Fast Visualization of Plane-Like Structures in Voxel Data

Steffen Prohaska, Hans-Christian Hege

Department for Scientific Visualization, Zuse Institute Berlin (ZIB)*

Abstract

We present a robust, noise-resistant criterion characterizing plane-like skeletons in binary voxel objects. It is based on a distance map and the geodesic distance along the object's boundary. A parameter allows to control the noise sensitivity.

If needed, homotopy with the original object might be reconstructed in a second step, using an improved distance ordered thinning algorithm.

The skeleton is analyzed to create a geometric representation for rendering. Plane-like parts are transformed into an triangulated surface not enclosing a volume by a suitable triangulation scheme. The resulting surfaces have lower triangle count than those created with standard methods and tend to maintain the original geometry, even after simplification with a high decimation rate. Our algorithm allows to interactively render expressive images of complex 3D structures, emphasizing independently plane-like and rod-like structures.

The methods are applied for visualization of the microstructure of bone biopsies.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Viewing algorithms I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations I.4.7 [Image Processing and Computer Vision]: Feature Measurement—Feature repesentation I.4.7 [Image Processing and Computer Vision]: Scene Analysis—Shape J.3 [Life and Medical Science]: Health

Keywords: skeletonization, thinning, distance transform, triangulation, visualization

1 Introduction

High resolution 3D imaging technologies are getting more and more common in science. Huge amounts of data have to be analyzed and visualized in a concise way. For scalar data isosurface representations and volume rendering are standard techniques. However, for data with rich internal structure, like porous media, these methods are not always well suited (see e.g. Fig. 9). Isosurfaces are likely to consist of several millions of triangles and cannot be rendered interactively using standard techniques. Even if rendering performance is not a problem, the structure might be visually too complex to be depicted as an isosurface. Structures near the view point occlude those farther away and prevent to properly visualize the internal composition of the object.

This suggests to start with a shape analysis procedure, creating a representation of the object that is better suited for visualizing its internal structure and that may serve also as starting point for geometrical and topological analysis.

*Takustr. 7, 14195 Berlin-Dahlem, Germany, prohaska|hege@zib.de

IEEE Visualization 2002 Oct. 27 - Nov. 1, 2002, Boston, MA, USA
0-7803-7498-3/02/$17.00 © 2002 IEEE

We propose an algorithm that extracts parts of the medial surface from a binary 3D image representation of an object. This algorithm is based on a measure which is sensitive to plane-like structures. It does not preserve the topology of the oritinal object. If needed, homotopy with the original object can be reconstructed in a second step. The plane-like parts of the resulting voxel skeleton are triangulated for fast rendering. The rod-like parts are rendered as lines. The triangulation method creates surfaces not enclosing a volume. Practical experience shows that these surfaces can also be easily simplified. They tend to better preserve the original geometrical structure than isosurfaces, even with a low triangle count.

A natural byproduct of our algorithm is the local thickness of the object.

2 Skeletonization

2.1 Problem

Imaging devices provide a stack of 3D image data. These are transformed to a binary image by a segmentation procedure assigning every voxel either to the object or to the background. If the data are of good quality, a simple thresholding procedure will be fine. Otherwise some prefiltering or more elaborated algorithms may be used. The medial surface shall now be extracted. To define the skeleton, Blum proposed the grassfire analogy [2]: If a fire was lit at all boundary voxels and the fire propagated with constant speed towards the center of the object, there would be planes inside the object where the fire would stop due to missing voxels it could propagate to. These voxels inside the object define the medial surface. Another definition, widely used, is that the medial surface is located at the discontinuities of the derivative of the distance transform of the object. Not sticking to these definitions, basic properties of a skeleton should be:

- homotopy: the skeleton should preserve the topology of the original object
- invariance under isometric transformations: the skeleton of a rotated object should be the rotated skeleton
- reconstructability: the original object should be reconstructable from the skeleton
- thinness: the skeleton should be only one voxel thick.

In general, it is not possible to exactly fullfill all these requirements simultaneously. Perfect reconstructability, e.g., will cause skeletons to be very sensitive to noise on the surface. One voxel added at the surface might lead to large spurious side branches. Thinness is conflicting with reconstructability and rotational invariance in most cases, because one direction has to be favored to remove parts being two voxels thick.

There are various approaches to extract skeletons. Basically, four variants can be distinguished:

- simulation of the 'grassfire'

- analytic computation of the medial axis
- topological thinning
- medial surface extraction from a distance map.

Most relevant for voxel objects are the last two. Thinning algorithms [11] strictly keep the topology of the object (see [9] for a survey of digital topology), but have problems in finding the medial surface. To fix this problem, Pudney [15] proposed to combine thinning algorithms with a distance map guiding the process towards the medial surface. As a side-effect the algorithm is sped up crucially, compared to a naive implementation that visits every voxel again and again, until no more voxels can be removed. A second problem is how to locally identify the voxels that are not needed for homotopy, but should be kept for geometrical reasons, e.g. those representing endpoints of lines or edges of surfaces. Different local tests were devised, all afflicted with the problem of trying to detect a global feature in a local neighborhood. If only endpoints of lines are to be detected this is less problematic. If the algorithm shall stop at the edges of planes, local tests are not able to distinguish between overall important voxels and others. Side branches may result.

A distance map based approach has the advantage to consider the overall geometry of the object. Common to such algorithms is a parameter controlling thickness and size of the skeleton. But it is not easy to design algorithms that strictly maintain the object's topology. Homotopy might be re-established in a second step using the whole object or a topologically equivalent skeleton. Gagvani [7] presents a measure, based on the distance map in the neighborhood of each voxel, allowing to identify skeleton voxels. Malandain [13] uses a more global point of view, taking the nearest boundary voxels into account. Costa [5] introduces the idea of the geodesic distance along the surface of the object. We follow the last idea which we consider to take the most global view. Not only the nearest boundary voxels, but the shape of the whole surface is taken into account. A similar idea is used in [14] for Voronoi diagram based skeletonization in 2D.

The most promising approach seems to be a combination that takes advantage of the strengths of both approaches: some kind of a distance map to grasp the overall geometric location of the skeleton and some type of thinning procedure to guarantee homotopy. Also important is the possibility to control noise sensitivity by a tuneable parameter.

2.2 Algorithm

We now present an algorithm to compute a skeleton of a binary image. In the first subsection we define a measure that characterizes plane-like skeleton voxels. Thresholding this scalar measure we are able to select the most prominent plane-like structures. The result is a skeleton that in general is not homotopic to the original object.

In the second subsection we describe how a skeleton with the topology of the original object can be re-established by a distance ordered thinning algorithm. This step is not mandatory. If only detection of plane-like parts is necessary, the first subsection presents all you need.

At the end of this section we will have computed a voxel representation of the skeleton containing plane-like parts. Optionally, rod-like parts are added to keep the topology of the original object.

2.2.1 Characterization of Skeleton Voxels

We start with a characterization of skeleton voxels based on a distance map. We assume that for every voxel in the object the nearest boundary voxel ist known. This information can be approximately calculated with well known distance map algorithms (for a comprehensive overview see [6]). Only small modifications have to be made to keep the information about the nearest boundary voxel during propagation.

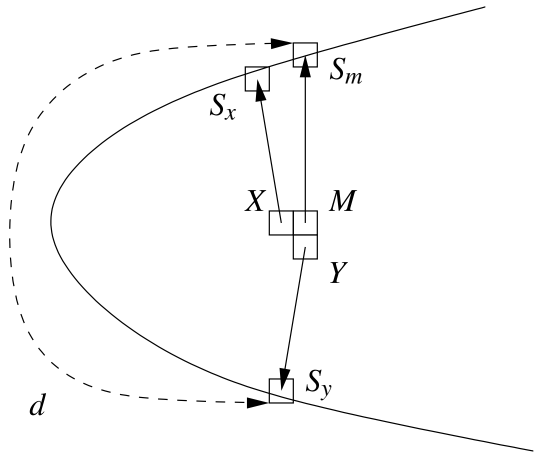

Figure 1: Definition of the measure d, detecting plane like structure in the skeleton. d is the maximum of the geodesic distances along the object's boundary between the surface voxels (S_m, S_x, S_y, S_z).

Now we calculate a measure d for a voxel M at position (i, j, k) in object O as follows (see Fig. 1):

- Look for neighbors in all three directions towards lower coordinates. In our example these are X at $(i-1, j, k)$ and Y at $(i, j-1, k)$ and Z (not depicted in the figure) at $(i, j, k-1)$.
- Identify the nearest surface voxels (S_m, S_x, S_y, S_z) for voxel M and its neighboring voxels X, Y, Z.
- Calculate the geodesic distances $(\overline{S_m S_x}, \overline{S_m S_y}, \overline{S_m S_z})$ along the boundary of object O.
- Take the maximum of these distances, here: $d = \overline{S_m S_y}$.

The geodesic distance can be calculated in a straight forward way by propagating in the set of the boundary voxels, starting from a seedpoint (see [17] for a fast algorithm). A skeleton is extracted by choosing a threshold t and selecting all voxels with $d > t$.

Except for the following small modifications this is the measure proposed in [5]. We take neighbors only with lower coordinates because we want to end up with an one voxel thick skeleton. This requirement contradicts symmetry considerations in most cases. For example a plane that is an even number of voxels thick would lead to a skeleton that is two voxels thick if symmetry were preserved. Therefore we have to break symmetry and favor one direction to end with a one voxel thick plane.

Suppressing errors due to noisy boundaries requires a minimum value of the threshold t. We find it useful to set t to at least 5 voxels. With larger t only the more important parts of the skeleton survive. The exact value to choose depends on the object and on the features to be extracted. A practical way to deal with this fact is to inspect only a part of the object and interactively vary the parameter. A useful value can then easily be chosen. In Fig. 2 skeletons for different thresholds are depicted. The steps needed to render the images are described below. Note that we can separate plane-like from thin rod-like structures and render them as surfaces and tubes, respectively. For higher values of t in these examples nearly all plane-like structures vanish and only loops remain. Topology is not preserved by the plane-like structures alone at any of the chosen thresholds.

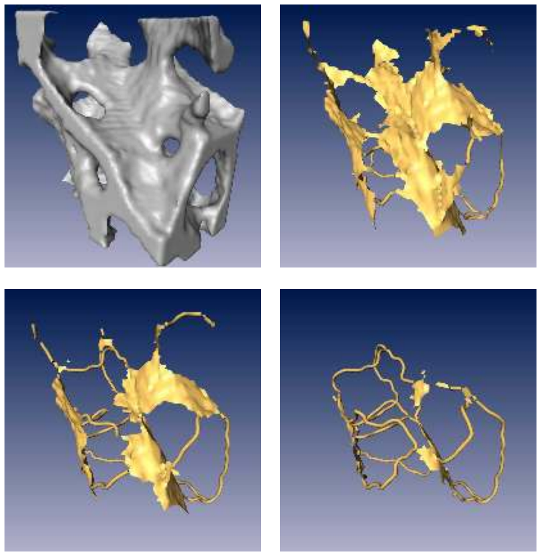

Figure 2: Skeletons with different thresholds t (measured in voxels). Structures that are detected as plane-like are rendered as a surface. Structures detected as rod-like are rendered as tubes. Top left: Isosurface, top right: $t = 5$, bottom left: $t = 10$, bottom right: $t = 20$.

In general, topology is conserved only for certain configurations. Cavities e.g. will be kept because there is no way from the boundary enclosing the cavity to any other boundary, which means $d \to \infty$ for some parts of the skeleton. But for other configurations topology might dramatically change.

Even though we favored one direction in the calculation of d, the such computed skeleton still contains voxels that are not needed for a unit-wide skeleton. They can be removed in a postprocessing step by a method described in [3].

Our measure is constructed to be sensitive to plane-like structures. Thin large planes can be detected because the geodesic distance encodes the extension of the plane additionally to the thickness. Measures using the euclidian distance [13] or only a local neighborhood [7] fail in such situations and will disconnect thin planes. Another related property is the following: if voxels close to edges of a plane-like structure, e.g. a disk, are considered to be skeleton voxels, then voxels located closer to the center of the disk will be considered even more important for the skeleton due to the growing geodesic distance (see Fig. 1: M is more important than X). This leads to a nice disk-like skeleton with smooth edges and no holes. Measures focusing on other properties like reconstructability [7] do not have this property. Voxels added at the boundary may increase the geodesic distance. As long as the increase is significantly smaller than the threshold, this 'noise' will be ignored.

Note that the measure is not useful for detection of rod-like structures. Long thin rods might be considered completely as 'noise' and will be deleted. If they are not closed to a loop they are topologically irrelevant and will not show up in topological reconstruction as described in the next subsection. Thick structures might be detected if their circumference is larger than $2t$. But the focus of our algorithm is on plane-like structures.

2.2.2 Homotopy

The skeletons constructed up to now are not guaranteed to be homotopic to the original object. If preservation of topology is needed, it can be assured in a second step. Note that strict homotopy with the original object will force significant changes in the skeleton if noise creates small loops on the boundary of the object. These loops have to be kept and may introduce large spurious side branches (see Fig. 8). Topology should be reconstructed only for objects with smooth boundaries. The spirit of our algorithm is close to the ideas presented in [13] and [4]. Assume we have a skeleton S and the corresponding object O. In the process of topological reconstruction [13] parts of the object are added to the skeleton until it is homotopic to the original object. These parts should be located at the center of the object and only voxels should be added that are really needed. One may start with all voxels of the object and remove those not needed for homotopy or belonging to S. The order of the removal should be chosen such that voxels at the center are left over.

We follow the second point of view and apply a distance ordered thinning algorithm [15]. During the thinning process all simple voxels except those belonging to S will be removed. A summary of the algorithm is given at the end of the subsection. The curious reader might have a look at it before proceeding to the details presented in the next paragraphs.

The connectivity of the object is chosen to be 26-connected. In order to have well defined topological properties the background is to be 6-connected [9]. The basic idea is to propagate a distance map starting at the boundary voxels of the object. Every voxel visited is tested for simplicity and removed if deletion does not break homotopy [15]. To get a good localization in the center we propose some modifications to this basic algorithm.

We do not remove voxels only sequentially, but first visit all voxels at a specific distance and mark voxels that will potentially be removed. After this step we revisit all marked voxels. If they are still simple in the new configuration after removal of earlier visited marked voxels, we remove them. Tests for simplicity therefore only depend on the configuration previous to the iteration. But removal of two simple voxels in parallel can change topology which requires the sequential rechecking. The advantage of checking in parallel is a more uniform erosion of the boundary.

To improve this uniform erosion even more, we propose to introduce a slowdown factor. Each distance will be visited more than once before advancing to the next and in every iteration all distances will be processed again up to the actual maximal value. Voxels tested to be non-simple at some time may get simple in the future if the neighborhood changes. Voxels should therefore be visited again to remove them and avoid spurious branches. The ideal case would be to remove all voxels at one distance first before proceeding to the next. But this is not possible in all configurations. Removal of voxels with higher distance value sometimes changes the configuration in a way allowing removal of voxels with a lower distance value. We will first process every distance several times to delete as many voxels as possible. But we cannot do this job completely and at some point we have to proceed to higher distance values. The slow down factor controls when. It allows to trade speed against accuracy.

But one major problem still exists. The distance transform is calculated in three dimensions and guides the process towards the medial surfaces of the object. However, in the center there may exist large, two voxel thick surfaces for which all voxels have the same distance value. If the object got thinned to such a surface, the order of the checks in the plane would be arbitrary due to the fact that all these voxels would get processed in parallel and all of the voxels would be simple before a voxel is removed. The sequential removal would then delete them in an arbitrary order. Localization in the middle (2D) of the plane is not guaranteed. To assure central lines, we have to devise a step that first reduces the two voxel thick plane to a voxel thick one, before starting to erode this plane from the edges. This is achieved by introducing two subiterations. In each subiteration only 13 boundary voxels to the top, right, back (grey voxels in Fig. 3, right) resp. front, left, down (black voxels)

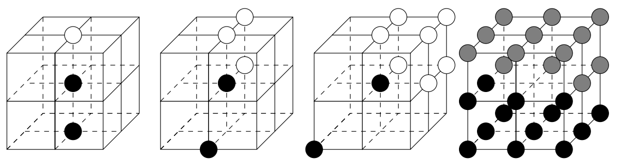
Figure 3: Masks to define two subiterations. From left to right: Masks to detect face, edge and corner configurations (black circles depict object voxels, white circles depict background voxels). Right: A voxel gets processed in subiteration 1/2 if the object voxel not in the center is located at a grey/black position.

are checked. The detection of these voxels is done by the face, edge, and corner masks of Fig. 3 and rotated versions. The voxel in the middle will be marked for checking, if the black voxels are part of the object and the white voxels are part of the background. Voxels left empty do not influence the test.

To summarize our algorithm:

- Calculate the chamfer distance map of the object.
- Create a queue for every distance value and insert all voxels with this distance not belonging to the initial skeleton S.
- Choose an even slowdown factor s, e.g. 6.
- Start with iteration $i = 0$.
- Process all queues up to distance $max = i/s$:
  - In odd resp. even iterations process boundary voxels of subiteration 1 resp. 2.
  - Mark all voxels that are simple without removing them.
  - Remove marked voxels sequentially if they are still simple, when tested against the remaining voxels.
  - All voxels that are not removed, not marked or that do not belong to the subiteration are queued again.
- Increment iteration count i and repeat until the computed distance is larger than the maximal distance and no voxel is changed.

2.3 Implementation Details

The most time consuming part is the calculation of the geodesic distances. For every voxel in the object a distance map in the boundary voxels has to be calculated. The number of voxels is N. In case of plane-like objects with thickness w the number of boundary voxels is $2N/w$, means $O(N)$ boundary voxels. Calculation of a distance map in an arbitrary set of voxels with the algorithm presented in [17] visits every voxel once. Runtime of every geodesic distance map is therefore $O(N)$. The overall runtime would be $O(N^2)$ which is problematic for large N.

But we do not need the whole geodesic distance map. The threshold t provides an upper limit. Therefore, we never propagate further than t which might touch $O(t^2) \ll O(N)$ voxels. t^2 might be rather large, but in regards of N it is $O(1)$. Thus the complexity reduces to $O(N)$. Practically, due to the large constant t^2, implementation should be done with great care.

Calculating the same distance map twice can be avoided. More than one voxel inside the object might have the same nearest boundary voxel and therefore needs the same distance map. To do this, we first invert the relation between object voxels and boundary voxels. We build a map for every boundary voxel that points to all object voxels that have it as the nearest boundary voxel. We then run through all boundary voxels and initialize a new distance map with the actual boundary voxel as a seed point. Next we run through all object voxels that the map points to and calculate the needed geodesic distances. We propagate the distance map only as far as needed. In most cases this is smaller than t.

There are many pitfalls along the way: The distance map algorithm starts with a cleared buffer. Naively setting the whole buffer has runtime $O(N)$ and would therefore break our previous discussion. To minimize the memory to be cleared we store the upper and lower index of the memory visited during the last propagation and only clear parts of memory in-between. With our stop criteria only small parts are needed compared to N, kicking out the factor N that would have to be payed.

The objects dealt with often fill up only a small percentage of the overall volume. In these cases it might be useful to introduce some kind of indexing scheme to get arrays storing only the object voxels and leaving out the background. The amount of memory is reduced which also might save time during initialization.

The basic thinning algorithm has complexity $O(N)$ like a distance map. Our modification to visit every voxel again and again changes this behaviour. Worst case is to visit every voxel N times (one voxel thin line tagged at one side together with the worst visiting order). Practically the number of iterations equals the product of thickness and slowdown factor. Some additional constant due to long side branches might be added. Overall this is much smaller than N and we have a runtime $O(N)$ in most of the cases. Efficiency of the implementation is mostly limited by the local test for simplicity. In [1] a boolean characterization is given which might be implemented straight forward. In real applications a look-up-table or binary decision diagrams [16] should be built.

3 Geometric Primitives, Rendering

At this point we got a skeleton consisting of lines and surfaces represented by voxels. This section describes how to convert the voxel representation for rendering. The plane-like parts will be transformed into a triangulated surface, the rod-like parts will be represented and rendered as lines.

Triangulating a voxel surface is different from standard ways of converting a voxel representation to a polygonal representation. We do not want to create a surface separating inside from outside, as is done in isosurface algorithms like e.g. the marching cubes algorithm [12] (for a detailed discussion from a digital topological point of view see [10]).

Our goal is to construct a surface not enclosing any volume, except for cavities in the original object. It has no inside and outside. Therefore it will be rendered two-sided. The surface should have the same connectivity as the skeleton. The vertices will be located at the centers of the voxels except for vertices that have to be added in the triangulation process. But not all of the connections can be represented properly by a surface. For rod-like structures tubes or one dimensional lines will be more suited.

In a first step, edges between voxel centers are constructed that represent the connectivity of the voxels. In a second step, edges being part of a plane-like structure will be connected by triangles. The remaining edges can be displayed as lines or tubes.

Similar to the marching cubes algorithm we try to break the problem down to an unit lattice cell which has a voxel at every of its corners. Special care has to be taken to assure continuous transitions from one cell to the next. Each of these unit lattice cells can be represented by one of its corners (here the lower, left, front corner). To avoid ambiguities we demand that every point in space should be member of only one cell which can be achieved by favoring one direction: We define a *reduced cell* as a cell including edges and faces (without their edges) adjacent to the representative corner and excluding all other faces, edges and corners. A cell including all

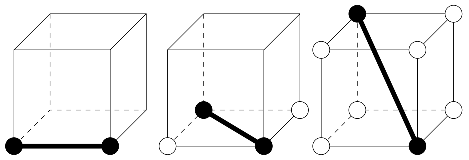

Figure 4: Cell configurations for 6-, 18-, 26-connections. Black circles depict object voxels, white circles depict background voxels.

faces and edges will be called *full cell*. With this definition every point of space unequivocally belongs to one of the reduced cells and the reduced cells fill up whole space. If the algorithm only creates primitives in the reduced cell no points in space will be touched twice. But to get connectivity, primitives have to be stiched together at the boundaries. This has to be done with care to avoid coplanar triangles.

3.1 Connectivity Graph

Two voxels can only be 26-connected if they are located in the same cell.

In every reduced cell the connectivity can now be represented by lines, called *connections*. A connection is added for every voxel that is 6-connected to the representative voxel. 18-connections are introduced if they are needed to keep connectivity on one face of the cell and 26-connections only if no other connection links the two voxels. See Fig. 4 for the possible configurations. There are three possible 6-connections, six possible 18-connections on the faces to the front, left and bottom (the other faces belong to another reduced cell) and four possible 26-connections. The result after applying this procedure to every reduced cell is a graph representing the skeleton's connectivity.

The connections are introduced in a way that 6-connections along an edge are determined by the voxels neighboring the edge, 18-connections lying in a face are determined by the four corner voxels of the face. But these voxels are completely shared with a neighboring reduced cell. The configuration is independent of the cell and therefore the connections too. Consequently it is possible to construct the correct 6- and 18-connections for all edges and faces of a full cell. This property guarantees a continuous transition to the neighboring cells.

3.2 Triangles

As described, we can construct all connections in a full cell. In the next step a surface will be generated inside every reduced cell having the described connections as edges. These edges are shared between full cells which will guarantee a connected surface in the end. We now going to describe a heuristic to construct triangles in these cells. To avoid ambiguities we construct the surface only inside a reduced cell (not a full cell), i.e. one has to take care of triangles that were to be constructed on the face of a cell. A detailed inspection of our proposed triangulation reveals few cases in which our heuristic fails or creates unnatural triangulations. In these cases a triangulation will be manually specified.

The construction of the triangles is as follows:

- Check for special case (see below). If no special case is detected perform the following steps:
- Construct all connections in the full cell.
- Detect the smallest loop of connections, triangulate it and mark all connections in that loop as visited.
- Repeat detection of the smallest loop that contains an unvisited connection and triangulate that loop until all connections are visited, or no loop is detected.

During triangulation of the loops creation of triangles on faces of the cell not belonging to the reduced cell has to be avoided. Otherwise ambiguities might arise. Note that there are no loops of length three belonging to the faces, because those would include two 6-connections that forbid the needed 18-connection. Therefore, only loops of length four can be located totally on a face, but these are identical to the face. Hence, we will accept only total faces belonging to the reduced cell. For all other loops a check is done whether they would create a triangle belonging to a face. If this is the case, the center of gravity of the vertices of the loop will be added and triangulation will be done by connecting all vertices to the center of gravity. In all other cases the triangulation is done by selecting an arbitrary vertex and connecting it to all other vertices.

The triangulation of a reduced cell is fully determined by the configuration of the eight corners and a corresponding lookup table can be build for the 256 cases. An inspection of all cases reveals one configuration (modulo 8 rotated versions) where the described procedure creates coplanar triangles. Have a look at the leftmost column of Fig. 5. To visit all edges, two loops are needed. If they were $2,6,8,3$ and $2,6,5,3$ the triangle $2,6,3$ would be created twice. This has to be fixed by storing the triangulation depicted in the lower row in the lookup table, overriding the automatic generation.

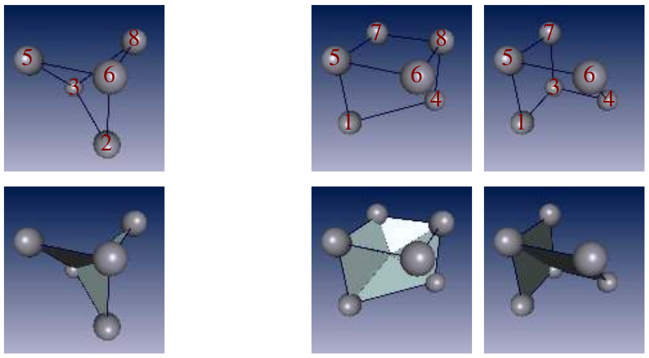

Figure 5: Cell configurations that need special triangulation. The top row shows the edges the bottom row the triangulations. Left and right: Triangulation is optimized as depicted. Middle: The top face does not belong to the reduced cell, triangulation cannot be optimized.

In some other cases a correct triangulation might be manually optimized to reduce the number of triangles and to improve the visual quality. The two rightmost columns of Fig. 5 show such a case. The top two edge configurations could be transformed into each other by a simple rotation. But symmetry is broken by our definition of the reduced cell: the top face does not belong to the reduced cell and therefore in the left case only one loop is accepted to create triangles. The loop $5,6,8,7$ is left untriangulated. In the rotated version to the right this loop is transformed to $5,1,3,7$ and is accepted. If we triangulated the second loop $5,6,4,3,7$ similar to the loop in the right case the result would not look very nice. Too many triangles would be involved and they would be only connected at the edges of the cell. The triangulation depicted at the bottom right looks much more natural. We insert this hand crafted triangulation in the lookup table, overriding the automatic version. We are not allowed to change the left case because it is unclear whether the configuration in the adjacent reduced cell to the top would create our hand crafted triangles. In this case we are stuck with the automatically generated version.

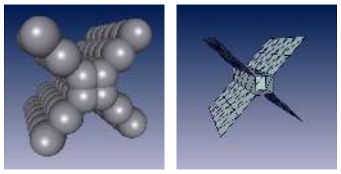

Figure 6: A junction configuration that causes problems. The triangulation introduces artificial cavities as seen in the right image.

The described triangulation is applied to every reduced cell to create the triangulation of the skeleton.

3.3 Remarks

Not all connections will be included in the surface created. Connections not included represent rod-like structures in the skeleton and cannot be visualized properly by a surface. They should be treated separately, e.g. rendered as lines or tubes.

Some junctions may lead to a surface that is locally thicker than one voxel (see Fig. 6). The triangulation created for these configurations will introduce many artificial cavities. This is an open problem to be solved. We think that an inspection of a larger neighborhood might do the job and might also be a way to get rid of the asymmetry introduced by the reduced cell. We will further investigated this idea in the future.

Tests show that the number of triangles in the skeleton surface is only about 15% of that in the corresponding isosurface. This alone entails a noticable higher framerate in interactive rendering. The application of surface simplification algorithms, e.g. [8], might reduce the number of triangles further. Due to the smaller starting number this process is much faster for the skeleton than for the isosurface. In case of a very low number of triangles, the skeleton tends to keep the geometric location of the planes whereas the isosurface tends to keep the included volume. In such cases the skeleton is more useful for visualizing the geometric structure (c.f. Fig. 7).

4 Application

The algorithms presented were integrated in the Amira framework [19, 18]. We applied them to a test object with various levels of noise added (Fig. 8). The plane-like parts are very stable whereas the rod-like parts may change dramatically due to loops introduced by the noise. A real world example is the visualization of bone biopsies (see Fig. 9) scanned with a micro-CT. We used this example to do some timings as described in the figure caption. The algorithm runs in acceptable time and rendering can be done at interactive framerates. The number of triangles is noticeable lower compared to the isosurface and simplification of the skeleton surface is easily possible. A goal of this application is to find measures that quantify the structural composition of the bone. The extracted skeleton surface is useful for the visualization of local properties of the network, e.g. the local thickness. These properties can easily be color-coded onto the surface. We also want to use the skeleton as a starting point for structural analysis. First steps for quantifying the structure based on the skeletonization are done. This will be further developed in the future.

5 Results and Future Work

We presented a noise-insensitive algorithm to calculate skeletons of complex voxel objects. The algorithm has a parameter that allows to control the size of plane-like structures being represented by the skeleton. This allows to suppress noise or to focus on the most prominent planes. Optionally, topology can be reconstructed in a second step. The skeleton is split in plane-like and rod-like structures. The plane-like structures are triangulated, the rod-like parts are transformed to lines. The amount of triangles necessary to render the structure is reduced compared to common isosurface algorithms and can be further reduced through surface simplification methods. The created surface allows for expressive visualizations and can serve as a base for depicting local properties, e.g. by coloring it. Rod-like parts are rendered as lines. Plane-like and rod-like structures can be independently emphasized.

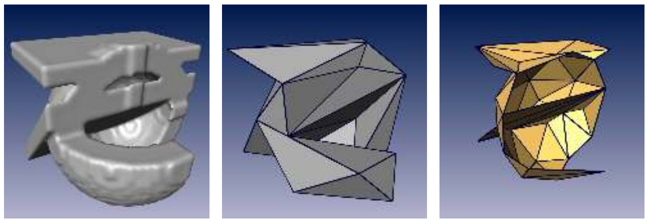

Figure 7: Comparison of the high resolution isosurface consisting of $\approx 50,000$ triangles (left). On the right are simplified versions of the isosurface (grey) and of the skeleton (yellow) with 50 triangles each. The geometric structure can be better comprehended by the simplified skeleton.

Some problems regarding the triangulating of junctions remain and will be tackled in the future. Inspection of a larger neighborhood might solve this problem. It might also allow to remove the asymmetry introduced by our definition of the reduced cell. Up to now we represented plane-like and rod-like structures in two distinct data structures. Therefore surface simplification does not take into account the connections maintained by the lines. For typical rendering purposes this is not a problem, but for a topological analysis a unified representation is necessary. It would also be nice to include not only planes and lines but also volumetric parts in this data structure. These parts could be created by well known triangulation algorithms if a measure were available to classify voxels as volume-, plane- and rod-like. Another interesting point might be to avoid the triangulation and to use other rendering algorithms directly on the voxel skeleton.

Acknowledgements

This work is performed in frames of the ESA-MAP project AO-99-030.

We thank Malte Westerhoff for helpful discussions. Image data are a courtesy of Wolfgang Gowin and Michael Giehl (Center of Muscle and Bone Research, Dept. of Radiology, Univ. Hospital B. Franklin, Free University Berlin).

References

[1] Gilles Bertrand. A boolean characterization of three-dimensional simple points. *Pattern Recognition Letters*, 17:115–124, 1996.

[2] Harry Blum. A transformation for extracting new descriptors of shape. In W. Walthen-Dunn, editor, *Models for the Perception of Speech and visual Form*, pages 362–380. MIT Press, Cambridge, MA, 1967.

[3] Gunilla Borgefors, Ingela Nyström, and Gabriella Sanniti di Baja. Computing skeletons in three dimensions. *Pattern Recognition*, 32:1225–1236, 1999.

[4] Sylvain Bouix and Kaleem Siddiqi. Divergence-based medial surfaces. In *ECCV'2000*, volume 1842 of *Lecture Notes in Compute Science*, page 603, Dublin, Ireland, June 2000.

[5] Luciano da F. Costa. Multidimensional scale space shape analysis. In *IWSNHC3DI'99*, pages 214–217, Santorini, Greece, 1999.

[6] Olivier Cuisenaire. *Distance Transformations: Fast Algorithms and Applications to Medical Image Processing*. PhD thesis, Université cotholique de Louvain, Laboratoire de Telecommunications et Teledetection, 1999. http://ltswww.epfl.ch/∼cuisenai/papers/.

[7] Nikhil Gagvani. Paramter-controlled volume thinning. *Graphical Models and Image Processing*, 61(3):149–164, 1999.

[8] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. *Computer Graphics*, 31(Annual Conference Series):209–216, 1997.

[9] T. Yung Kong and Azriel Rosenfeld. Digital topology: Introduction and survey. *Computer Vision, Graphics, and Image Processing*, 48(3):357–393, December 1989.

[10] Jacques-Olivier Lachaud. Continuous analogs of digital boundaries: A topological approach to iso-surfaces. *Graphical Models*, 62:129–164, 2000.

[11] Louisa Lam, Seong-Whan Lee, and Ching Y. Suen. Thinning methodologies — A comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(9):869, 1992.

[12] William E. Lorensen and Harvey E. Cline. Marching cubes: a high resolution 3D surface construction algorithm. In M. C. Stone, editor, *SIGGRAPH '87 Conference Proceedings (Anaheim, CA, July 27–31, 1987)*, pages 163–170. Computer Graphics, Volume 21, Number 4, July 1987.

[13] Grégoire Malandain and Sara Fernández-Vidal. Euclidean skeletons. *Image and Vision Computing*, 16:317–327, 1998.

[14] Robert L. Ogniewicz and Olaf Kübler. Hierachic Voronoi skeletons. *Pattern Recognition*, 28(3):343–359, 1995.

[15] Chris Pudney. Distance-ordered homotopic thinning: A skeletonization algorithm for 3D digital images. *Computer Vision and Image Understanding*, 72:404–413, 1998.

[16] Luc Robert and Grégoire Malandain. Fast binary image processing using binary decision diagrams. *Computer Vision and Image Understanding*, 72(1):1–9, 1998.

[17] Ben J. H. Verwer, Piet W. Verbeek, and Simon T. Dekker. An efficient uniform cost algorithm applied to distance transforms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11:425, 1989.

[18] Zuse Institute Berlin (ZIB) and Indeed - Visual Concepts, Berlin. *Amira 2.3 – Programmer's Guide*, August 2001. http://amira.zib.de.

[19] Zuse Institute Berlin (ZIB) and Indeed - Visual Concepts, Berlin. *Amira 2.3 – User's Guide and Reference Manual*, August 2001. http://amira.zib.de.

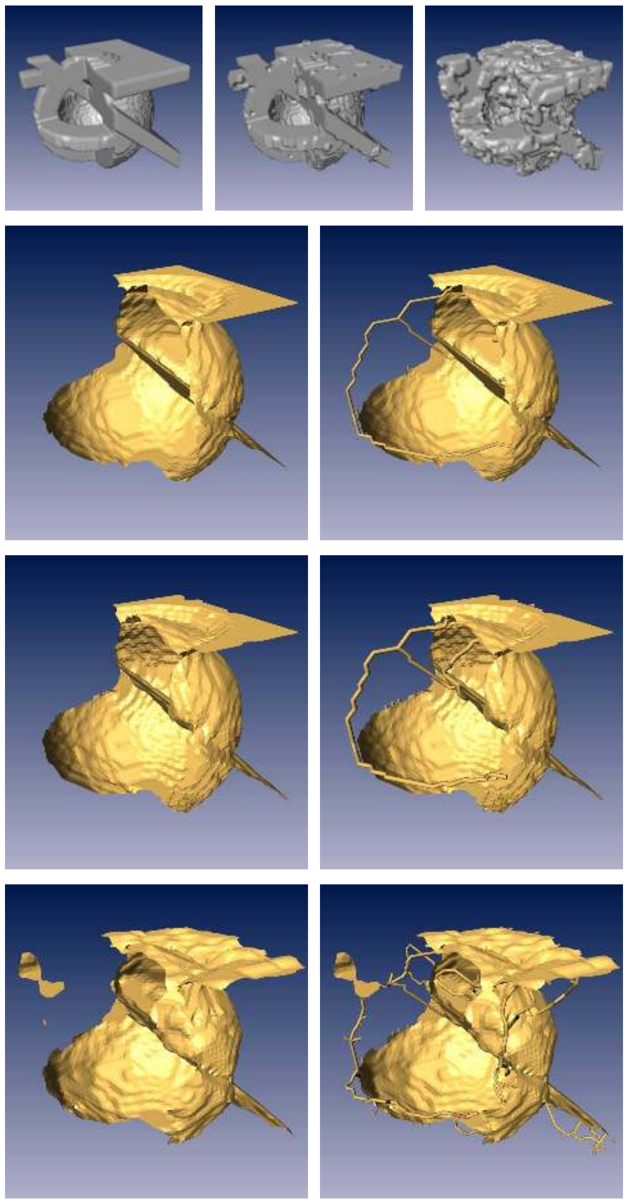

Figure 8: Test object of size 64^3 with noise added. Top row: grey isosurface without noise, with some noise, and with strong noise added. Next rows: plane-like skeleton (left), homotopic skeleton (right), without/with some/with strong noise (2nd/3rd/4th row). The plane-like part of the skeleton is rather insensitive to noise. Some small disconnected surfaces emerge in the bottom example caused by increased thickness of the object due to noise. Thicker parts are voted more important by the geodesic distance. Noise might lead to changes in topology. These changes have significant effects on the homotopic skeleton: New loops show up and change the rod-like parts completely – an intrinsic concomitant of topology preservation.

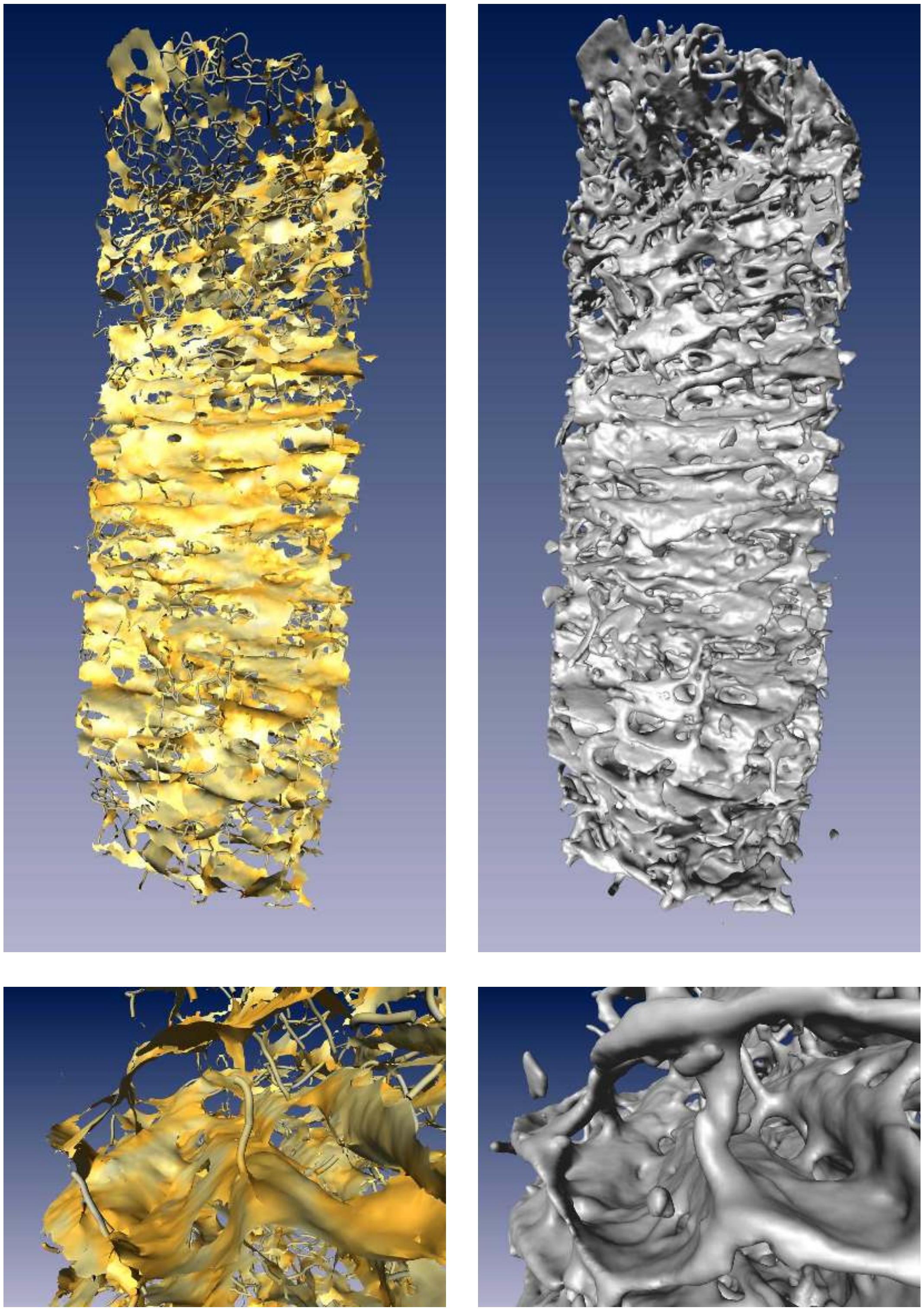

Figure 9: Bone biopsy (7 mm width and 18 mm length). The data set consists of $350 \times 285 \times 730$ voxels, 5% belonging to the object . Left: triangulated skeleton (250,000 triangles), local thickness color coded as saturation. Right: Isosurface (5,800,000 triangles), extracted with a marching cubes algorithm in 86 s. Accurate simplification including global topology checks of this surface to 250,000 triangles took 17 min. Extraction of the skeleton took 438 s (geodesic distance computation: 112 s, postprocessing: 6s, triangulation: 230 s, simplification 90 s) for the plane-like part and additional 90 s for the thinning procedure. The full resolution isosurface is rendered with 0.3 fps. The high resolution skeleton (960,000 triangles) renders with 1.8 fps and the simplified version with 6.5 fps; rendering of lines decreases speed to 4.6 fps. All timings were done on a SGI Onyx 3400 with R14000 processor (500MHz) and an IR3 pipe with 4 raster managers. The rendering of the skeleton clearly shows the more rod-like structures at the top, whereas the middle part is dominated by planes ranging from left to right. Note, how loops are depicted by lines. The focus images in the second row also clearly depict the various structural elements.