AN EXTENDED CONCEPT OF VOXEL NEIGHBORHOODS FOR CORRECT THINNING IN MESH SEGMENTATION

David Brunner, Guido Brunnett
Chemnitz University of Technology, Germany
{brunner, brunnett}@informatik.tu-chemnitz.de

ABSTRACT

We present a method for mesh segmentation based on volume representation and the so called skeleton graph. In this method, the triangle mesh is first trans-formed into a voxel representation. A thinning algorithm based on our definition of a local neighborhood is employed to extract a voxel skeleton which is then transformed into a graph representation. The branching points of this graph are used to define the segmentation of the mesh.
This article focuses on the data structure used to manage the voxel set and the extension of the neighborhood concept that allows to distinguish be-tween local and global neighborhoods. Our rasteriza-tion is not based on equilateral cells, leading to an efficient data structure in terms of memory usage.

Keywords

Thinning, Mesh Segmentation, Skeleton Graph, Volume Generation.

1. INTRODUCTION

In recent years, there have been many attempts to improve the segmentation of surfaces, since it is the starting point for numerous applications, including the recognition, classification, and matching of objects or geometric reconstructions [9], [12]. Segmentations can be calculated on the basis of various object features, such as curvature [3], stereoscopic image pairs, and topological structures. The examination of topological and geometric features has led to a range of new techniques and data structures, including shock graphs [9], Reeb graphs [6], level set methods [10], dynamical systems [5], and medial axes [7].
To describe the basic ideas of our method we consider a closed 2D curve C and its interior medial axis A, i.e. the set of points in the interior of C with non-unique nearest neighbors on C. If C is smooth any branching of A indicates that the shape of C branches into features that should possibly be separated by a segmentation method. Our method generalizes this idea to the segmentation of closed 3D-polygons by defining a graph whose branching points are used to define the segments of the surface. This graph is computed in a two-step procedure from a voxel representation of the mesh.
In a first step the voxel representation is reduced to a *voxel skeleton* by the use of a thinning algorithm. During the thinning process, it must be ensured that the resulting voxel set is topologically equivalent to the original object. To guarantee this, we employ the criterion proposed by Bertrand und Malandain in [8] for classifying which voxels may be thinned within an iteration. In order to achieve adequate smoothness of the voxel skeleton for our purposes, we supplement our thinning method with an additional criterion, which was proposed by Tsao and Fu in [11] and ensures uniform thinning.
In a second step we transform the voxel skeleton into a graph structure which allows the application of algorithms from graph theory. To simplify this graph we introduce a cost function to find and remove redundant junctions through merging of knots. The junction knots of the resulting graph are used to define the segmentation.
This approach has been presented by the authors in [4]. The results obtained have indicated that segmentation faults occur due to the fact that the initial voxel approximation may have a different topology than the object represented by the mesh. Since the thinning process does not change the topology the resulting graph representation of the skeleton is incorrect. To avoid this problem it is necessary that the thinning process not only considers the topology of voxels but also the behavior of the mesh contained by neighboring voxels.
In this paper we present a new concept of voxel neighborhood that distinguishes between local and global voxel neighbors. Local neighbors are characterized by the existence of a path on the mesh that joins the corresponding vertex sets and does not exceed a prescribed length. These basic concepts are introduced in section 2. In section 3 we report on how the new neighborhood concept is integrated in our data structure. Here we also describe how voxels that contain a part of the mesh are determined and how local neighbors are computed efficiently. For the latter we show that voxels that contain multiple disconnected segments of the mesh have to be treated in a preprocess.
In section 4 we describe how to modify our segmentation algorithm according to the neighborhood concept. Finally, results are given in section 5.

2. BASIC DEFINITIONS

In this section, we introduce our notation. We assume that the shape S of each object dealt with is represented as a consistent, closed triangle mesh $M(S)$.
The first step of our method consists of the transformation of the mesh $M(S)$ and its interior into a volume representation. For this purpose, 3D grids of equally-sized volume elements (cubes) are often used. If $v\,(r, c, b)$ is used to denote a voxel at position r, c, b

(rows, columns and bands) in the grid, the voxel set $V(S)$ can be described as

$$V(S) = \{v\,(r, c, b) \mid 0 \le r < R, 0 \le c < C, 0 \le b < B\},$$

where R, C and B denote the maximum number of rows, columns and bands.

Each voxel of $V(S)$ is assigned the value 1 if it belongs to the object (foreground) or 0 if not (background). We denote the foreground set and the background set $V(S)_f$ and $V(S)_b$. The complementary set is denoted with a superscript c, e.g.: $(V(S)_f)^c = V(S)_b$.

In order to classify border voxels, we say that $v \in V(S)_f$ is a directed border voxel of direction $r+$, $r-$, $c+$, $c-$, $b+$, $b-$ if the voxel adjacent to v in the indicated direction belongs to $V(S)_b$. In other words, $v(r, c, b)$ is a border voxel of direction $c+$, if $v(r, c, b) \in V(S)_f$ and $v(r, c+1, b) \in V(S)_b$.

Thinning algorithms exploit the concept of neighborhood. Two voxels v and w are said to be 6-adjacent if they share a face, 18-adjacent if they share a face or an edge, and 26-adjacent if they share a face, or an edge, or a vertex. The sets of all voxels which are α-adjacent to v are denoted $N_\alpha(v)$ ($\alpha \in \{6, 18, 26\}$).

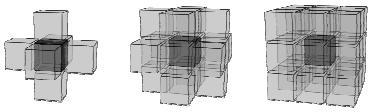

Figure 1: From left to right the neighborhood $N_6(v)$, $N_{18}(v)$, and $N_{26}(v)$ is shown for a single voxel v.

The voxel set $V(S)$ is considered α-connected if, for any two voxels v, w of $V(S)$, a series of voxels v_1, ..., v_m of $V(S)$ exists such that $v_1 = v$, $v_m = w$ and $v_{i+1} \in N_\alpha(v_i)$ for $1 \le i < m$ and $\alpha \in \{6, 18, 26\}$. In order to avoid connectivity paradoxes (see [7]) we set $\alpha = 26$ for foreground and $\alpha = 6$ for background connectivity.

To formulate "smoothing conditions" on the thinning process the concept of *checking planes* has been introduced be Tsao and Fu [11]. A checking plane, CP_r, CP_c, CP_b is defined in a similar manner to neighborhood sets, with the difference that one coordinate r, c, or b is constant. E.g.:

$$CP_r(v): \{w \in V(S)_f \mid |v_r - w_r| = 0 \text{ and } \max(\{|v_c - w_c|, |v_b - w_b|\}) = 1\}.$$

To avoid segmentation faults as described in section 1 we extend the neighborhood concept in order to distinguish between *local* and *global* voxel neighbors. Therefore we need to examine all voxels which contain a part of the mesh. With such *mesh voxels* we associate the vertices enclosed by the voxel and in addition all vertices of mesh triangles that are partially contained. Let ε be a prescribed distance and let a *path* on the mesh be a sequence of vertices v_0, v_1, ..., v_n such that v_i and v_{i+1} are adjacent for $i = 0$, ... , n-1. The *total length* of a path is the sum of the Euclidian distances between v_i and v_{i+1} for $i = 0$, ... , n-1. An *ε-path* is a path with a total length smaller than ε. Two mesh voxels are called *local neighbors* if the associated vertices of these voxels are connected via an ε-path. Neighboring voxels that do not satisfy this criterion are called *global neighbors*. For the thinning we will only consider local neighbors. Therefore we introduce the notion of a local $N^+_\alpha(v)$ - neighborhood ($\alpha \in \{6, 18, 26\}$) that consists of all local $N^+_\alpha(v)$ - neighbors of v. The superscript "+" refers to a stricter neighborhood concept.

Furthermore, we call a set of vertices V_{mv} associated to a mesh voxel mv *ε-connected* if each pair of vertices in V_{mv} is connected via a sequence of ε-paths such that at least one vertex of each ε-path belongs to V_{mv}. We call a set of triangles defined over a set of ε-*connected* vertices an *ε-connected surface segment* or shorter a *surface segment*.

A set of vertices V_{mv} associated to a mesh voxel mv which is not ε-connected characterizes a special case where a voxel contains more than one surface segment. Such voxels are called *multiple surface voxels* (see Figure 2c). To solve this problem any voxel v with m disconnected surface segments is (m-1)–times duplicated. Each copy of v receives the same coordinates as the original voxel but only one of its surface segments represented by a set of ε-connected vertices. Each voxel-duplicate is stored in the data structure described below. This operation is equivalent to splitting the voxel into subsets of smaller volume elements (see Figure 3). The advantage of duplicating as a method for "surface separation" is the feasibility to store this voxel-duplicates in the volume data structure with marginal effort. Only an additional identifier is needed to distinguish between the duplicated voxels. No changes concerning the resolution are required (e.g. sub-division).

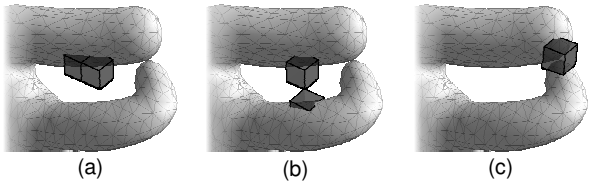

Figure 2: The neighborhood of two local voxel neighbors (a), two global voxel neighbors (b) and a multiple surface voxel are shown (c).

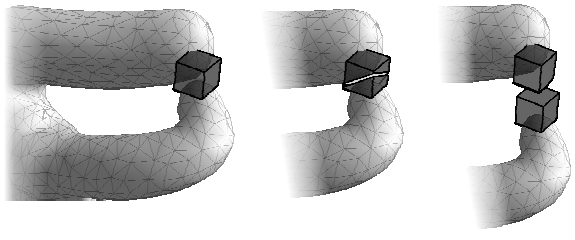

Figure 3: A multiple surface voxel (a) can be splitted into two volume elements (b). Alternatively, the voxel can be duplicated so that each voxel-duplicate contains a list with enclosed connected vertex set (c).

3. VOLUME GENERATION FOR THE MESH MODEL

3.1 Run length encoding

In our approach, we intend to minimize the voxel size for a fixed memory capacity in order to obtain a highly detailed curve skeleton. To avoid the cubic memory requirement of a 3D grid we use a *run length encoding* for the volume representation of the shape S:

$$R(S) = \{run(c, b, i) \mid 0 \le c < C, 0 \le b < B, 0 \le i < Q(c, b)\},$$

$$run(c, b, i) = (n(i), m(i))$$

where $Q(c, b)$ indicates the number of runs at column c and band b, $n(i)$ denotes the start position, and $m(i)$ denotes the end position of the i-th run within the row(c, b) ($0 \le n(i) < m(i) < R$).
Since a $run(c, b, i)$ has two intersections with the closed mesh, it is sufficient for each volume element in the column c and the band b to store only two coordinates $n(i)$, $m(i)$ that correspond to these intersections. Figure 4 illustrates the data structure.
We use a scanline algorithm to create the volume representation, because the results of the scans can be stored as runs. For details see [4].

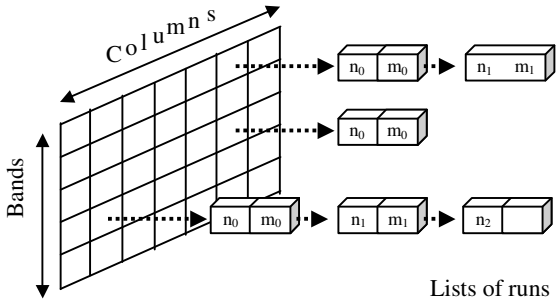

Figure 4: A graphical view of the data structure holding the runs. Each run contains two coordinates to specify the rows where the consecutive voxels start and end.

3.2 Determination of mesh voxels

In the next step we determine all mesh voxels and associate their vertices with certain runs in our data structure. To find the mesh voxels we proceed as follows. For each triangle $t \in M(S)$ with vertices t_a, t_b, t_c we determine the voxels v_a, v_b, v_c (which may of course be identical) with $t_a \in v_a$, $t_b \in v_b$, and $t_c \in v_c$. If v_a, v_b, v_c are not pairwise N_6- neighbors t is subdivided into four sub-triangles t_1, t_2, t_3, t_4 according to the split depicted in Figure 5.

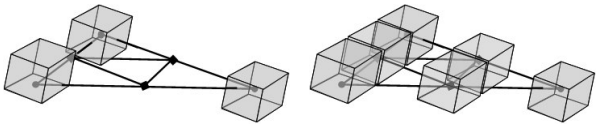

Figure 5: The three voxels are not N_6 - neighbored so the triangle is split into four sub-triangles.

This procedure is recursively applied to all sub-triangles until all determined voxels of each sub-triangle are pairwise N_6-neighbors.
Whenever for a vertex the corresponding mesh voxel mv with the coordinates (c_{mv}, b_{mv}, r_{mv}) has been determined this information is stored in our data structure. For this we have to distinguish the following cases. Remember, a run $\rho \in R(S)$ is described by the four coordinates c, b, $n(i)$, and $m(i)$. First case, no run ρ with the coordinates $c = c_{mv}$, $b = b_{mv}$, $n(i) \le r_{mv} \le m(i)$ exists in $R(S)$. Than mv is added as new run of length 1 (i.e. $n(i) = m(i)$) to $R(S)$. Second case, a run ρ with the coordinates $c = c_{mv}$, $b = b_{mv}$, $n(i) = r_{mv}$ exists. In this case, the additional information of mv (list of vertices, duplicate identifier) is assigned to ρ. The third case applies if a run ρ exists with the coordinates $c = c_{mv}$, $b = b_{mv}$, $n(i) < r_{mv} \le m(i)$. Than ρ is divided into two runs ρ_1 and ρ_2 with coordinates $\rho_1 = (c, b, n(i), r_{mv} - 1)$ and $\rho_2 = (c, b, r_{mv}, m(i))$. Now for ρ_2 the second case applies.

3.3 Duplication of multiple surface voxels

The distinction of neighboring voxels into local and global neighbors can be achieved much easier if the voxel set contains no multiple surface voxels. Therefore prior to the neighborhood analysis we determine all such voxels and distribute the surface segments of any such voxel onto different copies of itself. For this we visit all mesh voxels and test for any one of these whether the associated vertices are ε-connected. Starting from an arbitrary vertex $v \in V_{mv}$, we use an adapted breadth first search algorithm to determine all vertices connected to v via an ε-path. The resulting set of vertices is called the ε-set of v. This search is performed for all vertices of V_{mv} that are not yet found to be members of an ε-set or a union of ε-sets. During the search algorithm two sets (ε-sets or unions of ε-sets) are united if a vertex is found that belongs to both sets. After all vertices have been processed the resulting sets correspond to the ε-connected surface segments of V_{mv}. For ε we use the prescribed distance $\varepsilon = 2 \cdot$ side length of a voxel.
In regions where the given mesh is in one direction thinner than the voxel size multiple surface voxels may occur because a voxel penetrates opposite sides of the closed mesh (see Figure 6a). For topological reasons we have to distinguish this special case (we call it a *breakthrough*) where the surface segments within the voxel are "connected" over the interior of the mesh from the more common case (see Figure 6b). A breakthrough will not be duplicated because the duplication could lead to unwanted separation of the object in the thinning process. To determine breakthroughs we use a heuristic which is derived from the observation that in this situation most of the normal vectors n_i of the vertices $v_i \in V_{mv}$ ($i = 1, ..., q$; q = number of vertices) are oriented away from the voxel center (see Figure 7).

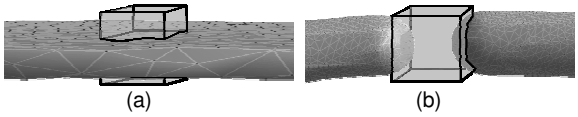

Figure 6: Both figures show multiple surface voxels. In (a) it is a "breakthrough" because it is situated in a thin region. In opposite in (b) the surface segments belonging to the voxel are really separated.

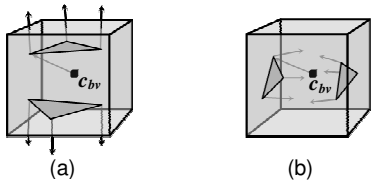

Figure 7: Two triangles for a breakthrough (a) and two triangles for a "real" multiple surface (b) are shown.

Let c_{mv} be the center of a mesh voxel mv that contains multiple surface segments. Then we define a function $dir(v, n)$ to distinguish between the possible principal directions of the normal vector n at the vertex v with respect to c_{mv}

$$dir(v,n)=\begin{cases} 1 & if\ \langle (v-c_{mv}),n\rangle > 0 \\ -1 & else \end{cases} .$$

A second Boolean function $breakthrough(mv)$ is used to determine breakthroughs:

$$breakthrough(mv) = \begin{cases} \text{true} & if \ \sum_{i=1}^{q} dir(v_i, n_i) > 0 \\ \text{false} & else \end{cases}$$

.

Any multiple surface voxel that is not a breakthrough will be (m-1)-times duplicated where m is the number of its surface segments. According to the duplication process described in section 2, each duplicate contains exact one surface segment. The duplicates are added to that list of runs which holds the original mesh voxel mv. Note, that because of the existence of duplicates, it is possible to have more than α adjacent neighboring elements in the N^+_α - neighborhood ($\alpha \in \{6, 18, 26\}$) for a voxel.

3.4 Finding global voxel neighbors

The preprocessing described in section 3.3 guaranties that each voxel contains only one surface segment or (in the exceptional case of a breakthrough) can be treated as if it contains only one surface segment. Therefore the task of determining global neighbors can be tremendously simplified: two adjacent mesh voxels mv_1 and mv_2 are local neighbors if there exists one ε–path from the set of vertices V_1 belonging to mv_1 to the set of vertices V_2 belonging to mv_2. Again an adapted breadth first search algorithm is applied to decide whether such an ε–path exists.
If no ε–path can be found between V_1 and V_2, we test whether the union $mv_1 \cup mv_2$ forms a breakthrough similar to the situation described in section 3.3. Here we use the vertices and normal vectors of both sets (V_1 and V_2) and the center c is defined as center of the centers of mv_1 and mv_2:

$$c = ½ (c_{mv1} + c_{mv2}).$$

If the union $mv_1 \cup mv_2$ forms no breakthrough mv_1 and mv_2 are declared to be global neighbors. This information is stored for both voxels. In the thinning process global neighbors are not considered as adjacent voxels.

4. THE SEGMENTATION ALGORITHM

The segmentation algorithm consists of the following steps:

1. Extract the voxel skeleton through a thinning process that distinguishes between local and global neighbors
2. Transform the voxel-based skeleton into a graph representation G
3. Associate the graph knots $k_i \in K$ with the mesh triangles $t_j \in M(S)$
4. Find segments of the graph G

4.1 Thinning of the volume

In order to extract the voxel skeleton, we employ a thinning approach. Thinning algorithms iteratively delete voxels in a 3D grid that additionally fulfill certain geometric constraints, so that geometric features and topology are retained. In order to ensure even deletion, six thinning directions are defined and iteratively traversed, in the course of which for the current direction r+, r–, c+, c–, b+, b– only the corresponding directed border voxels are considered for deletion.
In the literature, different criteria for voxel removal have been suggested (see [1], [2], [3], [11]). To obtain a thinning process that produces an ideal skeleton for our application, we found it necessary to combine different criteria. Bertrand and Malandain suggest in [2] a voxel characterization based on topological numbers within the N^+_{26} neighborhood (number of components (*#Components*), where components represent the maximum number of α-connected voxels). We use this classification in preselecting the voxels to be deleted. An additional condition also controls the thinning process, known as the "checking plane condition", which ensures that the resulting skeleton is smooth.
We use the following conditions (in this sequence) to characterize a voxel v for removal:

1. $\#N^+_{26}(v) > 1$ (end-voxel exclusion condition)
2. $\#Components(V(S)_f \cap N^+_{26}(v)) = 1 \ \wedge$
   $\#Components(V(S)_b \cap (N^+_6(v))^c) = 1$ (extended border condition)
3. at least two of the three checking planes fulfill the following two conditions, depending on the current thinning direction:
   $\#Components(V(S)_f \cap CP_\beta(v)) = 1$
   $\#Components(V(S)_b \cap (CP_\beta(v))^c) = 1$;
   $\beta = \{c, b, r\}$ (checking plane condition)

The intention of the first condition is to preserve curve end voxels. Without this condition, the current end voxels of the curve would be removed at each iteration until a single voxel remained. The second condition ensures that the deletion of voxel v does not disconnect the remaining voxels belonging to the N^+_{26} - neighborhood.
According to the criterion formulated in the third condition, a voxel can only be removed if the 2D analogy of the extended border condition applies. The two checking planes orthogonal to the current direction of thinning and containing the voxel to be removed are always used for this criterion. For instance, for the directions r+ or r–, the checking planes CP_b and CP_c would be used. Practical experience indicates that this condition shortens the length of the voxel curves and reduces the number of branches.
In Figure 8, two checking planes are shown (the candidate for removal, v, is located in the center of these planes). The removal of v in the illustration on the right would disconnect the foreground voxels, so the number of components increases to two.

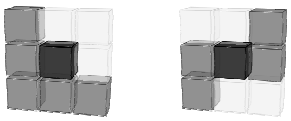

Figure 8: The removal of the center voxel in the checking plane on the right would separate the component into two. In contrast, the left plane still consists of one component after the removal.

In addition to meeting topological and geometrical demands, these three conditions ensure that the resulting skeleton consists of thin voxel curves (shown and discussed in [2], [3], [11]).
There are only two essential grid operations necessary to realize the thinning of the run length-encoded data structure:

- *delete* (c, b, r) to delete a voxel within a run
- *isfilled* (c, b, r) to check if the element at this position belongs to the object or not

From the following pseudocode, it is obvious that the *delete* operation may cause disconnections of runs:

```
procedure delete(c, b, r);
begin
  determine i so that n(i) ≤ r ≤ m(i)
  if run(c, b, i).n = r then          // case 1
    run(c, b, i).n = run(c, b, i).n + 1;
  else if run(c, b, i).m = r then     // case 2
    run(c, b, i).m = run(c, b, i).m - 1;
  else begin                          // case 3
    newrun.n = r + 1;
    newrun.m = run(c, b, i).m;
    run(c, b, i).m = r - 1;
    run(c, b).add(newrun);
  end;
end;
```

The *isfilled* function checks whether, for the two coordinates *c,b,* a *run*(*c*, *b*, *i*) exists for $n(i) \le r \le m(i)$. Since just a few runs are enough for typical objects, a linear search suffices. In the case of more complex objects, a binary search is more suitable.

```
function isfilled(c, b, r): boolean;
begin
  for all run(c, b, i) do begin
    if r >= run(c, b, i).n and
       r <= run(c, b, i).m then
      return true;
  end;
  return false;
end;
```

These two functions suffice to apply the thinning algorithm to the data structure *R(S)*. The result of thinning is a continuous voxel set, which we denote voxel skeleton and which consists of "voxel curves" (see Figure 9). A voxel curve is defined as follows: $C \subset V(S)_f$ is considered a voxel curve if, for the elements v_1, ..., v_n ($n >$ 1) of *C*, the following applies:

$N^+_{26}(v_1)=\{v_2\}$,
$N^+_{26}(v_n)=\{v_{n-1}\}$,
$N^+_{26}(v_i)=\{v_{i-1}, v_{i+1}\}$ for $i = 2, \ldots, n-1$.

This means that a voxel v_i of a curve has a maximum of two adjacent voxels: the preceding voxel v_{i-1} and the succeeding voxel v_{i+1}. The voxel skeleton now results from the joining of voxel curves, i.e. $L \subset V(S)_f$ denotes a voxel skeleton if a segmentation of *L* into voxel curves exists such that the following hold true:

1.) $L = C_1 \cup C_2 \cup \ldots \cup C_m$, where C_i (i = 1, ..., m) are voxel curves
2.) For any two voxel curves C_i, C_j, at most one neighboring voxel pair (v, w) exists with $v \in C_i$, $w \in C_j$ and $v \in N^+_{26}(w)$
3.) *L* is connected

With all these conditions and the two described operations *delete* and *isfilled*, the resulting algorithm appears as follows:

```
procedure ExtractSkeleton;
begin
  do begin
    for all directions d do begin
      for all bands b do begin
        for all columns c do begin
          for all runs r do begin
            for e = r.n to r.m do begin
              // check if current voxel is a
              // border voxel with the current
              // direction d using the function
              // "isFilled"
              if isFilled(c, b, e) and not
                 isFilled(c+d.c, b+d.b, e+d.r);
                 then add to Candidates;
              // the candidates list contains
              // all current border voxels
            end;
          end;
        end;
      end;
    end;

    for all Candidates cand do begin
      // analyze neighborhood N+26 of cand
      // after then, check the conditions
      if not curve end voxel condition then skip;
      if not border condition then skip;
      if not checking plane condition then skip;

      // all conditions fulfilled; so delete
      // cand using the procedure "delete"
      delete(cand.c, cand.b, cand.r);
    end;

  while some deletions are made;
end;
```

Different direction sequences lead to different skeletons, but the quality of the skeleton depends primarily on the resolution.

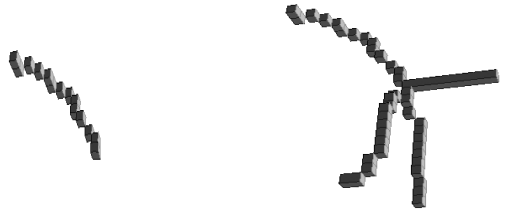

Figure 9: At left a single curve consisting of voxels is displayed, at right on the other hand several joined curves, which form a curve skeleton.

4.2 Segmentation based on the skeleton graph

It is not possible to perform a reasonable segmentation of the mesh based on the voxel skeleton because it contains a high number of junctions (voxels with more than two neighbors). Therefore we transform the voxel skeleton into a graph representation in order to remove redundant junctions.
The initial skeleton graph $G = \{K, E\}$ with knots *K* and edges *E* is build as follows: each voxel that is part of the skeleton defines a knot and each pair of 26-adjacent voxels define an edge.
Let $N(k)$ denote the set of knots that share an edge with the knot *k*. The number of neighboring knots can be used to distinguish the type of knot:

- one neighbor: end knot,
- two neighbors: regular knot,
- more then two neighbors: junction knot.

In order to remove junction knots from the graph neighboring junction knots are merged. For iterative merging we determine for each loop the edge of two neighboring junction knots with minimal cost according to the following cost function. Let $j, k \in K$ be junction knots and $k \in N(j)$ one of the neighbors of *j*. Let $m_i \in N(k)$ for $i = 1, \ldots, n$ further be all *n* neighbor knots of *k*. By merging *j* and *k*, all neighbors of *k* are passed over to *j*. This is why all distances between m_i and *j* or between m_i and *k* are introduced into the cost function:

$$\text{cost}(j,k) = \sum_{i=1}^{n} \|j, m_i\| + \|k, m_i\|$$

This function measures how many edges are concerned and how long they are in sum. It can be used to achieve small distances between knots.

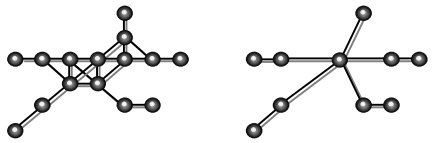

Figure 10: The left graph contains all junctions between the voxel skeleton as edges. In the right graph, some knots are merged in order to delete undesirable edges.

It is always the edge with minimal costs that is merged. This operation is repeated until no neighboring junction knot pair exists.
As can be seen in Figure 10 in the image on the right, the merging results in only one junction knot, instead of eight as on the left.
After the skeleton graph has been freed of superfluous knots, the remaining knots must be associated with triangles, in order to be able to draw conclusions for the mesh triangles. This is especially important if knots are combined into segments. The calculation is again very simple, since the related cost function includes only the distance between the triangle center and the knot. If m_t is the center of a triangle $t \in M(S)$, and $k \in K$ is a knot in the graph, then the cost function is given by:

$$\text{cost}(m_t, k) = \| m_t, k \|, \text{ with } m_t = \frac{1}{3}(t_a + t_b + t_c).$$

A segment of the triangulation is defined by a chain of knots in the skeleton graph, i.e. a series of neighboring knots that begin or end with a junction or end knot. With the exception of the first and the last knot, all knots in the chain are regular knots.
It should now be clear how important the removal of the superfluous knots was; this reduced the number of junction knots, yielding fewer shorter segments.
Finding the chains takes place recursively, similar to the well-known Depth-First-Search algorithm from graph theory. Every knot is assigned a number that indicates what segment it belongs to.
As indicated above, the segment affiliation of the graph knots in conjunction with the association of the triangles with the knots yields the implicit association of triangles with the segments.

5. RESULTS

Figure 12 indicates typical results that can be achieved with the presented method. The three objects displayed feature the following characteristics:

- "Pig": 7040 triangles, 10 segments, resolution: 150 x 84 x 53,
- "Hand": 10846 triangles, 8 segments, resolution: 100 x 30 x 22,
- "Horse": 39698 triangles, 10 segments, resolution: 85 x 124 x 150

In order to provide an overview of runtimes in practical use, we list several run time values for the three models mentioned. We achieved the results on an Intel P4 2.8 GHz, 1 GB RAM.

Table 1: Computational time for the complete segmentation.

	100	**200**	**300**	**400**	**500**
Pig	1.23 s	8.73 s	27.24 s	64.77 s	129.78 s
Hand	0.64 s	3.38 s	11.31 s	24.9 s	49.06 s
Horse	2.29 s	9.20 s	28.24 s	95.04 s	154.03 s

The computational times are of course significantly high. But keep in mind that low resolutions are often already adequate for good segmentation, and can be calculated within a few seconds. The segmentation shown in Figure 12, for example, required 5.5 sec for "Pig," 0.6 sec for "Hand," and 5.1 sec for "Horse."
Experience shows that implementations in which the thinning algorithm does not use the additional global neighbor information require on average only three-fourths the computational time cited here.

The memory requirements for the presented data structure are described in detail in [4].

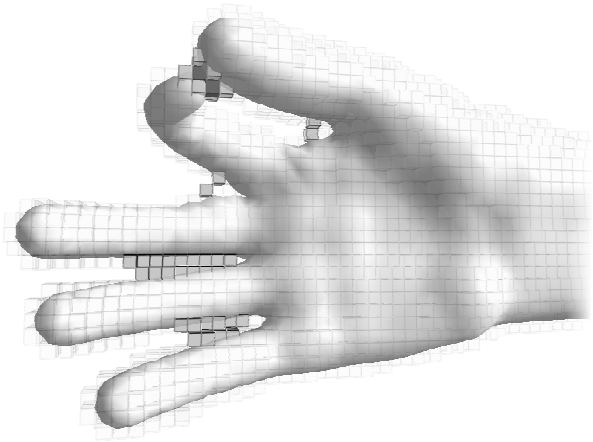

Figure 11: For the hand model all found global neighboring voxels (light gray) and two multiple surface voxels (dark gray) are shown.

6. REFERENCES

[1] Bertrand, G. and Aktouf, Z., "A three-dimensional thinning algorithm using subfields", *Proceedings of the SPIE Conference on Vision Geometry*, 1994, 113-124

[2] Bertrand, G. and Malandain, G., "A new topological classification of points in 3d images", *2nd European Conference in Computer Vision*, 710-714, 1992

[3] Bezerra, F. N. and Leite, N. J., "Some Comments on Thinning Algorithms for 3-d Images", University of Campinas, 1998

[4] Brunner, D., Brunnett, G., "Mesh Segmentation using the Object Skeleton Graph", *Computer Graphics and Imaging*, 48-55, 2004,

[5] Dey, T. K., Giesen, J. and Goswani, S., "Shape Segmentation and Matching with Flow Discretization", *Proc. Workshop on Algorithms and Data Structures* 2003, 25-36

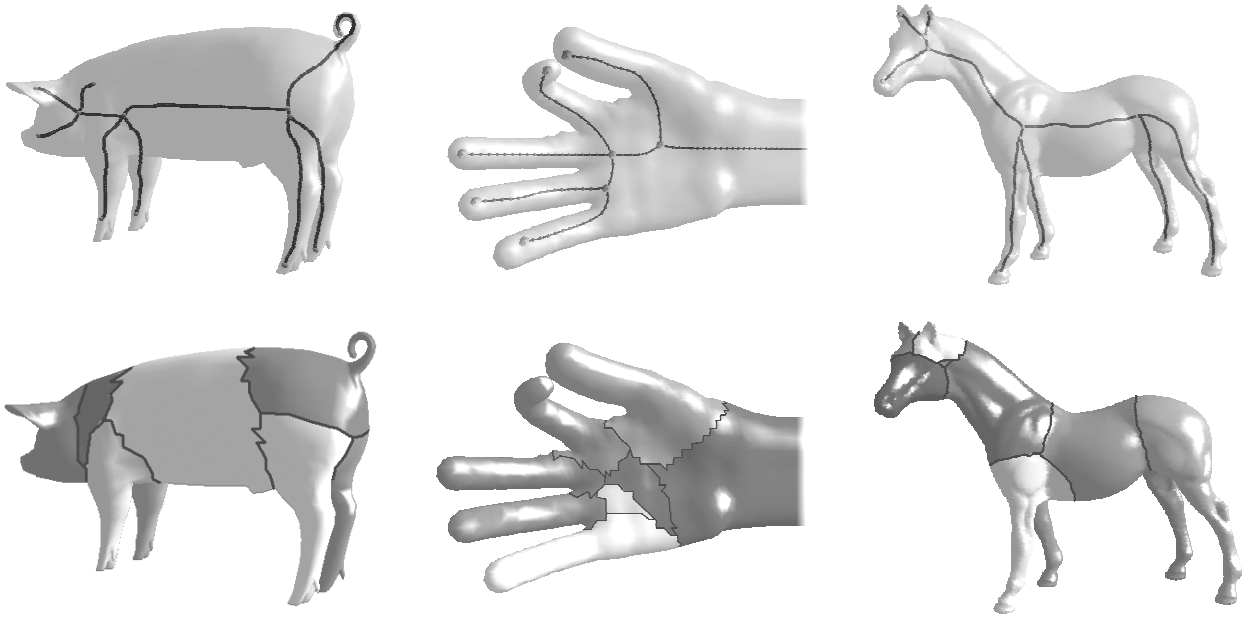

Figure 12: Segmentation results for different models. Respectively, the skeleton graph and the segmented mesh are shown.

[6] Hilaga, M., Shinagawa, Y., Kohmura, T. and Kunii, T., "Topology Matching for Fully Automatic Similarity Estimation of 3D Shapes", *Proc. 28th Conf. Computer graphics and interactive techniques*, 2001, 203 - 212

[7] Kong, T. Y., Roscoe, A. W., Rosenfeld, A., "Concepts of digital topology: Introduction and survey", *Computer Vision Graphics and Image Processing*, 48:357-393, 1989

[8] Rodríguez, J., Thomas, F., Ayala and L. Ros, D., "Efficient Computation of 3D Skeletons by Extreme Vertex Encoding", *DGCI 2003*, 338-347

[9] Sebastian, T., Klein, P. N. and Kimia, B. B., "Recognition of shapes by editing shock graphs", *Proc. ICCV*, 2001

[10] Sethian, J. A., *Level Set Methods and Fast Marching Methods*, Cambridge University, 1999

[11] Tsao, Y. F., Fu, K. S., "A parallel thinning algorithm for 3D pictures", *Computer Graphics Image Processing*, 17:315-331, 1981

[12] Vanco, M., *"A Direct Approach for the Segmentation of Unorganized Points and Recognition of Simple Algebraic Surfaces",* Ph.D. thesis, University of Technology Chemnitz, 2003