Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization (2003)
G.-P. Bonneau, S. Hahmann, C. D. Hansen (Editors)

A Robust Level-Set Algorithm for Centerline Extraction

Alexandru Telea [1], Anna Vilanova [2]

[1] Department of Mathematics and Computer Science, [2] Department of Biomedical Technology,
Eindhoven University of Technology Den Dolech 2, 5600 MB Eindhoven, the Netherlands,
[1] `a.c.telea@tue.nl`,[2] `a.vilanova@tue.nl`

Abstract

We present a robust method for extracting 3D centerlines from volumetric datasets. We start from a 2D skeletonization method to locate voxels centered with respect to three orthogonal slicing directions. Next, we introduce a new detection criterion to extract the centerline voxels from the above skeletons, followed by a thinning, reconnection, and a ranking step. Overall, the proposed method produces centerlines that are object-centered, connected, one voxel thick, robust with respect to object noisiness, handles arbitrary object topologies, comes with a simple pruning threshold, and is fast to compute. We compare our results with two other methods on a variety of real-world datasets.

1. Introduction

Skeletons and centerlines have become increasingly popular in a number of application areas, such as computer vision [2, 11], medical visualization [18, 17, 16], feature representation and tracking [10], and geometric modeling [9, 3, 8]. Sample applications using skeletons and centerlines are object modeling and simplification and camera path planning for 3D navigation. Since many (often conflicting) centerline and skeleton definitions exist, we make the following distinctions. A 3D object's skeleton is the locus of maximal 3D balls contained in the object. A 3D skeleton is, in general, a two-dimensional collection of points, lines, and surfaces [2, 5]. Centerlines are closely related to skeletons. A centerline is a one-dimensional object (a set of curves embedded in 3D) which captures a 3D object's main symmetry axes, but does not contain detailed information about the object boundary. Centerlines provide a compact, efficient, and simple to analyze description for tubular structures such as blood vessels, neurons, or elongated organs such as the human colon [16, 17, 1]. As a simple example, Fig. 1 shows the surface skeleton (a) and centerline (without branches) (b) of a 3D box.

Regardless of the extraction method used, several requirements exist for computing centerlines from volumetric datasets. Given the numerous (often purely algorithmic) centerline definitions, it is hard to come with unique criteria, such as what is 'thin' or 'centered'. However, it is widely accepted [18, 17, 4] that a centerline should be:

1. **connected:** the centerline of a compact 3D object should be a set of connected voxels (R_1).
2. **centered:** centerline voxels should be locally centered with respect to the object's boundary (R_2).
3. **thin:** centerlines, represented as voxelized curves, should be as thin as possible – ideally, not thicker than one voxel (R_3).
4. **insensitive to boundary noise:** small surface details should not produce large twists or numerous small branches on the centerline. A simple threshold should allow gradual removal of such branches, keeping the object thin and connected (R_4).
5. **efficient to compute:** computing centerlines should run robustly and quickly on large volumes (R_5).

In some cases, an extra requirement (R_6) is that the centerline should capture topological features such as bifurcations and holes via branches and loops, respectively. More specific requirements exist too, such as centerline smoothness or object reconstructibility from the centerline. However, one can not satisfy all these requirements simultaneously [4, 14]. In the following, we shall consider requirements R_1 to R_6.

In this paper, we present a new algorithm for extracting voxel-based centerlines from 3D objects. Our algorithm dwells upon the Augmented Fast-Marching Method (AFMM) [15] used for 2D skeletonization. The algorithm we propose here produces one voxel thick, centered, and connected centerlines, is robust with respect to boundary noise, handles 3D objects of any topologies, and has a single intuitive threshold for removing undesired branches ($R_1..R_6$). The paper is organized as follows. Section 2 overviews several existing centerline algorithms. Section 3 presents the new method's six steps: preprocessing (Sec. 3.2), 2D skeleton detector extraction (Sec. 3.3), centerline detector computaton (Sec. 3.4), thinning (Sec. 3.5), reconnection (Sec. 3.6), and ranking (Sec. 3.7). Section 4 discusses the obtained results and algorithm implementation. Section 5 concludes the paper and outlines future improvements.

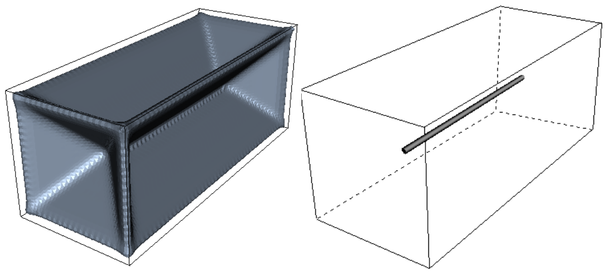

Figure 1: *a) Skeleton; b) Centerline*

2. Background

Centerlines and the more general skeletons are produced by three method classes [18, 17]: topological thinning, distance-based methods, and polygon-based methods. The last class (including Voronoi diagram [9] and boundary shrinking [14] methods) models objects and skeletons as polygonal objects. We focus here on voxel-based modeling, both for the object and its centerline, so we shall discuss only the first two classes.

2.1. Topological thinning

Thinning methods iteratively peel off boundary voxels (also called simple points) whose deletion does not alter the object topology (R_1,R_6). Parallel algorithms achieve high speeds by deleting a whole set of such points at a time [16, 13, 6]. Efficient detection of these points is done by checking a point's (small, usual 3x3) neighborhood against a set of deletion templates. Special templates prevent deletion of branch *end points*, i.e. those simple points which encode an object's geometry. Thinning produces usually connected (R_1), one voxel thick (R_3), though not necessarily geometrically centered, structures [4, 6]. Moreover, preserving end points leads to many undesired small branches. Template sets exist for producing both surface skeletons and centerlines [13, 6]. Finally, objects with complex topologies (e.g. holes) are correctly handled (R_6).

2.2. Distance-based methods

These methods compute the boundary's distance transform (DT) or distance map and define the skeleton as the DT's local maxima, or 'ridges'. This definition produces surface skeletons, as opposed to centerlines. However, these structures are usually disconnected, not guaranteed to be one voxel thick, and quite sensitive to boundary noise. Several DT computation methods exist, usually trading speed for precision (Chamfer-based, level-set based [15, 12], or true Euclidean [7]). Several enhancements of the basic idea exist, as follows. Zhou et al. [18] connect the directly extracted local maxima by local maximum paths (LM-paths), whose construction respects R_1, R_2, and R_3. However, the authors mention that this method is still sensitive to the DT accuracy. Moreover, the local maximum extraction might produce surface skeletons and not centerlines, e.g. in the case of an axis-aligned 3D box (such as in Fig. 1). Indeed, if all box's sizes are different, there would exist a planar rectangular set of voxels in the middle of the box which would comply with the local maximum criterion. An enhancement of this method [17] uses a second distance field (the SS-field) that encodes the distance to a given seed voxel set. From a cluster, i.e. all voxels with the same SS-field value, the one having maximal DT-value is added to the centerline. Various geometric, topologic, and ordering criteria are provided to cope with multiple centerline branches, multiple DT maxima for a cluster, and centerline reconnection and smoothing. A separate class of methods extracts centerlines without branches, used for camera path-panning in medical applications. For example, the CEASAR algorithm [1] computes the DT's gradient and finds voxels on and close to the centerline by analyzing the gradient's local variations, followed by reconnection of these structures. The final centerline is computed by applying various cost-based path tracing methods on the connected set of extracted voxels. However matching $R_1..R_5$, this algorithm does not handle objects with branches or holes. It is also not very clear how sensitive the algorithm is to the DT gradient-based detection and to the volume's discretization.

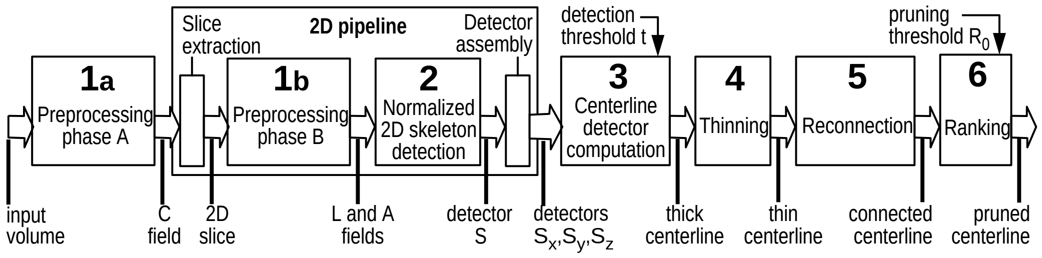

Figure 2: *Centerline extraction pipeline*

3. New Method

3.1. Overview

The proposed method extends and enriches the application of the AFMM to centerline extraction [15]. In [15], a tentative method for 3D centerline extraction is proposed, as follows. First, binary 2D skeletons (one marks the skeleton, zero marks the background) are extracted from each axis-aligned 2D data slice, yielding three volume 'stacks' of 2D skeletons, called the X, Y, and Z skeletons. Next, the centerline is taken as per-voxel binary intersection of the X,Y, and Z skeletons. This yields those voxels at maximal boundary distance, measured in the three orthogonal 2D slices. As mentioned in [15], the method has several serious drawbacks. The produced centerline is not connected, not one-voxel thick, and contains undesired branches. Moreover, the method is very sensitive to the dataset resolution and the 2D skeleton threshold. Slight resolution and/or threshold changes make the one pixel thick, discrete X, Y, and Z skeletons, 'miss' each other, yielding an empty intersection, i.e. a missing centerline voxel. Lowering the skeleton threshold produces more branches, thus potentially less missing centerline points. However, this leads to numerous undesired, not object-centered, branches as well as centerlines that are several voxels thick. Given all these, the above method is not suited for application on real world datasets.

The new method we propose has six steps, as follows: preprocessing, normalized 2D skeleton detector extraction, centerline detector computation, thinning, reconnection, and ranking (Fig. 2). These steps are described next. As running example, we use the simple object in Fig. 3.

3.2. Preprocessing

We work with 3D uniformly sampled binary volumes containing object (full), background (empty) and boundary (full, neighbor with empty) voxels. In the extracted 2D slices, we have object, background, and boundary pixels respectively. We follow the definitions in [4] of face (F), edge (E), and vertex (V) connectivity. We define a voxel (pixel) *component* as a set of nonempty voxels (respectively pixels) such that any two of them are V-connected via a nonempty voxel (recpectively pixel) path. Background voxels are thus F-connected.

Preprocessing has two phases. In phase A, all 3D disconnected object components (i.e. separate objects in the input volume) are found and all object voxels p are flagged in a field $C(p)$ by their component index. Phase B is done just before the 2D slices extracted from the 3D volume are skeletonized. In this phase, every 2D component of the image to be skeletonized is detected and its boundary length is computed. The boundary length of components with holes is the sum of the lengths of all the separate (inner and outer) boundary segments. We define then the fields L and A such that, for every pixel p, $L(p)$ equals the boundary length and $A(p)$ the area of the component p belongs to. Next, we eliminate (set to background) all pixels for which $A(p)$ is smaller than a given threshold and fill (set to object) all pixels for which $A(p)$ is larger than the same threshold. In practice, we set the elimination threshold to 2% of the maximum $A(p)$ on all components. This eliminates the small noise elements in the data. Hole elimination is important, since the smallest hole drastically affects the object's skeleton topology by creating undesired branches and loops.

3.3. Normalized 2D skeleton detector extraction

This step starts by applying the AFMM algorithm on the preprocessed 2D slices. We summarize the steps of the AFMM algorithm (see [15] for a detalied discussion):

1. initialize a distance-to-boundary field T to 0 on the boundary.
2. initialize a field U to a monotonically increasing numbering of the boundary pixels, starting from a random boundary location.

3. propagate T and U to all object pixels.
4. compute U's sharp discontinuity detector $D = max(\partial U/\partial x, \partial U/\partial y)$, the derivatives being computed via forward or central differences.
5. define skeleton pixels as those where D is larger than some threshold.

In contrast to the original AFMM, we now stop after step 4 and define the normalized skeleton detector for a pixel p as $S(p) = D(p)/L(p)$. Since $U(p) < L(p)$, $S(p)$ is normalized between 0 and 1 and represents the length of the boundary segment that collapsed into skeleton point p, normalized with respect to the total boundary length. Normalization is crucial for the next step (Sec. 3.4). Since the AFMM handles any topologies, requirement R_6 is fulfilled. We end this step by assembling all 2D detectors S for the X, Y, and Z slicing directions in three scalar volumes S_x, S_y, and S_z.

3.4. Centerline detector computation

In this step, a subset of the centerline voxels is detected. We define the centerline CL as the voxels c of the object O on which the detector function $d : S_x \times S_y \times S_z \rightarrow \mathbb{R}$ computed on the volumes S_x, S_y, and S_z, exceeds a threshold t:

$$CL = \{c \in O | d(S_x(c), S_y(c), S_z(c)) > t\} \qquad (1)$$

Several options exist for the detector d. The extraction method presented in [15] amounts to the detector $d_{AND} = (S_x > t) \wedge (S_y > t) \wedge (S_z > t)$ where S_x, S_y, and S_z are *binary* skeleton stacks. We discussed the important limitations of d_{AND} in Sec. 3.1. A better solution is to use $d_{prod} = S_x \cdot S_y \cdot S_z$, where S_x, S_y, and S_z are now the *continuous*, normalized skeleton detectors (Sec. 3.3). Similarly to d_{AND}, d_{prod} is maximal in the points where all three axis-aligned detectors S_x, S_y, and S_z are maximal. However, d_{AND} has a binary behavior, i.e. it either fully selects or rejects voxels. As outlined in Sec. 3.1, this yields either a very sparse centerline, for high t values, or many undesired, not centered branches, and a thick centerline, for low t values. Since t, for d_{AND},is an absolute boundary length, in pixels, large crossections deliver too many branches, whereas small ones are empty.

In contrast, d_{prod} has two advantages. First, it uses the per-boundary-length normalized detectors S, which allow correct comparison of skeletons of crossections having different lengths. Secondly, it uses a continuous combination of the continuous detectors S rather than a binary intersection of binary detectors. d_{prod} expresses 'centrality' with respect to the three slicing axes. Voxels on less central skeleton branches on one slicing axis i (lower S_i) are now selected only if they are central on another slicing axis j (high S_j). Conversely, central voxels in one slicing direction are rejected if they are too far off-center in another direction. Varying t with d_{prod} gradually and uniformly populates the centerline with object-centered branches. This step enforces thus requirements R_2 and R_4. In practice, setting t between 0.03 and 0.1 has given good results for all tested datasets. This corresponds to finding centerline voxels that are produced by surface features (in the axis-aligned cross-sections) of 3 to 6% of the cross-section length. For example, Figs. 3 a) and e) show the detected centerline voxels for $t = 0.01$ and $t = 0.15$ respectively. Even for this large t variation, the results (number and place of detected voxels) are very similar. Overall thus, this step enforces requirements R_2 and R_6.

3.5. Thinning

The previous step delivers a thick (3..8 voxels) centerline, consisting of disconnected components separated by gaps. We enforce one voxel thickness in this step (requirement R_3) by applying the directional 3D thinning method of Palagyi and Kuba [13] (denoted PK in the following). This algorithm uses a number of 3x3 templates to produce, in eight parallel subiterations, a 'curve skeleton' of the input 3D object. Special templates are used to preserve the branch end points from deletion (see Sec. 2). The algorithm delivers one voxel thin components for the thick input components, i.e. preserves the connectivity of the input data. The template choice and application order ensures that the algorithm is orientation independent. Other thinning algorithms, such as the one presented by Vilanova, König, and Gröller [16] (denoted VKG in the following) can be used instead of the PK method, if they preserve input connectivity and branch end points and deliver one voxel thin components.

One may be tempted to produce the centerline by direct thinning of the 3D object. This approach has several problems. First, due to their local nature, thinning methods do *not* usually guarantee the centeredness of their output [6, 13, 4]. Moreover, some thinning methods are also orientation dependent. Voxel deletion order may be used to enforce the centerline to be *roughly* close to the object's center [4, 6]. However, thinning even smooth and simple objects (e.g. Fig. 4 and Fig. 12 a-c), produces noisy, ill centered structures (Fig. 4 b,c and Fig. 12 b). Such centerlines usually have many small branches, since end points must be preserved (see Sec. 2). Finally, many thinning implementations on large volumes are slow. In contrast, for the three-rings structure in Fig. 4, we obtain the exact circular shape for the centerline, as visible in the side view in Fig. 4 d.

Our approach avoids such problems. Our thinning

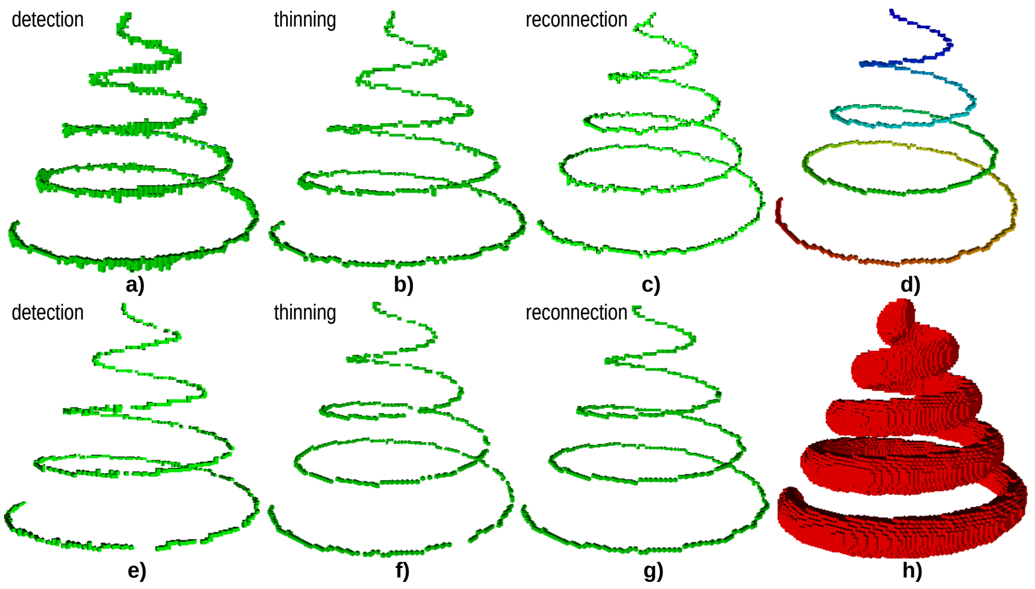

Figure 3: *Centerline extraction steps: detection (a,e), thinning (b,f), reconnection (c,g), for two different detection thresholds. Ranking (d) and initial object (h)*

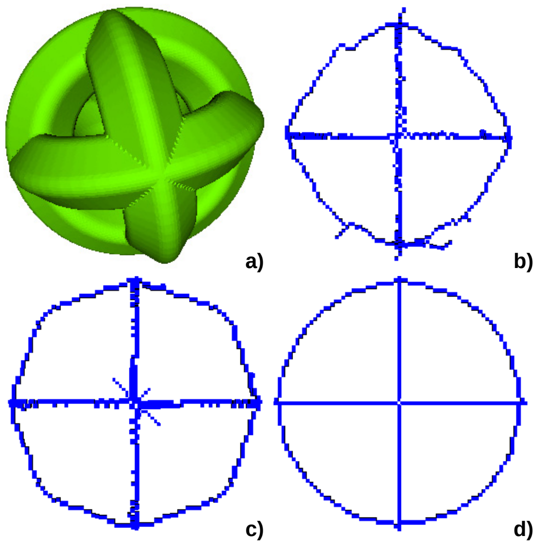

Figure 4: *Three-rings object (a). Centerlines: PK thining method (b), VKG thinning method (see also Sec. 4) (c), our method (d)*

input is only a few voxels thick and *already* centered, so the ill centeredness, orientation dependency, and speed problems of the chosen thinning method have practically no impact (see example in Fig. 3 b,f). Finally, augmenting the AFMM to use the produced centerlines for object reconstruction, approximation, and matching, is a promising research area.

3.6. Reconnection

In this step, the thin centerline components output by the previous step are reconnected, thus enforcing requirement R_1 (see example in Fig. 3 c,g). Several reconnection strategies exist, e.g. the local maximum path (LMpath) method presented in [17]. LMpaths connect centerline components, detected in [17] as local maxima, by thin, centered voxel sequences that follow the object boundary's DT gradient.

We propose an alternative technique that does not need the object boundary's DT or its gradient. We use the fact that the gaps in the centerline caused by our detector (Sec. 3.4) are very small, a few voxels only in most cases. First, we sort all disconnected centerline components c_i in increasing size (number of voxels) order, using a hash table. Next, we apply an algorithm that iteratively merges the smallest component with the closest component (see Fig. 5). Only components having the same C value (see Sec. 3.2) are merged, to prevent merging centerlines of independent, logicaly disconnected objects. The process stops when no more components can be merged. The distance between two components (function `closest()`in Fig. 5) is computed as the minimum of their inter-voxel distances. Merging is done by constructing a 3D voxel line between the closest two voxels of the components

```
struct Component { VoxelArray v; };

void reconnect(HashTable h)
{
 while (h.size()>1)
 {
  Component c1 = h.removeElem(0),c2,c;   //c1 = h[0] = smallest component
  Voxel v1,v2,va,vb;
  float dmin = MAX_DIST;                 //MAX_DIST is e.g. 10^6
  for(int i=1;i<h.size();i++)            //try merging c1 with another
  {                                      //component
   c = h[i];
   if (C(c1.v[0])!=C(c.v[0])) continue;  //c1,c not in same object
   float d = closest(c1,c,va,vb);        //va,vb=closest voxels in c1,c
   if (d<dmin)
   { v1=va; v2=vb; c2=c; dmin=d; }
  }

  if (dmin==MAX_DIST)                    //c1 can't be merged
   delete c1 from h;                     //eliminate c1 from reconnection
  else
  {
   add voxels of c2 to c1;               //merge c2 into c1
   VoxelArray line = drawLine(v1,v2);    //voxelize line v1-v2
   add voxels in line[] to c1;
   delete c2 from h;
   reinsert c1 in h;                     //needed as c1's size changed
  }
 }
}
```

Figure 5: *Reconnection algorithm*

to be merged (function `drawLine()`). Other solutions, such as LMpaths [17] or tracing steepest ascent paths in the DT gradient field [1] could be used too. However, given the small gap size delivered by our detector (see Sec. 3.4), tracing lines is a simple, cheap, and acceptable solution. Note that the LMpaths are also, in many cases, line segments.

Picking the smallest component is a good heuristic for minimizing the number of distance computations. Overall, reconnection takes less than a second even for hundreds of components of thousands of voxels in total.

3.7. Ranking

The previous step delivers a one voxel thick, connected centerline. In this last step, we assign a rank R to every centerline voxel, according to their distance, along the centerline, to an end point. This allows pruning centerline branches in function of their importance, as explained next. First, we describe the ranking algorithm (see also Fig. 6). We start initializing R to 0 for background voxels, 1 for centerline voxels, and 2 for centerline end points. End points are detected as those centerline voxels having exactly one centerline voxel V-neighbor, a property enforced by the PK thinning method (see [13] for details). In step 2, we propagate the rank values along the centerline branches until bifurcations are met, increasing the rank with 1 at every newly found point. Bifurcations are found as points with more than two centerline V-neighbors, a property enforced by the PK thinning method [13]. If we reach a previously ranked end point during propagation, we overwrite its rank if the propagated rank is higher.

Propagation is done in 'depth first' order, i.e. from the highest ranked voxel on, by keeping the traversed voxels in a hash table sorted in descending R order. When all points on branches have been ranked, we have found a number of bifurcation points. In step 3, we rank these points with the maximum of their discovered neighbors' ranks plus 1, i.e. with the length of the longest incident branch discovered. Finally, we rank their undiscovered neighbors with the bifurcation point's rank plus 1, and restart the algorithm with these neighbors as end points. The Color Plate (a,e,f,h,i,k) shows several ranked centerlines, using a rainbow colormap for the rank values.

Selecting those voxels having $R > R_0$ prunes, from end points on, the centerline branches shorter than the given pruning threshold R_0. This allows trivial removal of the short 'noise' branches, without disconnecting the centerline, as the example in Fig. 7 shows. If desired, other strategies can be used, e.g. ranking all voxels on a branch with the branch length or finding the longest path.

4. Discussion

We have tested the presented method on several 3D datasets (Figs. 9,10,11,12). The extraction process statistics are shown in Fig. 8, for a Pentium III PC

```
HashTable h; Voxel p,q; int n;

for (all voxels p)                  //STEP 1. Initialize R field
  if (p is end point)
  { R(p) = 2; insert p in h; }
  else if (p is centerline point)
    R(p) = 1;
  else if (p is background point)
    R(p) = 0;

while (h not empty)                //STEP 2. Propagate R field
{
  VoxelArray bf;                   //bf keeps bifurcation points
  while (h not empty)
  {
    p = h.removeElem(0);           //q=h[0]=highest ranked endpoint
    q = neighbor of p with R==1
    n = number of q's neighbors with R==1
    if (n<3)                       //q is on the branch
    { R(q) = max(R(q),R(p)+1); insert q in h; }
    else
      insert q in bf;              //q is bifurcation point
  }

  for (all points p in bf)         //STEP 3. Rank bifurcation points
  {
    R(p) = 1+max(R) over all p's neighbors with R>1
    for (all p's neighbors q with R==1)
    {
      R(q) = R(p)+1;               //Create and rank new end points
      insert q in h;
    }
  }
}
```

Figure 6: *Ranking algorithm*

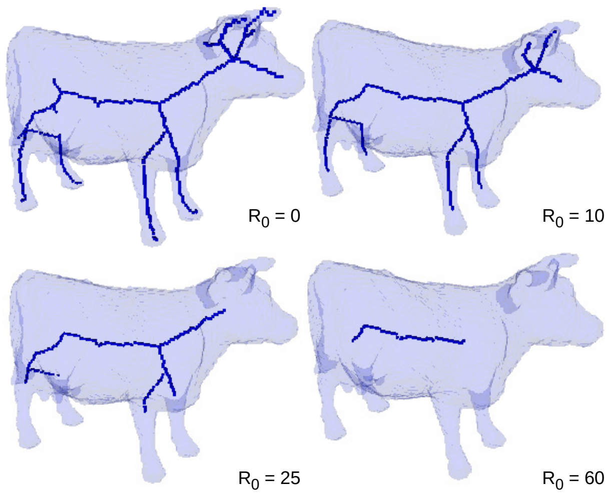

Figure 7: *Centerline pruning for different R_0 values*

running at 500 MHz with 128 MB memory. No separate per process step timings are done, since virtually all the time is spent for the 2D skeleton detector computation (Sec. 3.3). Our method's complexity is the same as for the AFMM skeletonization method [15], namely $O(N\ log\ N)$ for N object voxels. Figure 8 shows also the timings for the VKG thinning method (see Sec. 3.5), for a number of datasets. Finally, we note that our timings also compare favorably with the CEASAR algorithm [1].

Dataset name	rings	spiral	cow	twist	colon 1	colon 2	lobster
Dataset size X	100	100	165	150	256	381	180
Dataset size Y	100	100	107	150	256	120	213
Dataset size Z	100	100	64	150	311	632	282
Extracted voxels	631	992	326	2970	741	499	934
Components	14	6	39	357	165	134	90
Thinned voxels	87	411	72	790	184	91	305
Reconnection voxels	597	489	269	2885	702	915	510
Our method (sec)	7	4	6	17	130	163	45
VKG method (sec)	53	3		121	176	944	

Figure 8: *Extracted voxels, disconnected components, voxels removed by thinning, voxels added by rconnection, and total extraction time (colon 1: Fig. 11; colon 2: Fig. 10)*

It is interesting to compare the full 3D skeletons (extracted with the method presented in [12]) with the 3D centerlines (extracted by the method presented here). As Fig. 9 and Fig. 12 a,c (Color Plate a-b,f-l) show, centerlines are contained, and close to the centers, of the 3D skeletons – an insightful result, given that the skeletons and centerlines are extracted by two completely different methods.

Figures 10 and 11 show the centerlines, computed with our method and the VKG thinning method, on a dissected, respectively *in vivo*-scanned colon dataset. For the dissected colon (Fig. 10), the results are very similar. For the other dataset, our method delivers, even without pruning, less branches (Fig. 11 b,Color Plate c). A simple pruning yields a clean centerline (Fig. 11 b,Color Plate d). As in Fig. 4, Figures 12 a,b show that our method produces smoother, more centered structures than thinning methods. To outline the method's robustness, we show in Fig. 4 d the centerline we computed for a noisy, unsmooth object on a coarse dataset ($130 \times 130 \times 44$ voxels, pruning threshold $R_0 = 8$). Note, for comparison with other methods that perform post-extraction smoothing (e.g. [1]), that all examples shown here are *unsmoothed* centerlines.

The only user inputs the method has are the detection (t, Sec. 3.4) and pruning (R_0, Sec. 3.7) thresholds. As explained, t encodes our centerline model by specifying the minimal boundary feature length that causes a centerline point. Setting t to any value between 0.03 and 0.1 delivered practically identical results for all datasets we tried. Lower t values create too many branches, as the AFMM method becomes unstable for too small boundary features (see [15]). Higher t values may miss some secondary centerline branches. In practice, we simply fixed $t = 0.05$ so that users only need to set the pruning threshold R_0. Setting R_0 is simple, as it describes the length, in voxels, of the centerline branches to be pruned.

The thinning and reconnection steps are interchangeable, yielding the same result, since reconnection adds only one voxel thick line segments (Sec. 3.6). As most centerline methods, we use an implicit centerline model, defined via the three axis-aligned crossections. The choice of these slicing directions is indeed arbitrary, motivated only by the implementation simplicity. However, for a general 3D object exhibiting no local symmetry, it is very hard to give a unique, local definition of what a centerline-orthogonal crossection is. This advocates that our model based on the axis-aligned crossections is as acceptable as any other definition.

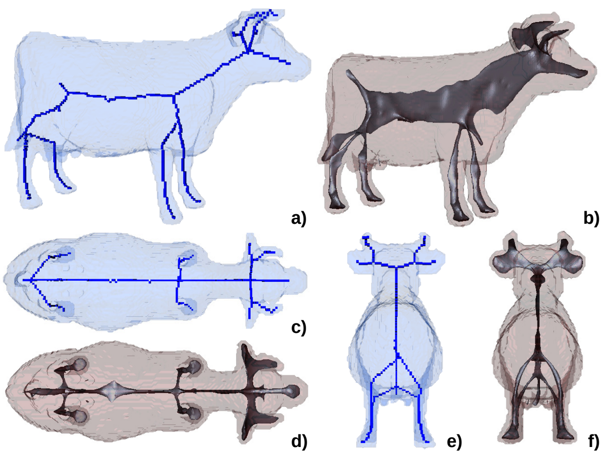

Figure 9: *3D skeletons (b,d,f) and centerlines (a,c,e) of the same object, different views*

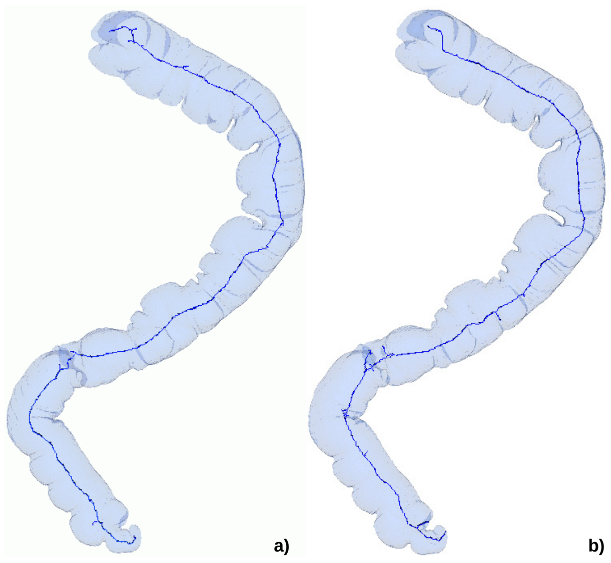

Figure 10: *Straightened colon centerline: our method (a), VKG method (b)*

For the implementation, we used the publicly available AFMM source code (see [15]). The PK thinning method [13] is trivial to implement. Finally, the reconnection and ranking steps are described in pseudocode in Secs. 3.6 and 3.7.

Acknowledgements

We thank Dr. Erich Sorantin from the Department of Radiology in Graz for providing the dissected colon dataset in Fig. 10. We also thank Dr. Dominik Fleischmann from the Department of Radiology at the AKH in Vienna for providing the colon dataset in Fig. 11.

5. Conclusion

The 3D centerline extraction method we have presented has a number of advantages, as follows. First, it delivers guaranteedly connected, object-centered, one voxel thin centerlines. Secondly, it works efficiently on

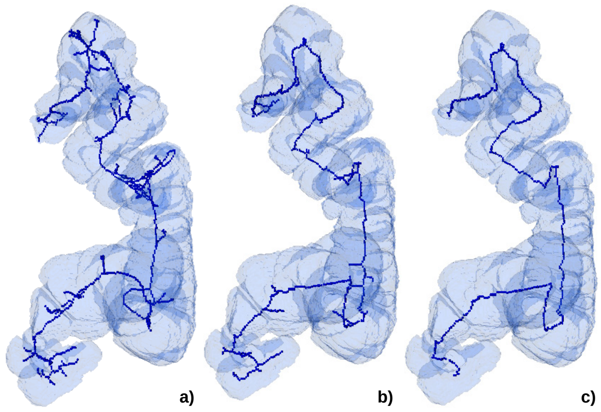

Figure 11: *Colon centerline: VKG method (a), our method $R_0 = 0$ (b) and $R_0 = 15$ (c)*

large, noisy, datasets. Thirdly, it captures topological features such as branches and holes. Fourthly, it is simple to implement, the core AFMM code being publicly available and well described in [15]. Finally, it has two intuitive, simple to set, user inputs: the detection threshold t and the pruning threshold R_0. No derivative and/or gradient computations are required, which makes the method stable.

Future research should address a number of issues. We are interested to exactly evaluate the method's 'worst case', i.e. the largest centerline disconnections the detection step (Sec. 3.4) may produce. More practically, considerable speed-ups can be achieved by using adaptive AFMM implementations to compute the 2D skeletons, making this technique interesting for near real time applications.

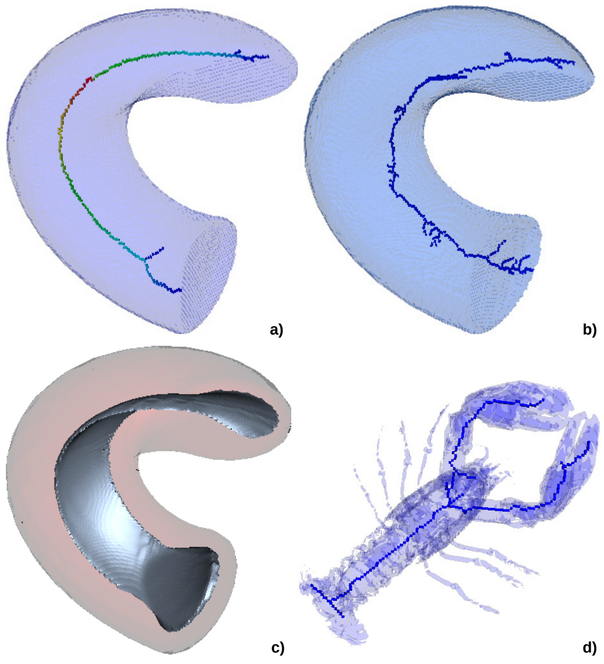

Figure 12: *Additional centerline examples. Twisted object: our method (a), VKG method (b), 3D skeleton (c). Lobster, our method (d)*

References

1. I. Bitter, M. Sato, M. Bender, K. McDonnell, A. Kaufman, M. Wan, *CEASAR: A Smooth, Accurate and Robust Centerline Extraction Algorithm*, Proc. IEEE Visualization '00, pp. 45–52, IEEE CS Press, 2000.
2. S. Bouix, K. Siddiqi, *Divergence-based Medial Surfaces*, Proc. ECCV 2000, pp. 603-618, 2000.
3. H. Blum, *A transformation for extracting new descriptors of shape*, In W. Walthen-Dunn, editor, *Models for the Perception of Speech and Visual Form*, MIT Press, 1967.
4. N. Gagvani, D. Silver, *Parameter Controlled Volume Thinning*, Graphical Models and Image Processing 61(3) Academic Press, pp. 149–164, 1999.
5. R. Kimmel, D. Shaked, N. Kiryati, A. M. Bruckstein, *Skeletonization vis Distance Maps and Level Sets*, Computer Vision and Image Understanding, vol. 62, no. 3, pp. 382-391, 1995.
6. A. Manzanera, T. Bernard, F. Preteux, B. Longuet, *Medial faces from a concise 3D thining algorithm*, Proc. ICCV '99, pp. 337–343, IEEE CS Press, 1999.
7. A. Meijster, J. Roerdink, W. Hesselink, *A genera algorithm for computing distance transforms in linear time*, Math. Morph. and its Appls. to Image and Signal Proc., pp. 331–340, Kluwer, 2000.
8. C.W. Niblack, P.B. Gibbons, D.W. Capson, *Generating skeletons and centerlines from the distance transform*, CVGIP: Graphical Models and Image Processing, nr. 54, 1992, pp. 420-437.
9. R. L. Ogniewicz, O. Kubler, *Hierarchic Voronoi Skeletons*, Pattern Recognition, nr. 28, 1995, pp. 343–359.
10. F. Reinders, M. E. D. Jacobson, F. H. Post, *Skeleton Graph Generation for Feature Shape Description*, Proc. IEEE VisSym 2000, Springer, 2000, pp. 73-82.

11. K. Siddiqi, S. Bouix, A. Tannenbaum, S. W. Zucker, *The Hamilton-Jacobi Skeleton*, Intl. Conf. on Computer Vision ICCV '99, p. 828–834, 1999.

12. M. Rumpf, A. Telea, *A Continuous Skeletonization Method Based on Level Sets*, Proc. IEEE VisSym '02, pp. 151–158, ACM Press, 2002.

13. K. Palagyi, A. Kuba, *Directional 3D Thinning using 8 Subiterations*, Proc. DGCI '99, LNCS 1568, pp. 325–226, Springer, 1999.

14. H. Schirmacher, M. Zöckler, D. Stalling, H.C. Hege, *Boundary Surface Shrinking - A Continuous Approach to 3D Center Line Extraction*, Proc. IMDSP '98, see also Tech. Report 7/98, IMMD IX, Univ. Erlangen-Nuernberg, 1998.

15. A. Telea, J. J. van Wijk, *An Augmented Fast Marching Method for Computing Skeletons and Centerlines*, Proc. IEEE VisSym '02, pp. 251–260, ACM Press, 2002.

16. A. Vilanova Bartrolí, A. König, E. Gröller, *VirEn: A Virtual Endoscopy System*, Machine Graphics & Vision, vol. 8, nr. 3, pp. 469–487, 1999.

17. Y. Zhou, A. Kaufman, A. W. Toga, *Three-dimensional skeleton and centerline generation based on an approximate minimum distance field*, The Visual Computer, no. 14, pp. 303–314, Springer, 1998.

18. Y. Zhou, A. W. Toga, *Efficient Skeletonization of Volumetric Objects*, IEEE TVCG, vol. 5, no. 3, 1999, pp. 210–225.

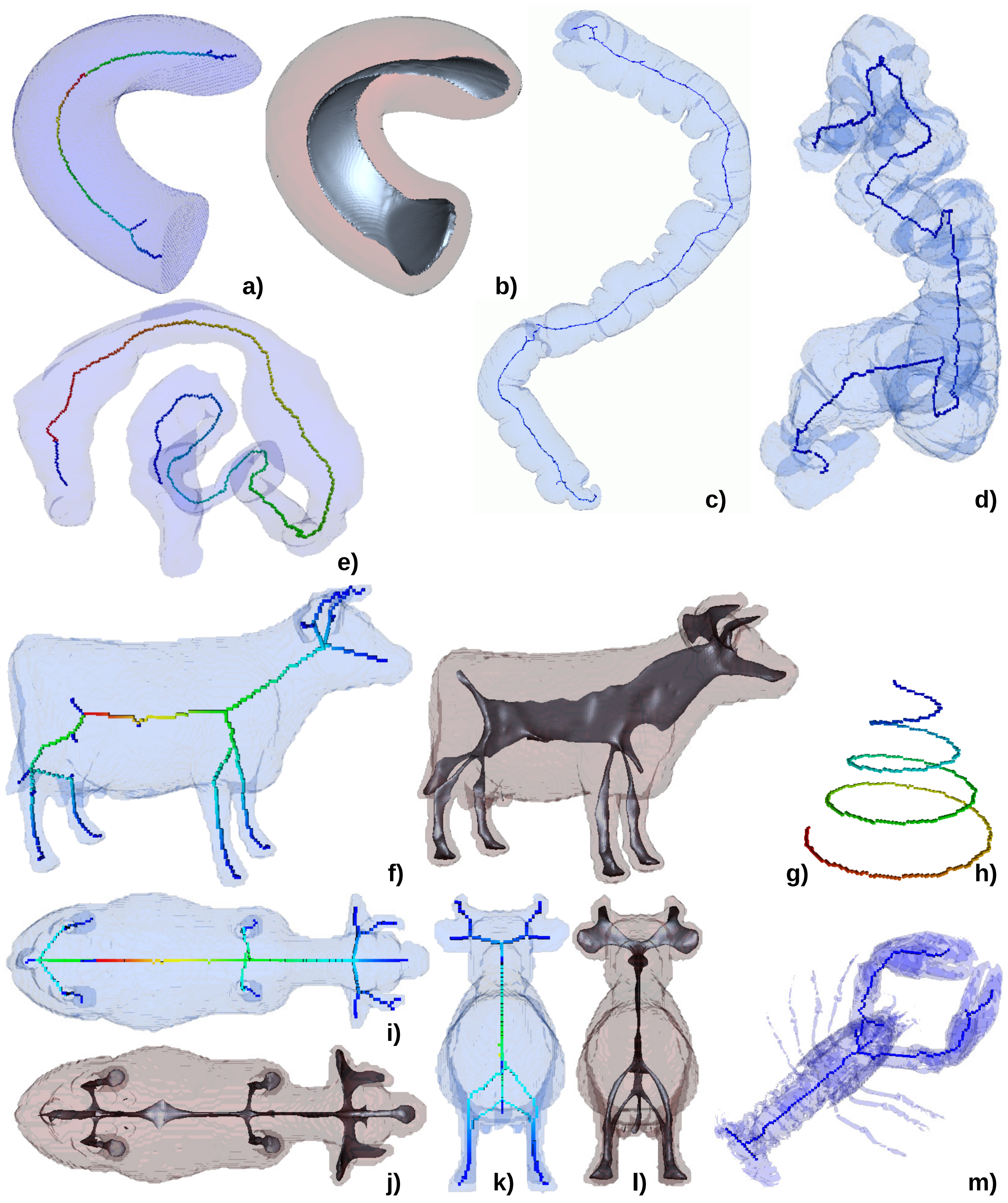

Figure 13: *Centerline examples: Simple object centerline and skeleton (a,b). Straightened and original colon (c,d). Frog duodenum (e). Spiral (h). Cow centerline and skeleton (f,g,i-l). Lobster (m)*