Penalized-Distance Volumetric Skeleton Algorithm

Ingmar Bitter, Arie E. Kaufman, Fellow, IEEE, and Mie Sato

Abstract—This paper introduces a refined general definition of a skeleton that is based on a penalized-distance function and cannot create any of the degenerate cases of the earlier CEASAR and TEASAR algorithms. Additionally, we provide an algorithm that finds the skeleton accurately and rapidly. Our solution is fully automatic, which frees the user from having to engage in manual data preprocessing. We present the accurate skeletons computed on a number of test datasets. The algorithm is very efficient as demonstrated by the running times which were all below seven minutes.

Index Terms—Skeleton, centerline, medial axis, automatic preprocessing, modeling.

1 MOTIVATION

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 (Fig. 2) uses the colon skeleton, its centerline, to control the movement and orientation of the virtual camera [7]. Similarly, accurate length measurements and navigation through other human organs, such as the aorta (Fig. 3), require skeleton computations. Additionally, 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 problems.

In this paper, we describe how to compute skeletons in binary discretized 3D occupancy maps of tree-like structures. We use segmented medical CT and MRI scans as our input data (Figs. 1, 2, 3). However, our techniques are general and may be readily applied to other domains (Fig. 4) 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, 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 [9]. Of volumetric sample points, called voxels [9]. 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 the grid is a sequence of 6/18/26-connected voxels. A discrete centerline is such a path, whereas a discrete skeleton is a tree composed of such paths. Continuous centerlines and continuous skeletons are the corresponding continuous curves in 3D space that are not restricted to only grid points.

There are many methods for creating a continuous centerline from a discrete centerline by using interpolating and approximating curves [2], [4], [5], [8], [11]. All these methods employ regularly spaced discrete voxel positions as the control points. Consequently, a large number of control points are required to ensure the high accuracy needed to keep the centerline inside the object in narrow regions. However, in wide object sections, larger differences between the continuous and discrete centerlines are acceptable. This arrangement is even preferred if it uses fewer control points and thus generates a smoother path and requires fewer computational resources.

This paper improves our previous work on finding centerlines [1] (CEASAR) and tree-structured skeletons [17] (TEASAR). Our refined combined approach, by design of the definition, does not create CEASAR’s “long thing spike” problem (see Section 2.6 and Fig. 9) nor TEASAR’s “late Y-fork merge” problem (see Section 2.6 and Fig. 10).

2.1 Radiologist Markings

In our environment, the object in question was often a human colon (Fig. 2). The most basic recipe for finding the centerline of a colon is to defer to the expertise of a radiologist [4]. In this least automated method, the radiologist is provided with a sequence of 2D cross sections of the colon. On each cross section, the radiologist manually marks the “center” of the cross section. Then, all of the centers of cross sections are connected to form a path, which is then defined to be the centerline.

Unfortunately, this method has severe practical and fundamental drawbacks. Practically, it is expensive to rely
on the radiologist to click on the centers of all cross sections. Fig. 5 explains the more fundamental drawback. Placing markings that are optimal in 2D cross sections is insufficient to create an optimal centerline in 3D. In fact, this method may lead to paths that are noncentered and even penetrate through the colon wall.

2.2 Topological Thinning

The technique traditionally considered to provide high quality results is called topological thinning or “onion peeling” [4], [5], [7], [12], [13]. In this general strategy, one layer at a time of voxels 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. Much research has been devoted to the problem of how to speed up the basic topological thinning algorithm. The highest improvement stems from separating the thinning phase from the connectivity preserving considerations [12]. The main idea here is to first determine a rough candidate centerline, then perform one step of parallel, unrestricted thinning, and, finally, compute the union of the remaining shape and the candidate centerline. This guarantees connectivity and, when iterated, finds a solution that is very close to or possibly identical to the normal onion peeling algorithm. However, there is no concise mathematical formulation of what the onion-peeled centerline should look like. In fact, there are examples, such as the banjo-shaped colon in Fig. 6, in which the onion peeling algorithm does not find the intuitively desired centerline.

2.3 Distance from Source Field and Dijkstra Shortest Path

Many algorithms that restrict the skeleton to 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 nonnegative weights. The algorithm has two phases. The first phase creates a distance from source field 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 source node. Note that this back-trace is not the same as the steepest descent in the distance from source field. In order to apply the Dijkstra algorithm as a substep 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 Fig. 11a).

The centerline algorithms that use 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 [16], the 1-2-3 metric [20], 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 (see Fig. 7). 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 masses of clusters with similar distance from source field values [16]. This technique would work well if the “wave fronts,” formed by clusters of voxels of the same distance from source field value were always perpendicular to the skeleton. Unfortunately, near sharp turns the wave fronts tilt and can even be parallel to the intuitive skeleton.

2.4 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 distance from boundary field can be used to improve the centrality of the skeleton by relocating skeleton points in the plane perpendicular to the skeleton at the maximal point of the distance from boundary field [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 distance from boundary field voxel within the “wave fronts” of same distance from source field values [20]. However, this disconnects the candidate skeleton and stitching it back together is based on local heuristics.

2.5 Centers of Maximally Inscribed Balls

Local maxima, in the distance from boundary field, can be geometrically viewed as centers of maximally inscribed balls. If the center points are moved, the balls must shrink in size to remain inside the colon. Intuitively, all of these points belong on the centerline. However, defining the centerline as the union of these points is insufficient as they typically form a disconnected set. Therefore, Ge et al. [4] extended the class of voxels that cannot be removed during topological thinning to include these centers of maximally inscribed balls. With this extra constraint, onion peeling does determine the intuitively desired centerline for a banjo-shaped object cross section as in Fig. 6, but fails on other shapes. For example, Fig. 8 depicts an object with a flower shaped cross section that causes excessive winding of the centerline. Here, the intuitive centerline would exclude the centers of maximally inscribed balls in the folds.
2.6 Penalized Distance from Source Field

In our CEASAR algorithm [1], we proposed to augment the Dijkstra Shortest Path with a penalty per voxel that was increasing toward the object boundary. In a multipass approach, CEASAR computes the centerline that has the largest total cost of all possible minimum cost paths. For any locally smooth object, this is the intuitively desired centerline. However, in an object with long thin spikes, such as the one in Fig. 9, the CEASAR centerline is drawn into those spikes instead of following along the main body of the object.

In our TEASAR algorithm [17], we extended this approach to also compute branches to yield a complete skeleton. The algorithm can be tuned to include exactly as much detailed structure as desired. Each branch follows the penalized-distance field toward the source voxel until it merges with the already computed skeleton tree. Unfortunately, this merging may occur much later than intuitively desired. Fig. 10a shows a “Y-fork” shaped object on which the two branches join very late. A join, as in Fig. 10b, would be more intuitive.

Our new penalized-distance algorithm handles these special cases of CEASAR and TEASAR properly.

3 Formal Skeleton Definition

We now introduce a concise, but general, definition of a skeleton. We then present an algorithm that can accurately and rapidly produce such a skeleton. 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 tree-shaped and composed of simple voxel paths. Thus, it is a 1D curve in a 3D space, never a 2D manifold. 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 2.3, the Dijkstra shortest path algorithm requires volume data to be mapped to graph vertices and graph edges. Fig. 11a depicts a straightforward implicit mapping. The edges represent the 26-neighbor relations between voxels. As weights, we use the exact Euclidian distances between the voxels that correspond to the graph vertices at both ends of the edge. However, even when including corrections for anisotropic volumes, an unembellished shortest path through the object has a defect that, when it turns, it cuts the corners instead of staying near the center (see Fig. 7). Therefore, we enhance the implicit graph by adding more edges and vertices, as depicted in Fig. 11b, to incorporate penalties for coming close to the object boundary and to create a penalized distance from source field. 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 of including a voxel into the path is divided equally into edge weights of pairs of edges incident with that voxel. Neighbor relation 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 minimum cost path. The cost along that path is the piecewise Euclidian distance of the path plus the sum of the penalties of all penalty edges visited along the path. The length of such a path is the piecewise accumulated Euclidian distance along the same sequence of voxels. The combination of maximum length and minimum cost enables finding the most intuitive centerline. Computing a second penalized distance from source field with all voxels of the prior found centerline as source voxels enables finding the most intuitive skeleton spanning the complete object.

We define the centerline to be the simple path of maximum length among all lowest cost paths found in the penalized-distance from source field. We further define the skeleton to be the tree of maximum length lowest cost paths found in the penalized-distance from all centerline voxels field. This definition has the following concrete advantages: It is precise, rapidly computable, and suggests a provably correct algorithm. It does not require any specific geometry in order to run correctly. Naturally, there is a range of penalties that may be applied to the penalty edges and the range of penalty functions defines a family of continuously varying skeletons. In Section 4.1, we suggest a choice of penalty function that yields a tree of high-quality, centered branches.

4 Penalized-Distance Algorithm

Our 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), but always produces a tree-shaped skeleton. Fig. 12 shows the algorithm in pseudocode.

4.1 Centerline Computation

Steps 1-10 of the penalized-distance algorithm (Fig. 12) compute an object’s centerline. We now describe these steps in detail.

4.1.1 Binary Segmented Object

The input for our algorithm is a binary mask, which labels the voxels belonging to the shape’s interior and wall as “inside.” No restrictions are imposed concerning genus, folding, or touching. We merely require the structure to be a single connected component. Most of our test data are human colon scans that were segmented.

---

1 Read binary segmented voxels inside the object
2 Crop to volume to just the object
3 Compute the distance from boundary field
4 Compute the gradient vector field
5 Flag Non-uniform gradient neighborhoods
6 Connect flagged voxels
7 Compute the distance from source field
   → The farthest voxel is the centerline start/root
8 Compute the penalized distance from root field
   → The farthest voxel is the centerline end
9 Extract the minimum cost path from that voxel to the root
   → This is the centerline
10 Smooth the centerline
   → This centerline is the initial Skeleton
repeat
11 Label voxels near the skeleton
12 Compute a new penalized distance from root field from all skeleton voxels
   → The furthest voxel is the next branch tip
13 Extract the minimum cost path from the tip to any skeleton voxel
   → This path is the next branch
14 Smooth the branch
15 Add the branch to the skeleton
16 until the distance of the next branch tip is below a user defined threshold

---
into a binary mask using an electronic cleansing and segmentation algorithm [10].

4.1.2 Crop Volume to Just the Object
One advantage of the algorithm 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 labelled as inside. For medical scans, this typically reduces the volume size by 30-50 percent.

4.1.3 Compute Distance from Boundary Field
The Euclidean distance between an inside voxel and the object boundary is recorded at each voxel. This forms the distance from boundary field, as shown in Fig. 13. We use an efficient four pass algorithm by Saito and Toriwaki [15] to compute the real Euclidian distance from boundary field accurately.

4.1.4 Compute Gradient Vector Field
For each volume position within the colon mask, we compute its central difference gradient, which requires reading of only six neighboring voxels. This forms the gradient vector field of the distance from boundary field (see Fig. 14). (We also experimented with the smoother 26-voxel neighborhood Sobel filter [18]. However, it is slower and does not affect the final centerline.)

4.1.5 Flag Nonuniform Gradient Neighborhoods
Fig. 15 depicts a section of an object with the centerline and the distance from the boundary gradient vectors. We are interested in labeling the voxels, on and next to the centerline, that often have local neighborhoods of nonuniform gradient vector field vectors. There are six classes of regions in the gradient vector field:

1. Local maxima on the centerline (depicted as green solid squares in Fig. 15). All the gradient vector field

---

Fig. 13. (a) Explicit distance from boundary field values (rounded to integers). (b) Aorta distance from boundary field visualized through a rainbow color map.

Fig. 14. Gradient vector field vectors (a) as arrows and (b) as XYZ = RGB pixel components along a colon cross section.
vectors surrounding such a maximum point toward it. This means each of these vectors have a different direction.

2. **Local minima on the centerline** (depicted as green solid hexagons). All the gradient vector field vectors surrounding such a minimum point away from it toward the neighboring maxima. For each minimum in a tubular shape, there must be at least two enclosing maxima. Thus, the gradient vector field vectors at a minimum split in at least two groups of vectors pointing in opposite directions.

3. **Uniform areas off the centerline** (depicted as black dashed circles). All positions close to the colon boundary have gradient vector field vectors that point in about the same direction as their neighbors because the steepest increase of distance from boundary is always perpendicular to the boundary, and because the boundary itself has only limited curvature.

4. **Uniform areas on the centerline** (depicted as red dotted circles). Unfortunately, there are also areas along the centerline where the colon widens quickly. Hence, all gradient vector field vectors neighboring the centerline point uniformly toward the center of the wide open area. Labeling all nonuniform gradient vector field positions does not capture this area.

5. **Local maxima off the centerline** (depicted as red dotted squares). There may exist regions in the object that enclose local maxima of the distance from boundary field that are not on the centerline. The gradient vector field vectors surrounding such a maximum have the same characteristics as the ones surrounding a local maximum on the centerline.

6. **Local minima off the centerline** (depicted as red dotted hexagons). Between maxima on and off the centerline, there may also be local minima off the centerline that are indistinguishable from those on the centerline as long as only local filters are applied.

We define a set of eight gradient vector field vectors in a $2^3$ cell of voxel positions to be uniform if and only if all eight vectors differ by less than 90 degrees from their average direction. Thus, to label local nonuniform gradient vector field vectors we:

1. Compute the normalized average gradient vector field vector for each $2^3$ cell of voxel positions.
2. Test if the dot-product between any of the $2^3$ gradient vector field vectors and the associated average vector is zero or negative.
3. Label all $2^3$ voxel cells that have nonpositive dot-products as nonuniform.

It is possible that a voxel is exactly at a local distance from boundary field maximum and its gradient vector field vector is a zero vector. Note that this leads to a zero-value dot-product, which does cause labeling of that position. Thus, no special case handling for degenerate gradient vector field vectors is necessary.

As each test involves only a small eight voxel position neighborhood, it can be executed very quickly (14 s for an average colon) and it produces flagged voxels that are only 1 percent of the inside colon voxels (see Fig. 16).
4.1.6 Connect Flagged Voxels

The previous penalized-distance algorithm step results in a number of disconnected components of flagged voxels. We connect these components by applying the following procedure for each flagged voxel.

1. Pick a voxel from the flagged source volume and flag the corresponding voxel in the destination volume.
2. Starting from that flagged voxel, traverse the source volume along the gradient vector field vectors and flag all voxels along the path in the destination volume.
3. Stop the traversal as soon as another flagged voxel is reached.

Fig. 17 shows an example in which, starting from a minimum, the path taken toward the next maximum results in a sequence of voxels along a path that is centralized in the distance from boundary field. In fact, the walk is self-correcting: Each gradient vector field vector can be viewed as a combination of the direction along the continuous centerline and the direction toward it. Due to the discrete nature of the path, we might reach a voxel to the left of the continuous centerline. At that position, the component of the gradient vector field vector toward the centerline points to the right. Thus, the next step brings us back to the right. Any error caused by discretization is corrected in the next step.

Starting from flagged voxels, near but off the centerline, the followed path is directed toward the centerline and eventually merges with one of the paths starting from a local minimum on the centerline. Consequently, this method also connects the off centerline minima and maxima to the set of flagged voxels on the centerline. At the end, we have labeled a set of voxels that includes all centerline voxels and totals just 15-30 percent of all colon voxels. All further operations are restricted to this set of voxels.

4.1.7 Compute Distance from Source Field

The following three algorithm steps are an adaptation of the Dijkstra shortest path graph algorithm as outlined in Sections 2.3 and 3. This first step computes the distance from source field with the anisotropically correct Euclidian distance as the weights in the implicit graph of Fig. 11a. Independently of the choice of 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. Fig. 19a shows the resulting distance from source field.

4.1.8 Compute Penalized Distance from Root 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 Fig. 19b.

For our implementation, we did not need to explicitly store all 26 penalty vertices and edges depicted in Fig. 11b 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

Fig. 17. Connecting flagged voxels. Starting from each flagged voxel, we follow along the gradient vector field vectors until we reach another flagged voxel.

Fig. 18. (a) Zoomed and (b) complete colon with all flagged voxels highlighted.
vertices from the distance from source field generation method, but add the penalty directly to the computation of the accumulated cost at each voxel $v$:

$$cost(v_k) = cost(v_{k-1}) + distance(v_k, v_{k-1}) + penalty(v_k).$$

Simultaneously, we compute the accumulation of the Euclidian distance through:

$$accumulatedDistance(v_k) = accumulatedDistance(v_{k-1}) + distance(v_k, v_{k-1}).$$

We use this distance to identify the farthest voxel in the Penalized Distance from Root Field, which is the other end of the centerline. Maintaining this separate Euclidian distance is the main reason for overcoming the “spike” problem of the CEASAR algorithm, as pointed out in Fig. 9.

### 4.1.9 Minimum Cost Path (Centerline)

We now run the second step of the Dijkstra algorithm that traces the discrete centerline from the end voxel back to the root. Because of our inclusion of strong penalties into the penalized distance from root field, the algorithm finds a global minimum cost path that is optimally centered following maximal values of the distance from boundary field. This path is the discrete centerline and becomes the initial main branch of our discrete skeleton tree, as defined in Section 3 and depicted in Fig. 21a.

### 4.1.10 Smooth the Centerline

For some applications, an optimal discrete centerline is not the most desired centralized path. For example, in guided virtual endoscopy, the camera is moved along the centralized path. To maintain a steady view, a smooth curve is favored over a discrete “stair step” path. Given the optimal discrete path and the distance from the colon boundary at each centerline voxel, we can compute an approximating spline [14], [19] with adaptive error tolerance. In very narrow areas, the allowed error should be very small, while, in wide openings, a little larger divergence is acceptable.

This can be elegantly expressed as a percentage of the distance from boundary. Any percentage below 50 percent guarantees that the centerline is always closer to the center than to the boundary. This additional freedom to place the continuous centerline is then used to minimize curvature along the centerline. Specifically, we use a B-spline curve that interpolates the first and last voxel and approximates the ones in between. The control points are placed close to centerline voxels at nonuniform intervals. We apply a number of heuristics (min/max curvature, min/max DBF, maximal control point separation) to minimize the number of control points that is needed to achieve the desired accuracy. In the example of Fig. 20, an error tolerance of 35 percent requires 17 percent of the discrete centerline voxels as control points, while a 50 percent tolerance needs 13 percent and results in a smoother centerline.

### 4.2 Skeleton Computation

Steps 11-16 of our algorithm (Fig. 12) extend the initial centerline of an object to its complete skeleton. These steps are now described in detail.

#### 4.2.1 Label Voxels Near the Skeleton

After setting the skeleton path, we “roll an adaptive sphere down all skeleton voxels.” We say that the sphere is adaptive because the radius is computed for each voxel on the path to be:

$$radius(v) = DistanceFromBoundary(v) \cdot scale + const.$$

Specifically, we label all object voxels within the radius to become processed 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 results 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. Fig. 21b shows color coding of the regions belonging to each skeleton branch.
4.2.2 Compute New Penalized Distance from Root Field

To find the next branch, we compute a new Penalized Distance from Root Field with all skeleton voxels as the initial source (cost and distance zero). The now furthest voxel is the tip of the next major skeleton branch.

This repeated Penalized Distance from Root Field computation is different from the TEASAR algorithm and ensures branches join the main skeleton independent of the order in which the centerline endpoints were picked. It also fixes the late join on “Y-fork” objects problem.

4.2.3 Minimum Cost Path (Skeleton Branch)

Tracing back from the branch tip voxel along the Penalized Distance from Root Field identifies the voxels of the current branch. The back trace is finished as soon as any of the prior skeleton voxels is reached.

Fig. 20. (a) Zoomed and (b) complete colon with the centerline after smoothing.

Fig. 21. (a) Aorta with shortest path after one iteration. (b) Aorta cut open to show regions labeled during multiple iterations. (c) Complete aortic skeleton.

<table>
<thead>
<tr>
<th>dataset</th>
<th>Lobster</th>
<th>Colon</th>
<th>Aorta</th>
<th>Ribs</th>
<th>Dinosaur</th>
</tr>
</thead>
<tbody>
<tr>
<td>original size X</td>
<td>256</td>
<td>514</td>
<td>256</td>
<td>512</td>
<td>512</td>
</tr>
<tr>
<td>original size Y</td>
<td>254</td>
<td>514</td>
<td>256</td>
<td>512</td>
<td>512</td>
</tr>
<tr>
<td>original size Z</td>
<td>57</td>
<td>363</td>
<td>211</td>
<td>247</td>
<td>512</td>
</tr>
<tr>
<td>cropped size X</td>
<td>242</td>
<td>416</td>
<td>193</td>
<td>351</td>
<td>173</td>
</tr>
<tr>
<td>cropped size Y</td>
<td>241</td>
<td>398</td>
<td>162</td>
<td>248</td>
<td>226</td>
</tr>
<tr>
<td>cropped size Z</td>
<td>37</td>
<td>363</td>
<td>211</td>
<td>247</td>
<td>509</td>
</tr>
<tr>
<td>all data voxels</td>
<td>96M</td>
<td>96M</td>
<td>13M</td>
<td>65M</td>
<td>129M</td>
</tr>
<tr>
<td>cropped voxels</td>
<td>59M</td>
<td>59M</td>
<td>6.7M</td>
<td>22M</td>
<td>20M</td>
</tr>
<tr>
<td>inside voxels</td>
<td>141K</td>
<td>3.2M</td>
<td>230K</td>
<td>1M</td>
<td>0.8M</td>
</tr>
<tr>
<td>skeleton voxels</td>
<td>1639</td>
<td>1644</td>
<td>850</td>
<td>5251</td>
<td>8078</td>
</tr>
<tr>
<td>skeleton branches</td>
<td>18</td>
<td>1</td>
<td>9</td>
<td>18</td>
<td>7</td>
</tr>
</tbody>
</table>
4.2.4 Smooth the Branch

The same smoothing that was already described in Section 4.1 for the main centerline is repeated for each branch.

4.2.5 Add the Branch to the Skeleton

Finally, the set of skeleton voxels is extended to also include the voxels of the current branch.

4.2.6 Repeat Branch Finding

We repeat the last five steps until the labeling procedure converts all object voxel into processed voxels. The union of all shortest paths is the desired penalized-distance algorithm skeleton.

5 RESULTS

We tested our algorithm on CT scans of a lobster, a human colon, a rib cage, an aorta dataset, and a voxelized dinosaur. Table 1 lists the details about the dataset sizes and the number of processed voxels. In all cases, the discrete skeleton was placed right in the center according to visual inspection and according to mathematical measurements such as the distance from boundary.

All tests were performed on a Wintel PC running Windows2000 using a single 1 GHz CPU. The times to compute the skeletons for the lobster, colon, aorta, rib cage, and dinosaur data were 87, 119, 36, 194, and 389 seconds, respectively. Note that all running times were below seven minutes and that the computation was as quick as 36 seconds for the aorta. Fig. 22 depicts the final skeletons computed with their associated volumes.

6 CONCLUSIONS

We introduced the efficient penalized-distance algorithm for smooth and accurate volumetric skeletons that are free of degenerate cases. Our algorithm is based on a new, robust definition of a skeleton. It computes the skeletons automatically, regardless of the geometry of the connected object. The skeleton branches are centrally located within their associated object regions. We explained our implementation in detail and reported results that not only empirically verify the correctness of the skeleton, but also demonstrated the superior speed of the penalized-distance algorithm that is less than seven minutes for all our test datasets.

ACKNOWLEDGMENTS

This work has been supported by grants from the US National Institute of Health CA79180, US Office of Naval Research N000149710402, 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 and Sören Grimm for helping with the implementation. Mie Sato conducted this research while visiting the State University of New York at Stony Brook.

REFERENCES


Ingmar Bitter is pursuing his PhD degree in computer science at the State University of New York at Stony Brook. He is Director of Research and Development at Viatronix Inc. His research interests include algorithms and hardware for volume rendering and volume analysis and modeling, as well as computer graphics and computational geometry. For more information, see http://www.cs.sunysb.edu/~ingmar.

Arie E. Kaufman received the BS degree in mathematics and physics from the Hebrew University of Jerusalem in 1969, the MS degree in computer science from the Weizmann Institute of Science, Rehovot, in 1973, and the PhD degree in computer science from the Ben-Gurion University, Israel, in 1977. He is the director of the Center for Visual Computing (CVC), a leading professor and chair of the Computer Science Department, and leading professor of Radiology at the State University of New York at Stony Brook. He was the founding editor-in-chief of the IEEE Transactions on Visualization and Computer Graphics (TVCG), 1995-1998. Dr. Kaufman has been the cochair for multiple Eurographics/Siggraph Graphics Hardware Workshops, the papers or program cochair for the IEEE Visualization ‘90-’94 and ACM Volume Visualization Symposium ‘92, ’94, ’98, and the cofounder and member of the steering committee of the IEEE Visualization conference series. He has previously chaired and is currently a director of the IEEE Computer Society Technical Committee on Visualization and Computer Graphics. He is the recipient of a 1995 IEEE Outstanding Contribution Award, the 1996 IEEE Computer Society’s Golden Core Member, 1998 IEEE Fellow, 1998 ACM Service Award, and 1999 IEEE Computer Society’s Mentornship Service Award. Dr. Kaufman has conducted research and consulted for about 30 years, specializing in volume visualization; graphics architectures, algorithms, and languages; virtual reality; user interfaces; and multimedia. For more information see http://www.cs.sunysb.edu/~ari.

Mie Sato received the BS and MS degrees in mathematics from Tsuda College, Tokyo, Japan, in 1995 and 1997, respectively, and the DrEng degree in computer science from the Tokyo Institute of Technology, Tokyo, Japan, in 2001. She was a visiting researcher in 1999 and 2000 at the State University of New York at Stony Brook. She is currently a postdoctoral fellow in the Department of Computer Science, the Swiss Federal Institute of Technology, Zurich, Switzerland. Her research interests include image processing and scientific visualization.

For more information on this or any computing topic, please visit our Digital Library at http://computer.org/publications/dlib.