TEASAR: Tree-structure Extraction Algorithm for Accurate and Robust Skeletons

Mie Sato[†⋆] Ingmar Bitter[†] Michael A. Bender[†] Arie E. Kaufman[†] Masayuki Nakajima[⋆]

[†]Department of Computer Science
SUNY at Stony Brook
{mie, ingmar, bender, ari}@cs.sunysb.edu

[⋆]Department of Computer Science
Tokyo Institute of Technology
{mie, nakajima}@img.cs.titech.ac.jp

Abstract

We introduce the TEASAR *algorithm which is a* Tree-structure Extraction Algorithm delivering Skeletons that are Accurate and Robust. *Volumetric skeletons are needed for accurate measurements of length along branching and winding structures. Skeletons are also required in automatic virtual navigation, such as traveling through human organs (e.g., the colon) to control movement and orientation of the virtual camera. We introduce a concise but general definition of a skeleton, and provide an algorithm that finds the skeleton accurately and rapidly. Our solution is fully automatic, which frees the user from having to engage in data preprocessing. We present the accurate skeletons computed on a number of test datasets. The algorithm is efficient as demonstrated by the running times on a single 194 MHz MIPS R10000 CPU which were all below five minutes.*

1. Introduction

The essential geometry of complicated 3D objects is well understood and manipulated by reducing the shapes to their 1D skeletons. For example, automatic virtual navigation through a human colon [7] uses the colon skeleton, its centerline, to control the movement and orientation of the virtual camera. Similarly, accurate length measurements and navigation through other human organs, such as the aorta, require skeleton computations. In addition, in the fields of virtual engineering and architectural design, the problem of finding an optimal path through hollow structures, where this path should have minimum collision probability [6], poses the same skeleton-finding problem.

In this paper we find skeletons in binary discretized 3D occupancy maps of tree-like structures. We use segmented medical CT and MRI scans as our input data. However, our techniques are general and may be readily applied to other domains, because our assumptions are not specific to the source of the data.

2. Overview of skeleton algorithms

The intuitive notion of a skeleton of a 3D object without holes is the central tree spanning that object. It is challenging to construct a formal mathematical definition of a skeleton. There has been extensive work on this topic. We summarize here traditional skeleton algorithms along with their concepts of what a skeleton should be.

All compared skeleton algorithms assume that the data is presented as a 3D rectilinear grid called a *volume* [8] of volumetric sample points called *voxels* [8]. Two voxels are *6-connected* if at most one of their 3D coordinates differs by 1, *18-connected* if at most two coordinates differ by 1, and *26-connected* if all three coordinates are allowed to differ. A *6/18/26-connected path* through this data is a sequence of 6/18/26-connected voxels. A *discrete skeleton* is a tree composed of such paths.

This paper extends our CEASAR algorithm [1] for finding centerlines. In contrast to a skeleton, the centerline is a central simple path that spans the object. Our new approach is a substantial improvement that is focused on more general tree-structure skeletons. The TEASAR algorithm applies CEASAR recursively; hence, in this paper we also explain those steps of CEASAR that are relevant to TEASAR.

2.1. DSF and Dijkstra shortest path

Many algorithms that force the skeleton to be a simple path use the Dijkstra shortest path graph algorithm [3] as an intermediate step. The Dijkstra algorithm provably finds the global minimal weight path in a (directed or undirected) weighted graph with non-negative weights. The algorithm has two phases. The first phase creates a **d**istance from **s**ource **f**ield (DSF) by labeling all graph vertices with the shortest distance from a single source to those vertices. The second phase creates the shortest path by tracing back to the

Proceedings of the Eighth Pacific Conference on Computer Graphics and Applications (PG'00)
0-7695-0868-5/00 $10.00 © 2000 IEEE

source node. Note that this back trace is not the same as the steepest descent in the DSF. In order to apply the Dijkstra algorithm as a sub-step of our skeleton algorithm, the volume data has to be transformed into a graph. We implicitly map the voxels to graph vertices, and the voxel neighbor relations to graph edges (for more details see Section 3 and Figure 1a).

The centerline algorithms using steepest descent or Dijkstra's method differ in how they assign the weights corresponding to orthogonal, 2D-diagonal, and 3D-diagonal vertex neighbor relations. The algorithms employ the 1-0-0 Manhattan metric [12], the 1-2-3 metric [13], the 3-4-5 Chamfer metric [4], or the 10-14-17 metric [2]. These metrics are sorted by decreasing error when compared with the Euclidian distance between the voxel positions. It is most accurate to use a 1-$\sqrt{2}$-$\sqrt{3}$ Euclidian metric for isotropic volumes and a metric with axis specific corrections for anisotropic volumes. This is the approach adopted in this paper.

Independent of the choice of the metric, the resulting shortest path visits vertices of the graph, and is therefore guaranteed to reside inside the segmented shape. Unfortunately, this path tends to cut the corners and travel along boundary voxels on the inside of sharp turns. Hence, this path generally does not qualify as centered.

A method for reducing this cutting of corners is to replace the Dijkstra back-trace path with a path along the centers of cluster masses with similar DSF values [12]. This technique would work well if the "wave fronts" formed by a cluster of voxels of the same DSF value were always perpendicular to the skeleton. Unfortunately, near sharp turns, the wave fronts tilt and can even be parallel to the intuitive skeleton.

2.2. Distance from boundary field

A slight modification to the first phase of Dijkstra's algorithm is to replace the single source voxel with the set of all boundary voxels. The result is a distance field that stores for each voxel the length of its shortest discrete path to the boundary. Again, a variety of distance metrics for edge weight assignments is possible.

This **d**istance from **b**oundary **f**ield (DBF) can be used to improve the centrality of the skeleton by relocating skeleton point candidates to the maximal point of the DBF within the plane perpendicular to the skeleton [2]. However, a single correcting step does not yield an optimal skeleton, and even iterating this method is not guaranteed to find a global optimum.

A better approach is to relocate skeleton point candidates to the maximal DBF voxel within the "wave front" of the same DSF values [13]. However, this disconnects the candidate skeleton, and stitching it back together is based on local heuristics.

Note, that the set of maximal DBF voxels may form a 2D manifold, which is not desired for a skeleton spanning an object. For example, an object in the shape of a shoe box sandwiched between two large spheres would have such a 2D voxel set in the middle of the shoe box. This falsely suggests that the shoe box is larger than the spheres.

2.3. Topological thinning

The technique that is traditionally considered to provide high quality results is called topological thinning or "onion peeling" [4, 5, 7, 9, 10]. In this general strategy, one layer of voxels at a time is peeled off the object until just the skeleton remains. Multiple invariants should be maintained to avoid errors. The branch end voxels cannot be removed and must remain part of the same connected component, and the topology must be preserved. No voxel can be removed that would cause these constraints to be violated.

Unfortunately, onion peeling is computationally expensive. Additionally, there is no concise mathematical formulation of what the onion-peeled skeleton should look like.

3. Formal skeleton definition

In this paper, we introduce a concise but general definition of a skeleton of an object without holes. We then present an algorithm that can accurately and rapidly produce a tree-shaped skeleton based on this definition. We provide a fully automatic solution, which frees the user from having to participate in the data preprocessing. Our skeleton algorithm is designed to be *provably robust*. It is guaranteed to perform correctly even for winding, twisted structures.

We now describe some basic properties a skeleton should have. Most importantly, the skeleton should be a tree composed of simple voxel paths without any 2D manifolds.

The skeleton should never leave the inside of the segmented shape. More specifically, the skeleton should tend to remain in the "center" of the shape. For winding and bulging shapes, the concept of center may not be well defined. Intuitively, the skeleton should be situated as far from the boundary as possible. On the other hand, it should also avoid too much winding because the skeleton should be as short as possible within all other constraints. This suggests that our algorithm should find some kind of shortest path through the object, or a union of shortest paths.

As pointed out in Section 2.1, the Dijkstra shortest path algorithm requires volume data to be mapped to graph vertices and graph edges. Figure 1a depicts a straightforward implicit mapping. Edges represent the 26-neighbor relations between voxels. As weights, we use the exact Euclidian distances between the voxels that correspond to the

Proceedings of the Eighth Pacific Conference on Computer Graphics and Applications (PG'00)
0-7695-0868-5/00 $10.00 © 2000 IEEE

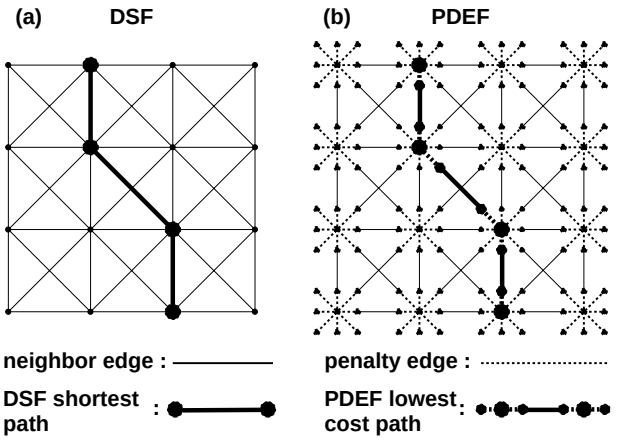


Figure 1. 2D top view of the implicit mapping of the voxel grid and neighbor relations to an undirected graph on which the Dijkstra algorithm can be applied. (a) A region of a "plain" distance field with part of the shortest path. (b) The same region, but now with penalties. Solid neighbor edges have weights equal to the distance between the voxels. Dashed penalty edges have a weight equal to half the penalty assigned to the associated voxel.

graph vertices at both ends of the edge. However, even when including corrections for anisotropic volumes, an unembellished shortest path through the object has the defect that when it turns, it cuts the corners, instead of staying near the center. Therefore, we enhance the implicit graph by adding more edges and vertices as depicted in Figure 1b to incorporate penalties for coming close to the object boundary and to create a **p**enalized **d**istance from **r**oot **f**ield (PDRF). There are now 27 vertices per voxel: one center vertex and 26 penalty vertices that each share a penalty edge with the center vertex. The penalty edges have a weight equal to half the penalty associated with including that voxel into the path. Neighbor edges now always connect to penalty vertices. Since this modification results in a graph that is a singly connected component with positive edge weights, it is guaranteed that the Dijkstra algorithm finds the globally minimal shortest path. The cost along that shortest path is the accumulated piecewise Euclidian distance of the path plus the sum of the penalties of all penalty edges visited along the path.

We define the skeleton to be the tree of minimum cost paths found in the penalized distance field. This definition has the following concrete advantages: it is precise, rapidly computable, and suggests a provably correct algorithm. Note it should not contain any 2D manifolds such as those possibly included in a set of maximal DBF voxels. Naturally, there is a range of penalties that may be applied to the penalty edges defining a family of continuously varying skeletons. In Section 4.5 we suggest a choice of penalty function that yields a tree of high-quality, centered branches.

4. TEASAR algorithm

The TEASAR algorithm works for any tree-shaped structure. Examples of such structures are pipes, tunnels, blood vessels, lungs and ribs. In fact, the algorithm is so robust that it even handles arbitrary connected shapes (that may have holes), although it always produces a tree-shaped skeleton. The algorithm consists of nine logical steps :

1. Read binary segmented voxels inside the object
2. Crop volume to only the object
3. DBF: Compute the **d**istance from **b**oundary **f**ield
4. DAF: Compute the **d**istance from **a**ny voxel **f**ield
5. PDRF: Compute the **p**enalized **d**istance from **r**oot voxel **f**ield
6. Find the farthest PDRF voxel labeled as "inside"
7. Extract the minimum cost path from that voxel to the root
8. Label all voxels near the path as "used to be inside"
9. Repeat the last three steps until no inside voxels remain

We now describe these steps in detail.

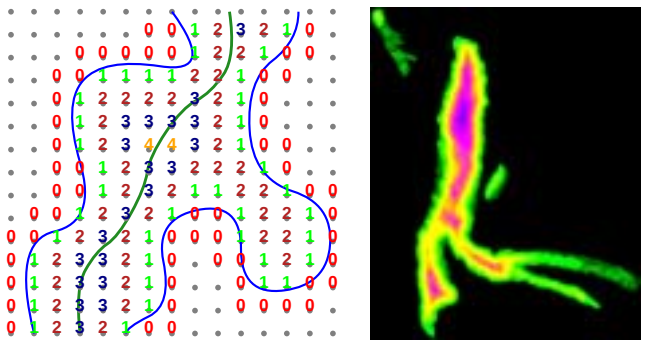
4.1. Binary segmented object

The input for our TEASAR algorithm is a binary mask, which labels the voxels belonging to the shape's interior and boundary as "inside." We merely require the structure to be a single connected component.

4.2. Crop volume to just the object

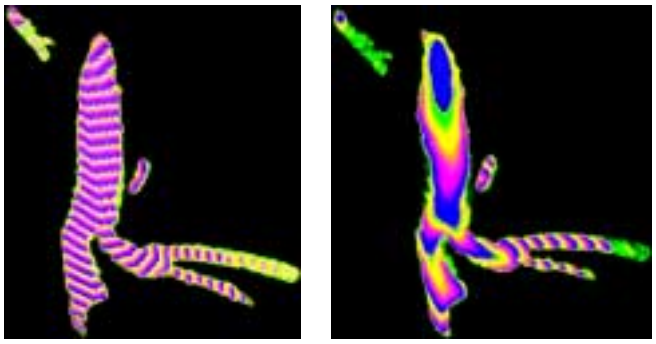
One advantage of TEASAR is that it is computationally efficient. Thus, in each step we strive to minimize the number of voxels that have to be processed. The first step in reducing the number of relevant voxels is to crop the volume automatically to enclose just the bounding box of the voxels labeled as "inside." For medical scans, this typically reduces the volume size by 30%-50%.

Proceedings of the Eighth Pacific Conference on Computer Graphics and Applications (PG'00)
0-7695-0868-5/00 $10.00 © 2000 IEEE



(a) (b)

Figure 2. (a) Explicit DBF values (rounded to integers). (b) Aorta DBF visualized through a rainbow color map.



(a) (b)

Figure 3. (a) Aorta DAF visualized through color rainbow mapping showing "same distance wave fronts." (b) Aorta PDRF visualized with the same rainbow mapping showing "same cost wave fronts" progressing most rapidly in the center of the aorta.

4.3. DBF: Compute distance from boundary field

The Euclidian distance between an inside voxel and the object boundary is recorded at each voxel. This forms the **d**istance from **b**oundary **f**ield (DBF) as shown in Figure 2. We use a four-pass algorithm by Saito and Toriwaki [11], whose running time is linear in the number of voxels, to compute the real Euclidian DBF accurately.

4.4. DAF: Compute distance from any voxel field

The following TEASAR steps are an adaptation of the Dijkstra shortest path graph algorithm as outlined in Sections 2.1 and 3. This first step computes the **d**istance from **a**ny inside voxel **f**ield (DAF) with the anisotropically correct Euclidian distance as the weights in the implicit graph of Figure 1a. Independently of the choice of an inside voxel to be used as the starting point, the farthest voxel is one of the extremal points of the shape and thus can be used as the root for our skeleton tree. Figure 3a shows the resulting DAF.

4.5. PDRF: Compute penalized distance from root voxel field

Repeating the search for the farthest voxel from the root voxel found in the previous step, we discover the other extremal voxels in the object. However, during this second search we extend the volume-data-to-graph mapping to incorporate penalties for coming close to the object boundary as illustrated in Section 3 and depicted in Figure 3b, and thus create the **p**enalized **d**istance from **r**oot voxel **f**ield (PDRF).

The penalty p at a voxel v is assigned based on the DBF value at that voxel v and a global upper bound of all DBF values ($M > max(DBF)$). The penalty function $p(v)$ is:

$$p(v) = 5000 \cdot [1 - \frac{DBF(v)}{M}]^{16}.$$

Note that $\frac{DBF(v)}{M}$ is always in the range of [0,1]. Thus $[1 - \frac{DBF(v)}{M}]^{16}$ is in the same range, but with the maximal values for voxels close to the boundary. The factor 5000 is needed to ensure that the penalty overpowers the advantages of choosing a straight path. Choosing 5000 is a heuristic that allows skeleton segments to be up to 3000 voxels long without exceeding floating point precision.

For our implementation, we did not need to store explicitly all 26 penalty vertices and edges depicted in Figure 1b because the only way to incorporate a center vertex in the path is to travel through two of its penalty vertices, and thus along the two penalty edges of equal penalty weight. Therefore, we can actually keep the implicit edges and vertices from the DAF generation method, but add the penalty directly to the computation of the accumulated distance d at each voxel v; thus, the distance function $d(v)$ is:

$$d(v_k) = d(v_{k-1}) + d(v_k, v_{k-1}) + p(v_k).$$

4.6. Find farthest voxel

Find the voxel that is labeled as "inside" and has the largest PDRF value and assign it to be the starting voxel.

4.7. Minimum cost path

We run the second phase of the Dijkstra algorithm rooted at the starting voxel chosen in the previous step. Because of our inclusion of strong penalties into the PDRF, the algorithm finds a global minimum path between the extreme point voxel and the root voxel that is optimally centered, and also follows maximal values of the DBF. This path becomes a branch of our discrete skeleton tree as defined in Section 3 and depicted in Figure 4a.

4.8. Label voxels near the skeleton

After extraction of the minimum cost path we "roll an adaptive sphere down that path." We say that the sphere is adaptive because the radius r is computed for each voxel v on the path to be

$$r(v) = DBF(v) \cdot scale + const.$$

Specifically, we label all voxels that are inside voxels and within the radius $r(v)$ to become "used to be inside" voxels. The combination of $scale$ and $const$ determines the minimum feature size that has its own skeleton tree branch. For a centerline of a human colon we can choose $scale = 3$ and $const = 50$ which result in the first path already labeling all colon voxels and thus producing only a single centerline. For objects such as the aorta, the values of $scale = 1.1$ and $const = 10$ result in finding all blood vessel branches.

4.9. Repeat farthest voxel, minimum cost path and labeling

We repeat the last three steps until the labeling procedure converts all inside voxels into "used to be inside" voxels. For each new branch, the minimum cost path extraction is performed on the never changing PDRF that was computed once in Step 5. Finding the minimum cost path continues even if "used to be inside" voxels are reached . The path terminates only if a voxel is reached that is already part of the skeleton. Repeatedly using the same PDRF guarantees that all branches are connected, because they ultimately terminate at the single global root voxel of the strong monotone deceasing PDRF. The union of all minimum cost paths is the desired TEASAR skeleton.

5. Results

We tested our TEASAR algorithm on CT scans of a lobster, a human colon, a rib cage, and an aorta dataset. Table 1 lists the details about the dataset sizes and the number of processed voxels. In all cases, the discrete skeletons were placed right in the center according to visual inspection and according to mathematical measures such as the DBF.

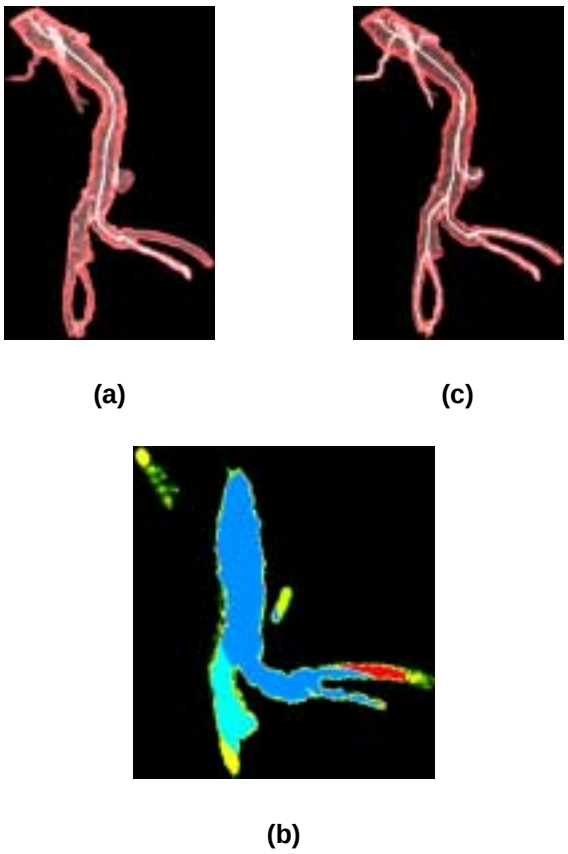


Figure 4. (a) Aorta with the minimum cost path after one iteration. (b) Aorta cut open to show regions labeled during multiple iterations. (c) Complete aorta skeleton.

Table 2 lists the platform and the timings of all the TEASAR algorithm steps for each test dataset. All total running times were below 5 minutes and just 37 seconds for the lobster. Figure 5 depicts the final skeletons computed with their associated volumes.

6. Conclusions

We introduced TEASAR — our tree-structure extraction algorithm delivering skeletons that are accurate and robust. TEASAR is based on a new, robust skeleton definition. It computes the skeleton automatically for any single connected component object. The skeleton branches are centrally located within their associated object regions. We explained our TEASAR implementation in detail and reported results that not only empirically verified the correctness of the skeleton, but also showed the superior speed of the TEASAR algorithm; that is, less than 5 minutes for all our test datasets.

Proceedings of the Eighth Pacific Conference on Computer Graphics and Applications (PG'00)
0-7695-0868-5/00 $10.00 © 2000 IEEE

Lobster

Colon and its skeleton

Lobster skeleton

Aorta and its skeleton

Lobster and its skeleton

Ribs and their skeleton

Figure 5. Four datasets with skeletons computed by the TEASAR algorithm. (See color plates.)

Proceedings of the Eighth Pacific Conference on Computer Graphics and Applications (PG'00)
0-7695-0868-5/00 $10.00 © 2000 IEEE

Table 1. Dataset sizes and the reduction of voxels that have to be processed during the execution of the TEASAR algorithm.

dataset	Lobster	Colon	Aorta	Ribs
original size X	256	514	256	512
original size Y	254	514	256	512
original size Z	57	363	211	247
cropped size X	242	416	193	351
cropped size Y	241	398	162	248
cropped size Z	37	363	211	247
all data voxels	96M	96M	13M	65M
cropped voxels	59M	59M	6.7M	22M
inside voxels	141K	3.2M	230K	1M
skeleton voxels	1639	1644	850	5251

Table 2. Time spent in each of the TEASAR algorithm steps. (All tests were done on an SGI Challenge with 4GB memory running IRIX 6.5 using a single MIPS R10000 CPU running at 194 MHz.)

dataset	Lobster	Colon	Aorta	Ribs
cropping	1s	16s	15s	7s
DBF	3s	106s	13s	41s
DAF	3s	9s	4s	16s
PDRF	13s	74s	36s	118s
skeleton	17s	3s	9s	96s
total	37s	208s	77s	278s

Acknowledgments

This work has been supported by grants from NIH CA79180, ONR N000149710402, E-Z-EM Inc., Hughes Research Laboratory, and Viatronix Inc. The patients' data sets were provided by the University Hospital of the State University of New York at Stony Brook. We thank Kevin Kreeger for segmenting the aorta and rib data sets, Ming Wan for his helpful discussions, and Marianne Catalano for English corrections. Mie Sato extends special thanks to Yoshida Scholarship Foundation for supporting her study at SUNY at Stony Brook.

References

[1] I. Bitter, M. Sato, M. A. Bender, K. T. McDonnel, A. Kaufman, and M. Wan. CEASAR: A smooth, accurate and robust centerline extraction algorithm. *Visualization 2000*, 2000.

[2] D. Chen, B. Li, Z. Liang, M. Wan, A. Kaufman, and M. Wax. A tree-branch searching, multiresolution approach to skeletonization for virtual endoscopy. In *SPIE's International Symposium on Medical Imaging 2000*, San Diego, CA, Feb 2000.

[3] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerishe Mathemetik*, 1:269–271, 1959.

[4] Y. Ge, D. R. Stelts, and D. J. Vining. 3D skeleton for virtual colonoscopy. *Lecture notes in Computer Science 0302-9743*, pages 449–454, 1996.

[5] Y. Ge, D. R. Stelts, J. Wang, and D. J. Vining. Computing the Centerline of a Colon: A robust and efficient method based on 3D skeletons. *Journal of Computer Assisted Tomography*, 23(5):786–794, 1999.

[6] T. He and A. Kaufman. Collision detection for volumetric models. *IEEE Visualization 97*, pages 27–34, Oct. 1997.

[7] L. Hong, S. Muraki, A. Kaufman, D. Bartz, and T. He. Virtual Voyage: Interactive navigation in the human colon. *Proc. SIGGRAPH '97*, pages 27–34, 1997.

[8] A. Kaufman. *Volume Visualization*. IEEE Computer Society Process, Los Alamitos, CA, 1991.

[9] D. S. Paik, C. F. Beaulieu, R. B. Jeffery, G. D. Rubin, and S. Napel. Automated flight path planning for virtual endoscopy. *Medical Physics*, 25(5):629–637, 1998.

[10] T. Pavlidis. A thinning algorithm for discrete binary images. *Computer Graphics and Image Processing*, 13:142–157, 1980.

[11] T. Saito and J. Toriwaki. New algorithms for Euclidian distance transformation of an n-dimensional digitized picture with applications. *Pattern Recognition*, 27(11):1551–1565, 1994.

[12] Y. Samara, M. Fiebrich, A. Dachman, J. Kuniyoshi, K. Doi, and K. R. Hoffmann. Automated calculation of the centerline of the human colon on CT images. *Acad Radiol*, 6:352–359, 1999.

[13] Y. Zhou and A. W. Toga. Efficient skeletonization of volumetric objects. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):196–209, 1999.