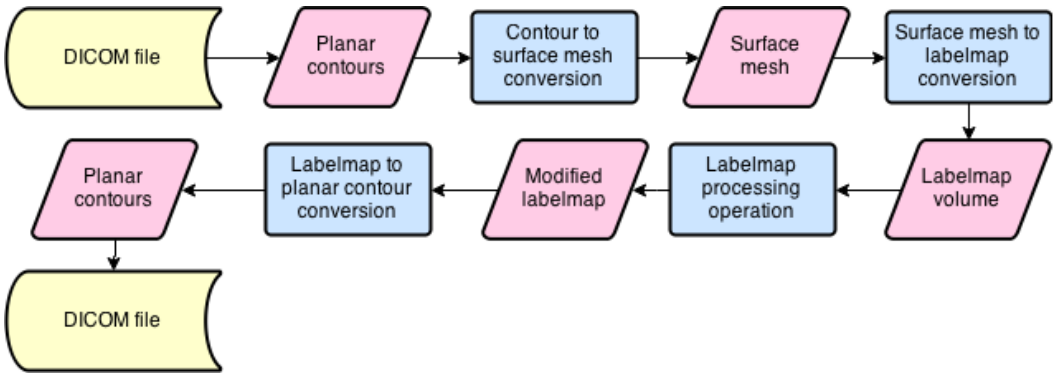

Figure 2: Data flow diagram representing a typical contour analysis workflow. Our algorithm is the contour to surface conversion step.

2.1 Keyholes

The existence of keyholes within contours can cause problems with the process of triangulation, as they can cause the triangulation algorithm to incorrectly connect to points contained within the channel. In order to remove the keyholes

from the contours, each contour is addressed individually in a pre-processing step comparing each point in the contour to all other points in the same contour. If two points in the contour are within a specified threshold distance from each other and not considered to be adjacent to each other, then the two points are noted to be in conflict with each other.

Once all of the points in the current contour have been assessed, the algorithm walks through the list of points for the current contour in order to rebuild the contour and add additional contours as is necessary to remove the keyholes. If a point has not been found to have violated the threshold distance constraint, then it is not part of a keyhole and can be added to the current contour. If a point is encountered which is in conflict with a point whose index location is greater than the current one, this represents the start of a keyhole channel. At the start of a new keyhole, a new contour is added to the current list of contours, which represents the inner keyhole volume. In the keyhole, points that are not in conflict are added to the new contour until a point is encountered that is in conflict with a previously visited point. This represents the closure of a keyhole and the end of the newly added contour. The current contour is then added to a list of completed contours and points are now added to the previous contour. In this way, keyhole contours can be separated into their internal and external contours, which can include multiple and layered keyholes on each slice. After the new contours have been composed, we make sure that they all form closed contours.

2.2 Correspondence

Before the process of triangulation can begin, the correspondence between contours must first be identified so that we can triangulate between the correct contours, while ignoring contours that should not be triangulated together. The process of finding contour correspondence is also required before branching patterns can be calculated. In order to determine whether there is correspondence between contours on different layers, simple bounding-box overlapping was used to define correspondence, as mentioned by Meyers et al[5]. If the bounding boxes of two contours overlap, then the algorithm will consider them to be connected structures when constructing the surface triangulation. One issue that is often present in interpolation algorithms is the presence of internal contours on the same slice. For our algorithm, no distinction is made between the external (counter-clockwise) contours and the internal (clockwise) contours. This means that the internal contours on one layer can connect to the external contours on another.

2.3 Branching

When a contour on one slice corresponds to multiple contours on an adjacent slice, we need to be able to create a triangulation which branches between all three contours. In order to do this, the next step in our algorithm is to compute a branching pattern for contours that possess a correspondence to multiple contours on an adjacent slice. In order to reduce the computation time, only contours that have correspondence identified from the previous step are used when calculating the branching patterns. The points in each of the contours are checked against all of the points contained in their corresponding contours in order to determine which of the corresponding contours is closest to the current point. The points on each of the contours are then divided into separate sections depending on which of the corresponding contours they are closest to. The branching regions are then resolved by performing the triangulation on each of the sections with the closest section located on a corresponding contour, treating these sections as isolated triangulations.

2.4 Triangulation

Our method for triangulation works by creating triangles that span between two corresponding contours on adjacent layers and is based on the PointWalk algorithm which is included in the vtkRuledSurfaceFilter class from the VTK library[6]. While the original PointWalk uses a greedy algorithm to attempt to minimize the length of the edges between contours, the greedy nature of the algorithm means that it will sometimes construct a non-optimal triangulation. Our algorithm variation interpolates between the pairs of contours by finding the triangulation with the shortest spanning edge length through the use of a dynamic programming algorithm. The two line segments which are connected can be considered the nodes in a directed graph, with the node weights represented by the length of the edges spanning the contours (see Figure 3). Each edge in the directed graph represents a possible triangle in the mesh, consisting of two edges denoted by the connected nodes, as well as a third edge from the previous triangle. This ensures that the final surface will not contain any gaps and will be a closed surface. The two axis in the graph are represented by the points that are contained within each of the two connected contours. We then use the graph as a basis for a dynamic programming algorithm with which the edge length between the contours can be minimized. The implementation of this method enforces that any triangle created from the triangulation must have two points on one of the two contour sections and one point on the other.

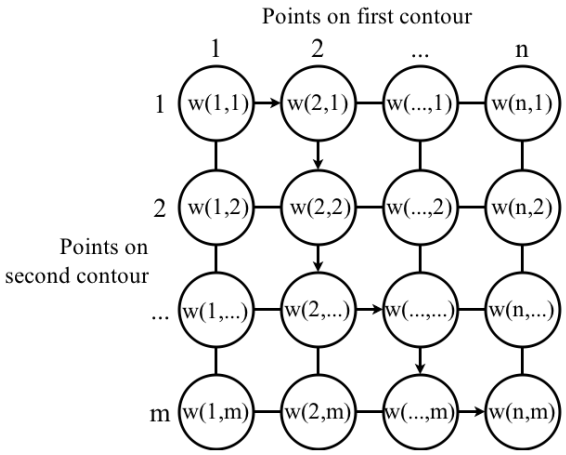

Figure 3: Example diagram of a directed graph representing the triangulation of a contour with 'n' points and a contour with 'm' points. The dynamic programming method finds the path through the graph that minimizes the sum of the weights at each node.

2.5 Seal Mesh

After the triangulation is complete, there are still some gaps that are left in the surface model which need to be sealed. The gaps in the surface are caused by the contours that reside on the top and bottom of the 3D model. The method for determining which contours need to be sealed looks at the triangles that have been produced by the triangulation for each contour. If a contour is connected to triangles both above and below it, then it resides in the middle of the model and does not need to be sealed. The contours that do need to be sealed will only be connected to triangles on only one side of the contour, or on neither side, in the case of a contour that was not triangulated. The final closed surface then is constructed by performing a Delaunay triangulation on the external contours using the vtkDelaunay2D class in the VTK library[6] (see Figure 4).

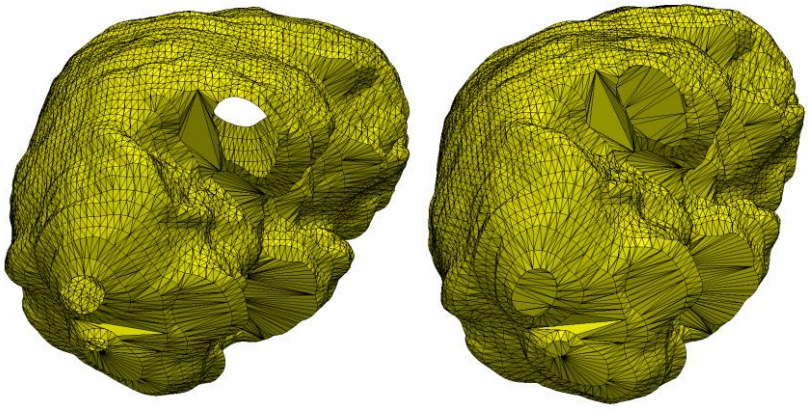

Figure 4: Example mesh containing gaps introduced by contours on the external surface. The image on the left shows the original mesh, while the image on the right shows the same mesh with the external contours sealed.

3. RESULTS

The contour interpolation algorithm was successfully implemented as a Python scripted module in SlicerRT. In order to test the performance of our contour interpolation algorithm we tested the algorithm on contours which have been isolated from structures contained within several different sets of real CT images (see Figure 5).

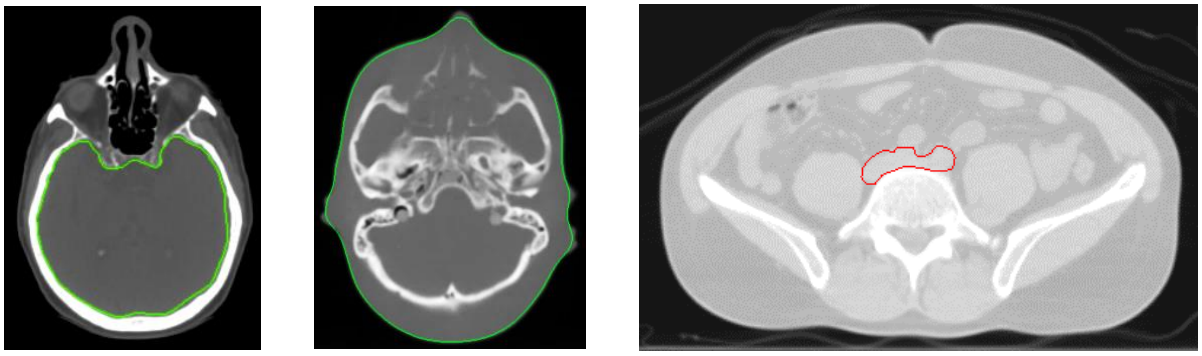

Figure 5. Example CT slices used to produce the meshes in Figure 6, with brain (left), head (center), and vessel (right) contours.

The algorithm was found to produce qualitatively good 3D meshes for most contours such as the examples in Figure 6. We chose to use qualitative analysis on the mesh since quantitative analysis was prohibitively complex to implement in our timeframe. Deliberately complex examples were used to test the capabilities of our algorithm. Once the algorithm is fully implemented in SlicerRT, a fallback mechanism will be implemented in our conversion algorithm in case the mesh fails according to user feedback. If this occurs, the process of contour conversion will temporarily utilize the algorithm which was previously implemented. There were three main issues with the previous implementation that we sought to handle with our algorithm: keyhole contours, rapid changes and branching.

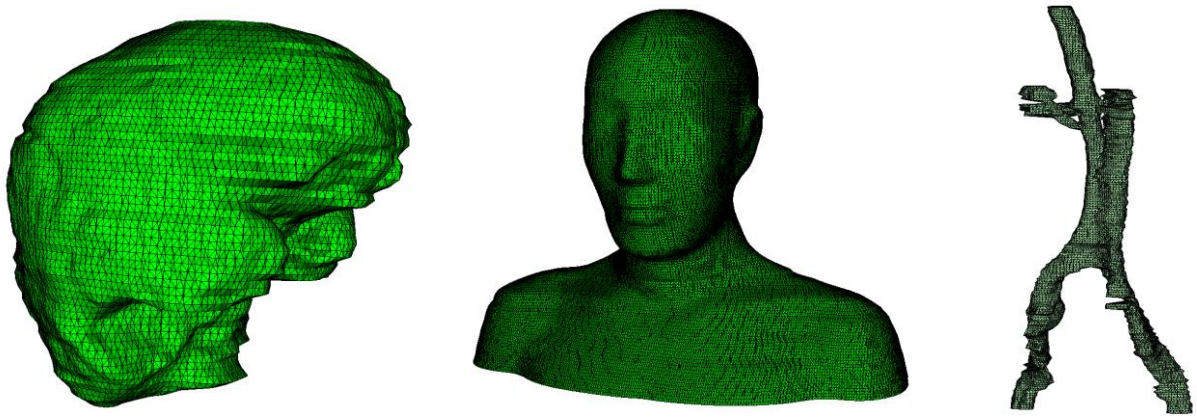

Figure 6. Example mesh generated using the algorithm from the CT scan images shown in Figure 5, from brain (left), head and neck (center), and vessel (right) contours.

In order to test the keyhole removal mechanism of the algorithm, we created a set of fabricated set of contours which contained several layers of keyholes. The keyhole algorithm was executed on the contours using a threshold distance of 0.1mm and an adjacency threshold of 2 points. The method for removing keyholes from contours was found to be effective at dealing with the fabricated examples that were used for testing (see Figure 7).

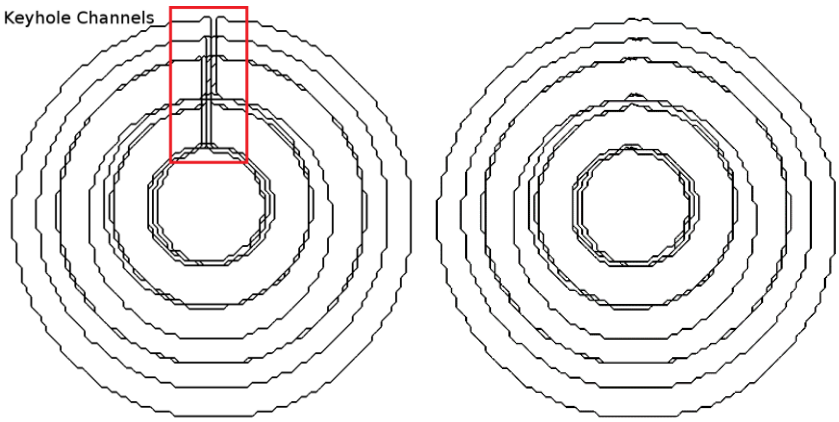

Figure 7. Example of handling keyhole contours. The left image shows the fabricated contours prior to removing the keyholes and right image shows the fabricated contours after removing the keyholes.

One of the problems that was present in the original PointWalk implementation was that because it was a greedy algorithm, it would sometimes fail to create an optimal surface in situations where contours rapidly changed from slice to slice. The dynamic programming approach of our algorithm ensured that the contours were connected in a true length minimization. Problems still occurred however when there were rapid changes from one contour to the next and the contours were not similar in shape or size (Figure 8/left). These cases often caused triangles on the mesh to all converge to a single point at the edge of the smaller contour. This is likely a cosmetic issue, since the meshes are still qualitatively acceptable when rasterized.

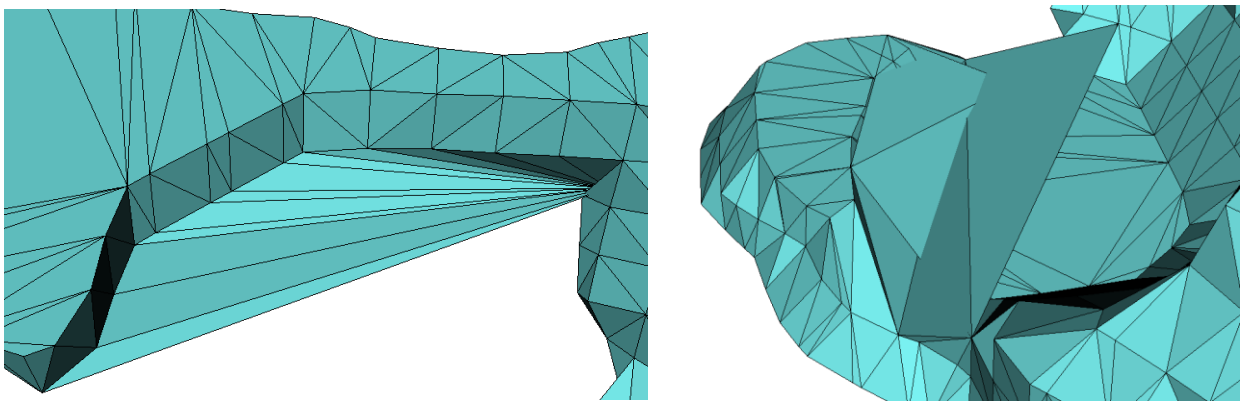

Figure 8. Example of issues with rapid change, causing the surface triangles to all converge on a single point (left), and branching on internal an internal contours, which causes some intersecting mesh triangles (right).

The implemented method for calculating branching patterns was found to work for simple cases (see Figure 9). One example for which the branching algorithm fails is where the algorithm produced unwanted triangles when it was used on top of an inner contour (see Figure 8/right). When calculating the branching patterns, the inner contour can incorrectly alter the division of the contours into branching sections, causing defects such as overlapping triangles. In a similar manner, since the inner contours behave the same way as external contours, the algorithm can sometimes encounter problems in the triangulation when there is a rapid change in contours between layers. This can cause the algorithm to preferentially choose to connect to the closer inner contour, rather than the farther, external one (see Figure 10). Tseng et al.[7] handled the case of internal contours by constructing triangulations for the internal and external contours separately. We opted not use this approach, since there were situations in which the internal and external contours should be triangulated together.

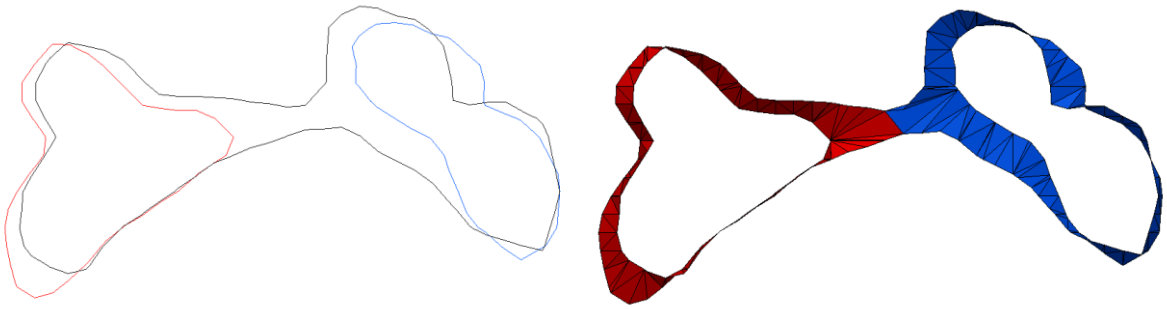

Figure 9: Example of multiple correspondence requiring branching contours. The image on the left shows the larger contour with correspondence to two smaller contours on an adjacent layer. The image on the right shows the contours after being separated into sections and triangulated.

The behavior of the bounding box correspondence can sometimes cause contours that are irregular in shape or that are oriented in a manner that expands the bounding box size to form correspondences with contours that should not be connected. This was found to not be an issue, as the branching algorithm would refrain from connecting these distant contours during the branch calculation step, and would instead prefer to connect the contours that were closer together.

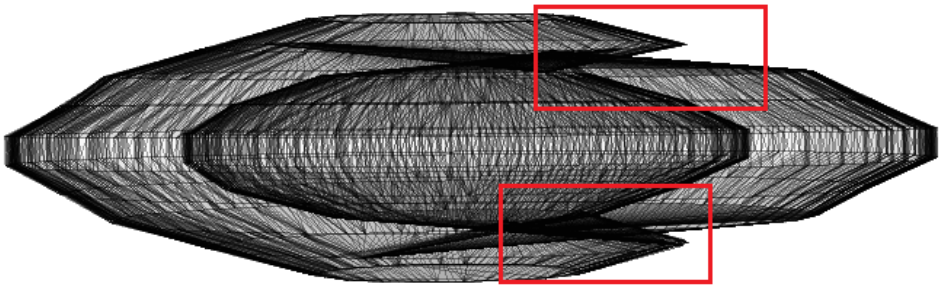

Figure 10. Example of a problem with rapid changes and internal contours, in which the outer contour is connecting to the inner volume, rather than the outer surface. The regions in which the problems occur have been highlighted. The mesh was constructed from the fabricated example contours shown in Figure 7.

4. DISCUSSION

We discovered that our algorithm was able to appropriately handle individual occurrences of the issues of rapid changes, branching and keyhole contours, however the presence of several of these issues in the same location simultaneously was found to cause problems for the final surface mesh. In the case of the issue involving rapid changes and dissimilar contours, a possible method for preventing this was discussed by Sederberg et al.[8] in which the restriction that two points in a triangle must lie on one contour and one point must lie on another could be removed, allowing for triangles to be created in which all three points occurred on the same contour.

Future work will need to be done in order to deal with this problem, as well as handling combined cases of rapid change, branching and keyhole contours. In addition, further work will need to be done on the algorithm before it can be fully integrated into SlicerRT. The algorithm will need to be ported from Python to C++ and must undergo thorough quantitative testing to ensure that the algorithms performance is up to the proper standard.

5. CONCLUSIONS

We implemented an algorithm in SlicerRT as a Python scripted module that converted sets of 2D planar contours into 3D surface mesh and was able to handle the issues of keyhole contours, rapid changes, and branching. The algorithm was found to be able to produce 3D surface mesh that were considered qualitatively good for most of the sets of contours that were tested.

ACKNOWLEDGEMENTS

This work was supported by the Natural Sciences and Engineering Research Council of Canada. Gabor Fichtinger is supported as a Cancer Care Ontario Research Chair in Cancer Imaging.

REFERENCES

[1] Ebert, M., Haworth, A., Kearvell, R., Hooton, B., Hug, B., Spry, N., Bydder, S., and Joseph, D., "Comparison of DVH data from multiple radiotherapy treatment planning systems", Physics in medicine and biology 55(11), 337-346 (2010).

[2] Pieper, S., Halle, M., and Kikinis, R., "3D Slicer," Proc. IEEE International Symposium on Biomedical Imaging, 632-635 (2004).

[3] Pinter, C., Lasso, A., Wang, A., Jaffray, D. and Fichtinger, G., "SlicerRT: radiation therapy research toolkit for 3D Slicer," Med. Phys. 39(10), 6332-6337 (2012).

[4] Fuchs, H., Kedem, Z. M., and Uselton, S. P., "Optimal surface reconstruction from planar contours," Communications of the ACM 20(10), 693-702 (1977).

[5] Meyers, D., Skinner, S., and Sloan, K., "Surfaces from contours," ACM Transactions on Graphics 11(3), 228-258 (1992).

[6] Schroeder, W., Martin, K., and Lorensen, B., [Visualization Toolkit: An Object-Oriented Approach to 3D Graphics], Kitware, Clifton Park, NY, (2006).

[7] Tseng, K. K., and Lu, P., "Construction of three-dimensional models for structural objects from tomographic images," Proc. ISARC 17, (2000).

[8] Sederberg, T. W., Klimaszewski, K. S., and Hong, M., "Triangulation of branching contours using area minimization," Journal of Computational Geometry & Applications 8(4), 389-406 (1998).