Coronary vessel cores from 3D imagery: a topological approach

Andrzej Szymczak[a] and Allen Tannenbaum[b] and Konstantin Mischaikow[c]

[a]College of Computing, Georgia Tech, Atlanta, GA 30332, USA;
[b]Department of Biomedical Engineering, Georgia Tech, Atlanta, GA 30332, USA;
[c]Department of Mathematics, Georgia Tech, Atlanta, GA 30332, USA

ABSTRACT

We propose a simple method for reconstructing thin, low-contrast blood vessels from three-dimensional greyscale images. Our algorithm first extracts persistent maxima of the intensity on all axis-aligned two-dimensional slices through the input volume. Those maxima tend to concentrate along one-dimensional intensity ridges, in particular along blood vessels. Persistence (which can be viewed as a measure of robustness of a local maximum with respect to perturbations of the data) allows to filter out the ‘unimportant’ maxima due to noise or inaccuracy in the input volume. We then build a minimum forest based on the persistent maxima that uses edges of length smaller than a certain threshold. Because of the distribution of the robust maxima, the structure of this forest already reflects the structure of the blood vessels. We apply three simple geometric filters to the forest in order to improve its quality. The first filter removes short branches from the forest’s trees. The second filter adds edges, longer than the edge length threshold used earlier, that join what appears (based on geometric criteria) to be pieces of the same blood vessel to the forest. Such disconnected pieces often result from non-uniformity of contrast along a blood vessel. Finally, we let the user select the tree of interest by clicking near its root (point from which blood would flow out into the tree). We compute the blood flow direction assuming that the tree is of the correct structure and cut it in places where the vessel’s geometry would force the blood flow direction to change abruptly.

Experiments on clinical CT scans show that our technique can be a useful tool for segmentation of thin and low contrast blood vessels. In particular, we successfully applied it to extract coronary arteries from heart CT scans. Volumetric 3D models of blood vessels can be obtained from the graph described above by adaptive thresholding.

1. INTRODUCTION

Medical datasets, including CT scans, are inherently noisy and inaccurate. Algorithms that process such data need to distinguish features resulting from noise from those carrying useful information about the dataset’s structure. Topological tools are a natural choice for this task: one of the major goals of discrete topology is finding and describing persistent features that cannot be removed by small perturbations of the data. Our goal is to apply topological tools to segment thin blood vessels from 3D greyscale imagery. Although the main focus of this paper is coronary arteries in CT scans, we believe our algorithm is general enough to be applied to other types of vascular trees and other kinds of datasets.

Vessel extraction algorithms are receiving significant amount of attention in recent years, as their structure can yield important information for diagnosis and treatment of cardiovascular diseases, which are the most frequent cause of death in the developed nations. An excellent overview of the field can be found in.[1] The algorithm described in this paper can be classified as a ridge-based approach. Contrary to most ridge based approaches, we do not estimate the local characteristics of the vessel directly from the image. We extract a set

---

Further author information: (Send correspondence to Andrzej Szymczak)
Andrzej Szymczak: E-mail: andrzej@cc.gatech.edu
Allen Tannenbaum: E-mail: tannenba@ece.gatech.edu
Konstantin Mischaikow: mischaik@math.gatech.edu

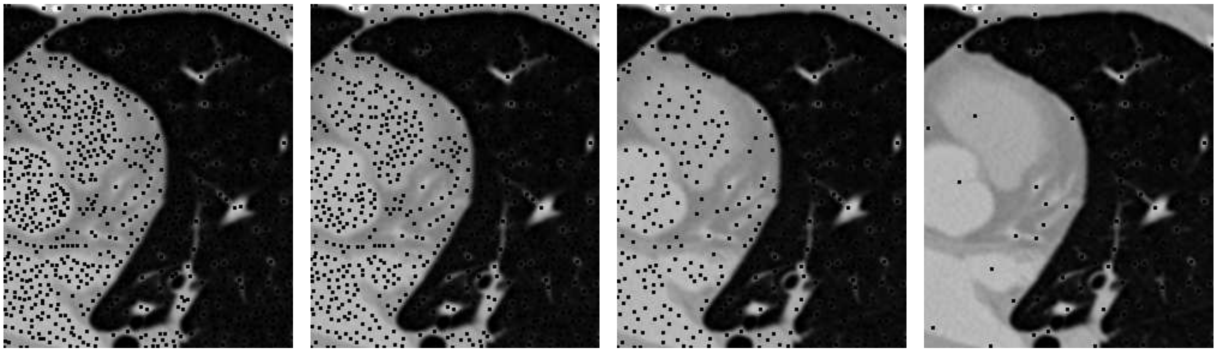

Figure 1. Maxima in a slice through a CT scan found by our algorithm. The figures show maxima of persistence 0, 160, 640 and 2560 (respectively) as black dots (our CT image has dynamic range $[0, 65535]$). The last persistence threshold is the most suitable for use in our algorithm.

of sample points (robust intensity maxima in 2D slices) near the vessel cores and use them to reconstruct the vascular trees.

The structure of the paper is as follows. In Sections 2 and 3 we describe the two major stages of our algorithm: computing the set of robust maxima on 2D slices through the data and reconstruction of blood vessel trees from those robust maxima. Section 4 describes how the trees can be turned into volumetric models using adaptive thresholding. Experimental results obtained for clinical datasets are discussed in 5. Finally, we discuss future research directions in Section 6.

2. COMPUTING ROBUST MAXIMA

The purpose of this step is to generate points near the centerlines of blood vessels. Blood vessels have larger intensity than the surrounding tissue and are essentially one-dimensional structures. Therefore, a blood vessel that intersects a 2D slice through the input data produces a local maximum of intensity on the slice near the intersection point. The converse is generally not true because of noise and presence of other features: there are several local maxima that do not correspond to blood vessels (Figure 1, left). In fact, noise makes the distribution of the local maxima rather unstructured. We deal with this problem by filtering out the unwanted local maxima using persistence. This allows to reject local maxima due to noise while mostly preserving the ones defined by blood vessels. Persistent local maxima tend to be aligned with the blood vessels, as shown in Figures 1 and 2. Our algorithm computes such local maxima on all axis-oriented 2D slices through the input volume.

Persistence can be thought of as a measure indicating how robust a local maximum is with respect to perturbations of the data. Low persistence maxima can be removed by applying a smaller perturbation. This makes persistence a natural tool for deciding which local maxima arise because of noise and which correspond to true features of the input volume. Persistent maxima that can intuitively be defined using a flooding metaphor as follows. Treat the intensity as the depth of a terrain, i.e. make height equal to minus intensity. Start flooding the terrain by pouring water into a dip of the terrain x, corresponding to local maximum of intensity, until the water depth above the dip reaches the desired persistence threshold M. We declare x as a persistent maximum of the intensity if and only if the maximum depth of the pool obtained this way is no more than M, equivalently, no dips lower than x have been flooded. The persistence threshold M suitable for our approach should grow together with the amplitude of noise expected to be encountered in the input datasets. It is currently one of the user-specified parameters required by our procedure, although one could speculate that it should be possible to determine automatically based on the statistics of the input data.

In order to efficiently find robust maxima, we use a variant of the procedure of.[2] Starting from the empty set, we keep inserting voxels in order of decreasing intensity. As we do that, we keep track of the connected components of the set of voxels inserted so far using the union-find datastructure. With each connected component we maintain a record of attributes that includes the location of the voxel of maximum intensity within the component

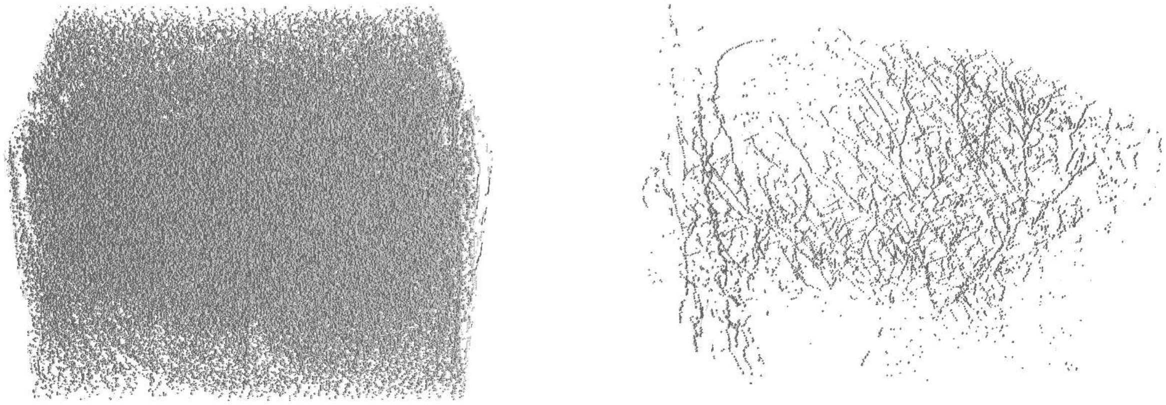

Figure 2. Distribution of all maxima (left) and robust maxima (right). The one on the right is obviously more structured (some blood vessels can be observed, although clutter is still present).

and (optionally) the size of the component. Whenever a new voxel of intensity I is inserted, the union-find data structure as well as the connected component attributes are updated. All connected components of vertex neighbors of the new voxel brighter than I are merged to one. Its size is the sum of sizes of the merged components plus one. Its brightest voxel is computed as the brightest voxel among the merged components' maximum intensity voxels. If the intensity of the maximum intensity voxel $\mathbf{v}$ in one of the merged component exceeds $I + M$ (where M is the prescribed persistence threshold), $\mathbf{v}$ is marked as a persistent maximum.

The above procedure can be augmented with additional heuristics allowing one to reject some of the persistent maxima. For example, one can put a bound on the size of the connected component whose maxima can possibly be marked as persistent. Such a bound on size could be made proportional to the expected maximum width of a blood vessel to be extracted.

The running time of the above procedure is $O(n \log n)$ where n is the number of all voxels in the input volume. The dominating cost is that of sorting all voxels in each of the slices.

3. VESSEL RECONSTRUCTION FROM THE ROBUST MAXIMA

In this section we discuss the procedure for reconstructing the blood vessel trees from the set A of robust maxima on all axis aligned slices.

Points of A tend to be concentrated along the centerlines of blood vessels. There are numerous outliers that do not belong to any vessel, but their distribution tends to be structureless. The procedure described in this section attempts to connect the neighboring points on the same blood vessel trees. We first compute a forest $\mathcal{F}$ connecting the input points with short edges, which serves as a crude reconstruction of the blood vessels. This forest is subsequently cleaned up and improved using three simple geometric filters that:

(a) Trim short branches of the trees in the forest.

(b) Fill 'gaps' in the vessels by finding branches that seem to geometrically fit to each other and connecting them with edges.

(c) Clean up the tree by removing edges that cause sharp turns in the blood flow direction predicted from our reconstruction.

The whole process is illustrated in Figure 4. Below we describe each step in more detail.

3.1. Building the forest

To obtain a crude reconstruction, we use the Kruskal's minimum spanning tree algorithm[3] to build a minimum forest that uses edges connecting the points in A shorter than a certain threshold. We sort the edges connecting the points of A whose length is smaller than the threshold according to length. Then, we attempt to insert them to the forest one at a time in order of increasing length. As we do that, we keep track of the connected components of the forest using the union-find data structure. Edges whose endpoints belong to the same connected component are not inserted into the tree (since this would cause a loop in the graph).

In order to speed up this procedure we use only edges of the Delaunay tetrahedralization of the input points rather than all possible short edges connecting them to build the forest. While the worst-case bound on the number of such edges is quadratic, it tends to be linear in practice and efficient Delaunay tetrahedralization algorithms with free robust implementations exist.[4]

Let us note that the idea of using Euclidean minimum spanning tree for geometric reconstruction from points is not new. The work[5] contains an in-depth analysis of this approach to curve reconstruction in the planar case.

3.2. Forest trimming

The points in A are often somewhat off the axis of the blood vessels or are outliers defined by intensity maxima at random places. This may lead to short branches of the trees in $\mathcal{F}$. The purpose of forest simplification is to remove such short branches. We do that by iteratively removing the shortest branch in the forest until the length of all branches is above a certain threshold.

For the purpose of this procedure, we treat leaf vertices of the forest $\mathcal{F}$ as branch endpoints. The branch defined by a leaf vertex v is defined as the simple path, i.e. path that uses no vertex more than once, in the forest starting at v and ending at a vertex of degree other than 2. Note that such a path is unique. To efficiently keep track of short branches due to be removed we use a priority queue. At startup, all leaf nodes of the forest are inserted into the queue. The priority of a leaf node is equal to minus the length its branch.

The main loop takes a leaf vertex off the priority queue and removes all edges in its branch. Such a removal may potentially change the length of some leaf vertices' branches (Figure 3). This happens whenever the last vertex w of the branch has degree 2 after the removal. In such cases, we trace the two paths starting at w until they reach a vertex of degree other than 2. Let the endpoints of these paths be w_1 and w_2 and their lengths be l_1 and l_2. If the degree of w_1 or w_2 is one, its priority should be updated to minus $l_1 + l_2$. We implement this by simply inserting the endpoints with new priorities into the queue. This means an endpoint can appear in the priority queue more than once, and that some of these instances may have outdated priorities. However, the priorities of branches can only decrease in time (as their length can only increase) and therefore we can detect whether a leaf taken off the priority queue is valid or not by checking how many times it appears on the queue: if it appears just once before it is taken off, it means that its priority is valid. In order to be able to do that efficiently, for each vertex we maintain the number of times it appears on the queue.

The above procedure terminates when the length of the branch taken off the priority queue becomes larger than the threshold.

3.3. Gap filling

In some cases, the points in A are not dense enough along a blood vessel for the tree growing algorithm to obtain its continuous reconstruction. This is typically caused by variations of contrast along the blood vessels: in the low contrast parts the persistence of the local maxima on 2D slices drops below the threshold and, consequently, they are not included in A. We attempt to fix this problem by adding edges connecting leaf vertices in the forest that appear to belong to the same blood vessel based on certain geometric criteria.

The heuristic we use to decide whether two leaf nodes u and v should be connected is based on the following assumptions:

(a) Since gaps tend to be small, u and v should not be too far from each other

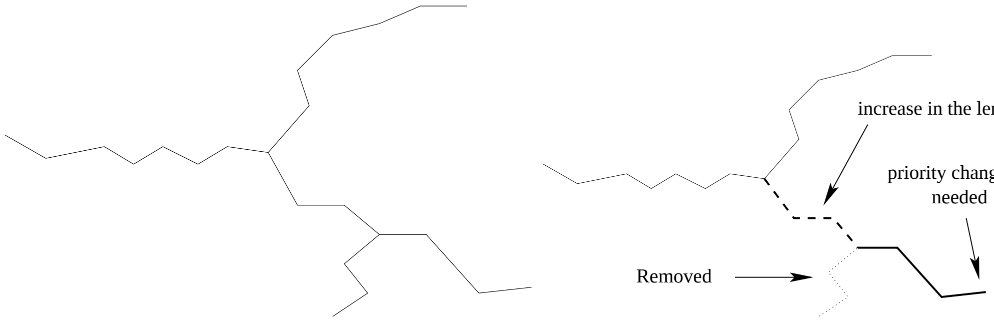

Figure 3. An example showing when a priority needs to be updated after a removal of a branch. The shortest branch of the tree on the left (dotted line in the figure on the right) is removed. The priority of the endpoint of the branch indicated by the thick line needs to be changed, since its length increases by the length of the path shown as the dashed line.

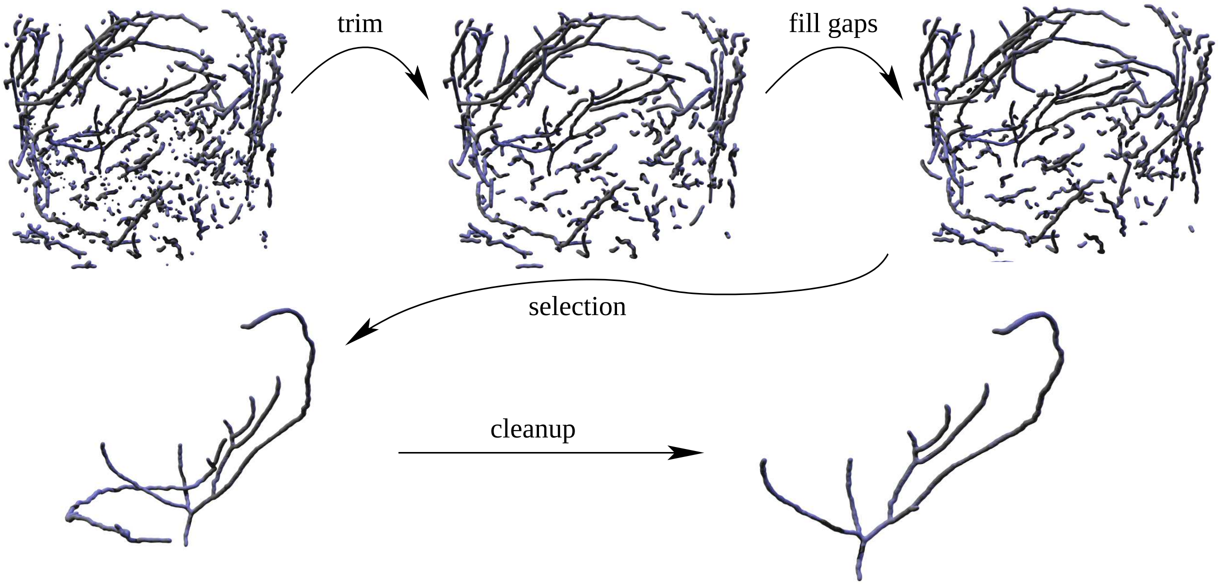

Figure 4. Consecutive stages of processing the forest computed from the robust maxima.

(b) The newly inserted edge should blend well with the tree branches starting at u and v. This means that the outward-pointing tangent vector to the vessel at u should point in a direction close to the vector $\vec{uv}$ and that the outward-pointing tangent vector at v should point in a direction close to $\vec{vu}$.

This naturally leads to the following penalty function defined for all pairs of leaf vertices:

$$f(u,v) = \alpha|u-v| + |\frac{T_u}{|T_u|} - \frac{\vec{uv}}{|uv|}| + |\frac{T_v}{|T_v|} - \frac{\vec{vu}}{|vu|}|.$$

where T_u and T_v are tangent vectors at u and v and α is a scaling constant, determined experimentally. To estimate the tangent vector T_u at u, we follow the path in the graph starting from u. The path is terminated as it reaches a prescribed length or it hits a junction (vertex of degree greater than 2), whichever happens first. We approximate T_u as the vector from the endpoint of the path to u. The tangent at v, T_v, is estimated in the same way.

Our algorithm greedily connects pairs of leaf vertices in order of increasing penalty, stopping when the penalty exceeds a certain threshold. We reject edges that would cause loops to appear in the graph. This can be implemented efficiently by utilizing the union-find data structure in the same way as in the Kruskal's minimum spanning tree algorithm.

3.4. Tree selection

So far we were operating on the entire forest, that typically includes all blood vessel trees reconstructed by our algorithm. Now, the user needs to select a blood vessel tree of interest from this forest. The selection is made by clicking on a voxel near the root of the tree of interest. We find its nearest node in the forest and output its connected component. As illustrated in Figure 4, this tree may potentially contain spurious branches that need to be cleaned up. Note that the root node selected by the user should define the point from which the blood flows into the tree in order for the tree cleanup step (described in the next section) to work properly.

3.5. Tree cleanup

We use an automatic procedure that follows paths toward the user-selected root of the tree, detecting sharp turns. The edges leading into sharp curves are removed to produce a cleaner variant of the tree. Clearly, one could leave an option of manually selecting certain edges of the tree for removal, possibly using the sharp turn detection procedure as a source of hints provided to the user.

The cleanup procedure first directs the edges of the tree so that they point toward the root (i.e. so that by following the directed edges from any vertex one can obtain a simple path leading to the root). The directions can be thought as opposite to the blood flow direction (computed assuming that the tree perfectly reconstructs the blood vessel tree) and hence should not abruptly change. Building upon this observation, we detect and remove edges that lead into sharp turns. Because of the noisy and discrete nature of the tree generated by our algorithm, a vector running along an edge cannot be treated as a reliable estimate of the blood flow direction. Therefore, as the estimate of the direction opposite to the blood flow at v we use the vector connecting a vertex v of the tree with the vertex a certain distance along the path toward the root starting at v. We also need to make sure that the branching points are treated correctly, i.e. that the 'good' branches are not cut away from the root together with the 'bad' ones. These considerations led us to the following algorithm.

For each vertex v of the directed tree, we follow its path toward the root until it gets longer than some minimum length L. Let the endpoint of this path be $F(v)$. Denote the vector from v to $F(v)$ by $D(v)$ (this is our estimate of the direction opposite to the blood flow). We first mark all vertices of the tree for which $D(v)$ and $D(F(v))$ are defined and form angle greater than a certain threshold. Then, for each marked vertex we trace its path until it reaches length L or reaches a vertex of degree other than 2 (whichever happens first) and remove its last edge. Finally, we compute the connected component of the resulting forest that contains the root vertex. This is our final reconstruction of the blood vessel tree.

4. ADAPTIVE THRESHOLDING

We now describe a method for converting the trees into volumetric models for the purposes of visualization. There are have been several other techniques for doing this, e.g., codimension two active contours[6] as well as shape-based Bayesian methods.[7] Here we employ simple adaptive thresholding technique, based on two requirements:

- **(a)** Threshold should be roughly proportional to the intensity at the nearest voxel intersecting the tree
- **(b)** The output volume must be confined to a ball of a small (but slightly larger than the expected maximum width of the blood vessel to be reconstructed) radius R, centered at the tree.

We start off by approximating the discrete closest point transform for the tree. We scan-convert the tree onto a regular grid of the same resolution as the input volume. Let T be the set of voxels marked as belonging to the tree during scan conversion. For each voxel v in T, we scan-convert a ball of radius $R+1$ centered at v, updating the distance and the information about the coordinates of the closest voxel in T for each encountered voxel. Let B be the set of all voxels for which the distance information is available (i.e. those for which there exists a voxel in T less than $R+1$ away). We then color all voxels in B in order of decreasing priority. The

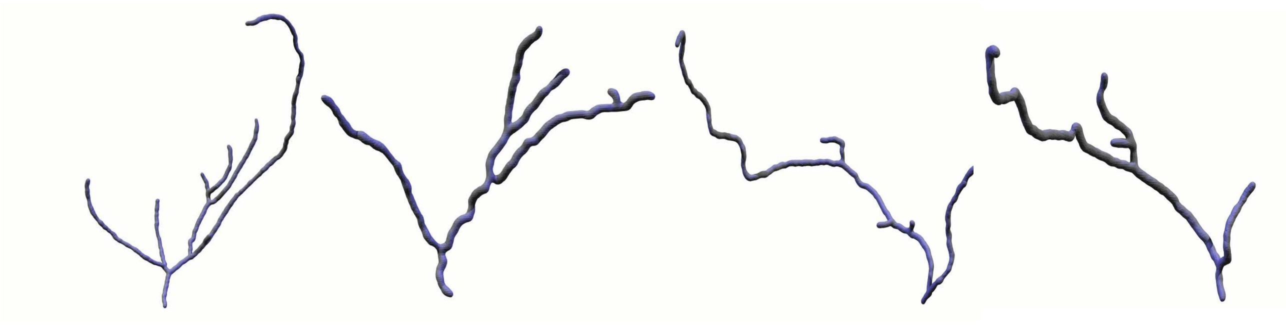

Figure 5. Examples of coronary trees obtained using our method.

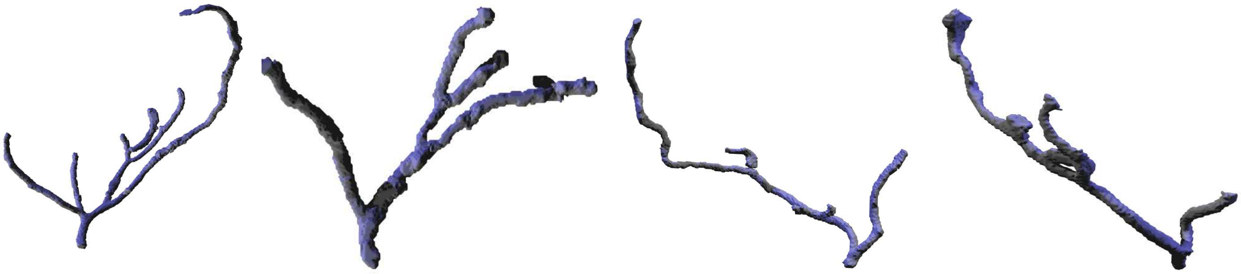

Figure 6. Results of adaptive thresholding around the coronary trees shown in Figure 5.

priority of a voxel is defined as the ratio of its intensity and the intensity of the closest voxel in T. We keep track of the connected components of the set of colored voxels using the union-find datastructure. With each of the connected components, we keep two binary flags, one indicating whether the component contains a voxel in T and one indicating whether it contains a voxel with distance value greater than R. Before a voxel is colored, we make sure that the resulting connected component does not have both flags set simultaneously (if it does, we do not color the voxel), enforcing the requirement **(b)**. We also stop coloring voxels when the priority drops below a certain threshold. The output volume consists of all connected components of colored voxels that intersect the tree (in most practical cases, this is going to be just one component).

For larger datasets or larger values of R, it may be beneficial to use fast marching algorithm[8] or other closest point transform algorithms[9] to increase the efficiency. We also plan to use geodesic active contours[10, 11] for extracting the vessels by using the results of the algorithm described above as the initial segmentation.

5. RESULTS

Our algorithm has been tested on several heart CT scans. In most cases, we have been able to obtain a correct reconstruction of the coronary tree. The sizes of the test datasets were between about 10 and 15 million voxels. The running times of our implementation (which is currently far from optimal) were about 6 minutes for each of the datasets on a Pentium III-850MHz workstation. The coronary resulting coronary trees are shown in Figure 5. Solid models of the coronary trees obtained by adaptive thresholding are shown in Figure 6.

5.1. User-defined parameters

Our algorithm requires fixing several parameters: the robustness threshold for local maxima, the maximum edge length for use in the forest growing phase, minimum length of a branch to be used when trimming, the α parameter and the maximum allowable penalty function value for gap filling purposes, the maximum allowed turn angle for the cleanup stage and the minimum allowed priority for adaptive thresholding. We found out that, although the number of these parameters is high, they are relatively easy to set and, once they are set, they tend to work well for all datasets coming from the same source. In particular, all example outputs shown in this section use the same parameters which we chose after a about half an hour of experiments on one of the datasets. Perturbing the parameters induces easily predictable changes to the output tree: lower maximum penalty for gap filling might make some branches shorter. Higher maximum turn angle for cleanup stage might cause some of the spurious branches to survive. The robustness threshold should be set so that enough maxima are found near the centers or important blood vessels and few of them are found in other places. In most cases, the effect of suboptimal choice of the parameters seems to be easy to fix with a small amount of user interaction. We believe that, ultimately, the automatic procedure discussed in this paper should be augmented with a user interface allowing to manually edit the trees for optimal results.

6. SUMMARY AND FUTURE WORK

We presented a new, simple and effective method for segmenting blood vessels from 3D images motivated by topological techniques. We have successfully applied our algorithm to segment coronary arteries from heart CT scans. The method presented here can be improved in numerous ways as well as applied to other problems where detecting thin tubular structures in noisy volume datasets is of importance.

We are investigating ways to unify the forest building, trimming, gap filling and cleanup stages into one tree construction procedure using a more sophisticated priority for the edges that are to be inserted into the tree. We believe a good way of selecting such edges should be based on estimating the vessel properties from the partially constructed forest. Such estimates should play increasingly important role as the whole process progresses and longer and longer edges are considered for inserting into the tree. Further interesting insights for this stage may come from work on reconstruction of curves from unorganized points.[12–14]

We also hope that the approach taken in this paper, i.e. generating point set describing an object of interest and using reconstruction algorithm that take unorganized points as the input, will prove useful in other segmentation problems, including those where the object to be segmented does not have one-dimensional structure.

Finally, we would like to investigate how the trees obtained here could help improve results of deformable surface-based algorithms, for example by providing ways to initialize them properly or prevent leakage.

Acknowledgements

We would like to thank Dr. Arthur Stillman of the Cleveland Clinic for providing us with the data sets used in this study. This research was supported by grants from the NIH (NAMIC and NAC) through Brigham and Women's Hospital in Boston and NSF grants DMS-0138420 and DMS-0107396.

REFERENCES

1. C. Kirbas and F. Quek, "A review of vessel extraction techniques and algorithms," tech. rep., VisLab Wright State University, Dayton, Ohio, November 2000.
2. H. Edelsbrunner, H. Harer, and A. Zomorodian, "Hierarchical morse-smale complexes for piecewise linear 2-manifolds," in *Symp. on Computational Geometry*, pp. 70–79, ACM, ACM Press, New York, NY, USA, 2001.
3. T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, McGraw-Hill, 1990.
4. C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Transactions on Mathematical Software* **22**(4), pp. 469–483, 1996.
5. L. H. de Figueiredo and J. Gomes, "Computational morphology of curves," *The Visual Computer* **11**(2), pp. 105–112, 1995.
6. L. Lorigo, O. Faugeras, W. Grimson, R. Keriven, R. Kikinis, A. Nabavi, and C.-F. Westin, "Codimension-two geodesic active contours," in *Proceedings of CVPR, Hilton Head, SC.*, June 2000.
7. Y. Yang, A. Tannenbaum, and D. Giddens, "Knowledge-based 3d segmentation and reconstruction of left coronary arteries using ct images," in *Proceedings EMBS04*, 2004.

8. J.A.Sethian, *Level Set Methods and Fast Marching Methods*, Cambridge University Press, Cambridge, UK, 1999.
9. S. Mauch, *Efficient Algorithms for Solving Static Hamilton-Jacobi Equations.* PhD thesis, California Institute of Technology, April 2003.
10. V. Caselles, R. Kimmel, and G. Sapiro, "Geodesic snakes," *Int. J. Computer Vision* **22**, pp. 61–79, 1997.
11. S. Kichenassamy, A. Kumar, P. Olver, A. Tannenbaum, and A. Yezzi, "Conformal curvature flows: from phase transitions to active vision," *Archive for Rational Mechanics and Analysis* **134**, pp. 275–301, 1996.
12. T. K. Dey and P. Kumar, "A simple provable algorithm for curve reconstruction," in *Proc. 10th. ACM-SIAM Symposium on Discrete Algorithms (SODA '99)*, pp. 893–894, 1999.
13. T. K. Dey, K. Mehlhorn, and E. Ramos, "Curve reconstruction: connecting dots with good reason," *Computational Geometry: Theory and Applications* **15**(4), pp. 229–244, 2000.
14. N. Amenta and M. W. Bern, "Surface reconstruction by voronoi filtering," in *Symposium on Computational Geometry*, pp. 39–48, 1998.