A revision of a 3D Skeletonization algorithm

Mian Pan and Gisela Klette

Department of Computer Science University of Auckland
Centre of Image Technology and Robotics (CITR)
Tamaki Campus, Morrin Road, Building 731, Glen Innes,Auckland
New Zealand

Abstract

This report is about a project that started with studying and testing an existing program by K.Palagyi et al. published in [4]. It provides several combinations of preprocessing steps with the traditional thinning approach. This study lead to a proposal of a new characterization of non-simple voxels, which has been implemented and proved to be efficient for reducing the running time of the algorithm. Altogether, we suggest a revision of the test of voxels to be simple in the discussed 3D skeletonization algorithm.

1 Introduction

Modern imaging techniques allow the generation of large 3D volume data sets, captured at high resolution, such as MRI and CT data. New ways must be found to maximize the information acquired from data sets while minimizing the cost of interacting with it. Skeletonization of such a 3D volume is a well-known preprocessing step to extract object features for complex classification methods. The literature contains a large variety of algorithms for finding 3D skeletons. Most of them can be divided into two categories of thinning based methods or distance transform based methods. Thinning is an iterative method that first detects simple points in components satisfying specific constraints, and then converts those into background points. A thinning procedure will not stop

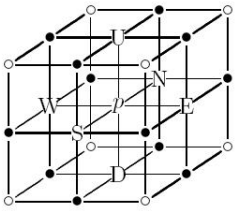

Figure 1: The neighborhood of a voxel p. $N_6(p)$ includes the central voxel p and six 6-*adjacent* voxels which are U, D, N, S, E and W. $N_{18}(p)$ includes $N_6(p)$ and the voxels marked with "•". $N_{26}(p)$ includes $N_{18}(p)$ and the voxels marked with "∘".

until no further simple point can be found. The result of thinning is a skeleton that preserves the topology of the image.

In this report, we discuss an algorithm proposed by Palagyi et al. [3], which is based on a 6-subiteration thinning method with options to calculate anchor points based on a special distance transform or based on a shrinking algorithm.

The notion of a simple point (pixel or voxel) is of basic importance for thinning algorithms. A point is simple iff the change of its value does not change the topology of the image, in the sense that there is a bijective map between all components before and after thinning. In a binary 3D image a voxel has 26 neighbors. Therefore, it has 2^{26} possible configurations in its neighborhood. Many thinning techniques use lookup tables with 2^{26} entries and an actual configuration will be compared in order to determine simple voxels. We use instead the characterization of simple points based on the concept of attachment sets. It turns out that this approach simplifies the implementation and saves running time.

2 Basic Notions, Simple Voxels

2.1 Basic Notions

A *digital image* I is a function defined on a discrete set C, which is called the carrier of the image. The image carrier is defined on an orthogonal grid. In the grid point model, the vertices of a cube in a 3D volume are 3D voxel locations with integer coordinates.

We use the grid cell model in order to introduce the notion of the *I-attachment set*. In the grid cell model, a voxel is represented by a 3-*cell*. The *frontier* of p is an union of i-cells (for i = 0, 1, 2). Kong [5] defined the *I-attachment set* as follows:

Definition 1. [5] *The I-attachment set S of p in I is the set of all points of an α-cell, $\alpha \in \{0, 1, 2\}$ on the frontier of p that also lies on an α-cell of the frontier of at least one other point q with I(p) = I(q), $p \neq q$.*

Kong [5] also proposed Schlegel diagrams to represent S (see Figure 3).

2.2 Characterizations of Simple Voxels

There are formally different definitions of simple voxels in the literature. Actually, most of them are equivalent. We introduce two definitions of a simple voxel for (26,6) pictures. Theorem 1 is formulated in terms of the grid point model.

Theorem 1. [3] *An object point p is simple in picture (Z^3, 26, 6, B) if and only if all the following conditions hold:*

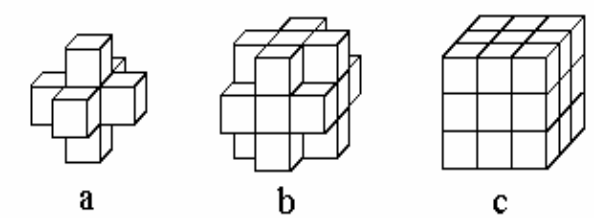

Figure 2: (a) $N_6(p)$ (b) $N_{18}(p)$ (c) $N_{26}(p)$

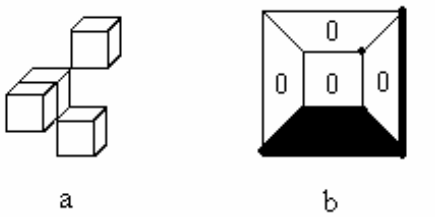

Figure 3: (a) The voxel in the middle and (b) its I-attachment set S in form of a Schlegel diagram.

1. *The set* $N_{26}(p) \cap (B \setminus \{p\})$ *is not empty (i.e., p is not an isolated point);*
2. *The set* $N_{26}(p) \cap (B \setminus \{p\})$ *is 26-connected (in itself);*
3. *The set* $(Z^3 \setminus B) \cap N_6(p)$ *is not empty (i.e., p is a border point);*
4. *The set* $(Z^3 \setminus B) \cap N_6(p)$ *is 6-connected in the set* $(Z^3 \setminus B) \cap N_{18}(p)$.

In Theorem 2 we use the grid cell model.

Theorem 2. [6] *An object point p in image* $\boldsymbol{I}$ *is simple if and only if the I-attachment set S and the complement of S* ($\bar{S}$) *are connected and the Euler number of the I-attachment set is 1* ($\varepsilon(S(p)) = 1$).

The Euler number of the I-attachment set is defined by: $\varepsilon(S(p)) = n_0 - n_1 + n_2$, where n_0 is the number of 0-cells in S, n_1 is the number of 1-cells in S and n_2 is the number of 2-cells in S.

Based on these two theorems, we conclude that a simple voxel p has following properties:

1. An *object voxel* p must be 6-*adjacent* to at least one *white voxel*. That means p is a *border voxel*.
2. There is one 26-*component* of *object voxels* in the set $(N_{26}(p) \setminus p)$
3. There is one 6-*component* of *white voxels* in the set $(N_{18}(p) \setminus p)$
4. S is not empty and connected.
5. $\bar{S}$ is not empty and connected.
6. If an *object voxel* p is simple, then $\varepsilon(S(p)) = 1$. However, if $\varepsilon(S(p)) = 1$, p might not be simple. See an example in Figure 4.

In Figure 4, the number of 0-cells is 6 ($n_0 = 6$). The number of 1-cells is 5 ($n_1 = 5$). The number of 2-cells is 0 ($n_2 = 0$). The Euler value (ε) is equal to 1. However this voxel is not simple because S and $\bar{S}$ are not connected.

An *object voxel* p is called *end point* if there is only one *object voxel* in $(N_{26}(p) \setminus p)$. An *object voxel* p is called *curve point* if p is not simple and there are exactly two object *voxels* in $(N_{26}(p) \setminus p)$. An *object voxel* p is called *branching point* if p is not simple and there are more than two *simply connected components* in S. We use a *thinning algorithm* as usual for the algorithm that iteratively deletes simple voxels that are not end points. A *shrinking* procedure deletes all simple points and the result of shrinking is a single voxel for a 3D simply connected object. An *object voxel* is called *anchor point* if the voxel is determined in a preprocessing step and the image value stays constant during the following thinning procedure.

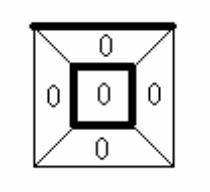

Figure 4: $\varepsilon(S(p)) = n_0 - n_1 + n_2 = 1$, where $n_0 = 6, n_1 = 5, n_2 = 0$ and p is not simple.

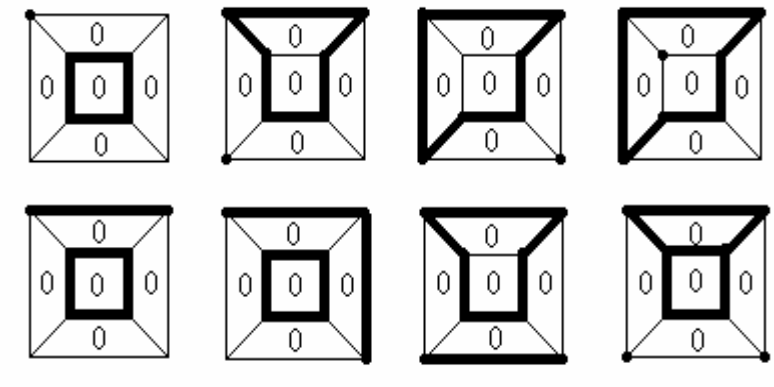

Figure 5: Voxels are not simple and $\varepsilon(S(p)) = 1$ for $N_6 = 6$.

3 Adjustments and Characterizations

In this project, we use Theorem 2. The basic idea is to identify voxels that are not simple.

1. We calculate $\varepsilon(S(p))$. If $\varepsilon(S(p)) \neq 1$, then p is not simple.
2. If $\varepsilon(S(p)) = 1$, then we determine all cases where S is not connected, or $\bar{S}$ is not connected.

Let N_6 be the number of *white voxels* which are 6-*adjcent* to p. All configurations for $N_6 = 6$, $\varepsilon(S(p)) = 1$ and p is not simple are shown in Figure 5.

Proposition 1: *Let $N_6 = 6$. An object voxel p is not simple and $\varepsilon(S(p)) = 1$ iff the I-attachment set S consists of two or three disjoint connected subsets of points. For two sets, one of these sets is a simple curve in the Euclidian space,*

and the other set is a single point or an arc in the Euclidian space. For three disjoint sets, one of these sets is a non-simple curve in the Euclidian space, and the other sets are single points in the Euclidian space.

Proof. Let p be a *black voxel*. Let X be a subset of S ($X \subseteq S$). If X is a simple curve in the Euclidian space, then $\varepsilon(X(p)) = 0$ ($n_0 = n_1, n_2 = 0$) (See Figure 6/a). If X is a single point or an arc in the Euclidian space, then $\varepsilon(X(p)) = 1$ ($n_0 - n_1 = 1, n_2 = 0$) (See Figure 6/a). If X is a non-simple curve in the Euclidian space, then $\varepsilon(X(p)) \leq -1$ ($n_0 - n_1 \leq -1, n_2 = 0$)(See Figure 6/b/c).

1. We assume p is not simple and $\varepsilon(S(p)) = 1$ and $n_2 = 0$. Based on Theorem 2 we know that the I-attachment set S or the complement of S ($\bar{S}$) are not connected. $\varepsilon(S(p)) = 1$ if $\sum_{i=1}^{3} \varepsilon(X_i(p)) = 1$. If S consists only of one such subset then $\varepsilon(X(p)) = 1$ if $n_0 - n_1 = 1$. But then S and $\bar{S}$ are connected. This is a contradiction to our assumption. For two nonempty subsets X_1 and X_2 we have only the option that $\varepsilon(X_1(p)) = 1$ and $\varepsilon(X_2(p)) = 0$. X_1 can only be a 0-cell or an arc. For X_2 follows that $n_0 = n_1$ and this is a curve. For three nonempty subsets we have only the option that $n_0 - n_1 \leq -1$ for one subset and then there must be two others with $\varepsilon(X(p)) = 1$. This is only possible if one subset constitutes a non-simple curve and the other two are both single points.

2. Now we assume that S consists of two or three disjoint connected subsets of points. For two sets, one of these sets is a simple curve in the Euclidian space, and the other set is a single point or an arc in the Euclidian space. For three disjoint sets, one of these sets is a non-simple curve in the Euclidian space, and the other sets are single points in the Euclidian space. It follows immediately that p is not simple and the $\varepsilon(S(p)) = 1$.

In conclusion, for $N_6 = 6$, all possible cases for a non-simple voxel p and $\varepsilon(S(p)) = 1$ are covered in Figure 5. □

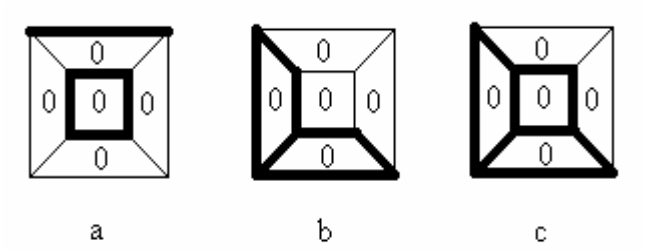

Figure 6: (a)S consists of two disjoint sets. X_1 is a simple curve ($\varepsilon(X_1(p)) = 0$, $n_0 = n_1$, $n_2 = 0$) and X_2 is an arc ($\varepsilon(X_2(p)) = 1$, $n_0 - n_1 = 1$, $n_2 = 0$) (b)S is a non-simple curve and ($\varepsilon(S(p)) = -1$, $n_0 - n_1 = -1$, $n_2 = 0$) (c)S is a non-simple curve and ($\varepsilon(S(p)) = -2$, $n_0 - n_1 = -2$, $n_2 = 0$)

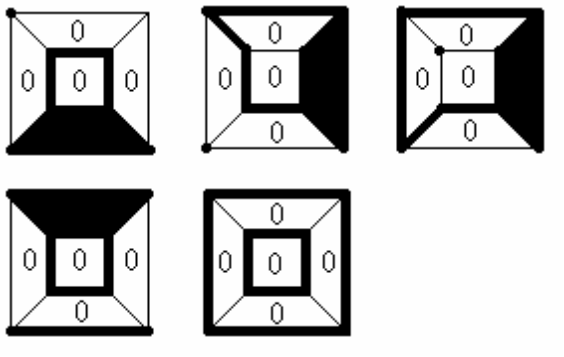

Figure 7: Voxels are not simple and $\varepsilon(S(p)) = 1$ for $N_6 = 5$

For $N_6 = 5$, all cases for a non-simple voxel p and $\varepsilon(S(p)) = 1$ are shown in Figure 7. For $N_6 = 1, 2, 3, 4$, only one case meets this condition, which is shown in Figure 8.

We can give the following preposition:

Proposition 2:*Let* $1 \leq N_6 \leq 6$, $N_{26}(\mathrm{p}) \cap B(\mathrm{p}) > 1$. *An object voxel* p *is not simple iff* $\varepsilon(S(p)) \neq 1$ *or* $\varepsilon(S(p)) = 1$ *and* S *includes an isolated 0-cell or an isolated 1-cell or* $\bar{S}$ *includes an isolated 2-cell.*

As a conclusion we can simplify the algorithm for the identification of non-simple voxels in the following way: Let $N_{26}(p) \cap B(p) > 1$.

1. Calculate the Euler number $\varepsilon(S(p))$.
2. If $\varepsilon(S(p)) \neq 1$ then we keep p.
3. If $\varepsilon(S(p)) = 1$ then do the following steps.

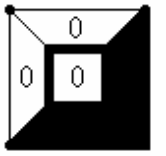

Figure 8: Voxels are not simple and $\varepsilon(S(p)) = 1$ for $N_6 = 1, 2, 3, 4$

4. Find a single isolated point (0-cell) in S.

5. Find an isolated arc (1-cell) in S.

6. Find a simple curve in S. That is, find an isolated 2-cell in $\bar{S}$.

Palagyi [3] identifies simple points in his program based on checking the connectivity (Theorem 1). In his code, each *object voxel* gets a label. If all *object voxels* in $N_{26}(p) \setminus p$ have the same label, then all *object voxels* are 26-*connected*. A similar algorithm is used to check that all *white voxels* in $N_6(p) \setminus p$ are 6-*connected* in $N_{18}(p) \setminus (p)$. For the details of the implementation, please see [3]. We replaced the part of the code with our method and applied both algorithms for the same examples in a computer with 500Hz CPU and 192M RAM. We identified exactly the same number of simple points. Our method is nearly four times faster than the method proposed in [3].

4 Algorithm for 3D Skeletonization

Thinning is described as boundary peeling which iteratively peels off the boundary layer by layer by identifying and removing the simple voxels with additional conditions (for example the end point condition). Changing the image values of simple voxels does not affect the topology of the object. Typical thinning algorithms work in sequential order or in parallel order. A local operation on an *object voxel* p of an image I is called sequential if the decision whether p holds the conditions or not is based on the new values of its neighbors, which have already been processed, and its succeeding neighbors in the defined raster sequence. A local operation on an *object voxel* p of an image I is called parallel if the decision is only based on its neighbor's original values.

We analyzed the C-code provided by K.Palagyi on the internet. The new value of each voxel depends on its $3 \times 3 \times 3$ neighborhood. An object voxel changes to a white voxel in one subiteration if and only if its $3 \times 3 \times 3$ neighborhood matches at least one of the given masks (lookup table). The masks are constructed for each subiteration according to the six directions.

4.1 Set up lookup tables for checking simplicity

The code uses two lookup tables. The first set of masks is called *thin table*. The thin table is used to identify voxel configurations that are simple and not end points. The second table is called *shrink table*. The *shrink table* is only used for checking if a corresponding voxel is simple. In the provided code, shrinking is done to generate *anchor points* in a preprocessing step. The aim of shrinking is to convert a simply-connected object into a single point and a not simply connected object into a Jordan curve for 2D slices.

4.2 3D sequential 6-subiteration thinning algorithm

The code is an implementation of a sequential 3D 6-subiteration thinning algorithm proposed in [3,4]. The pseudo code below is a representation of the provided code. The variable *surfaceVoxelList* is a Linklist storing all the surface voxels in six directions. The variable *list* is a Linklist storing the surface voxels in one direction. According to the pseudocode in [3], the *object voxel* p will be added into *list* if p is a simple but not end point. In *DetectBorderPoints()* method of provided code, p is added into *list* as long as it is a border voxel.

```
    void DetectBorderPoints(linkedlist *list, char *image)
    {
       for each point p in surfaceVoxelList do
          if p is border voxel in direction i then
        //Change here
   ---> //if p is simple and not end in direction i then
             add p into list;
    }
    int iteration_step(char *image)
    {
       int modified = 0;
       linkedlist list;
       for each direction i do
       {
          DetectBorderPoints(&list, image);
          while list is not empty do
          {
             p = GetFromList(&list);
             if p is simple and not end then
             {
                set p to 0;
                modified++;
                update surfaceVoxelList;
             }
          }
       }
       return modified;
    }
    void sequential_thinning(char *image)
    {
       initialize thin table;
       initialize surfaceVoxelList;
       modified = 1;
       while modified > 0 do
          modified = iteration_step(image);
    }
```

We used the 2D image in Figure 9/a to create a 3D object shown in Figure 9/b. The image in Figure 9/c is the skeleton generated by the provided code. Figure 9/d is the skeleton generated by the adjusted code. Figure 9/d is a good representation of the shape. The reason is that simplicity of a border voxel changes depending on its already processed neighbors. If all the border voxels in one direction are checked in the deletion procedure, some border voxels become deletable or undeletable due to its processed neighbors. In fact, the border voxels becoming deletable during the deletion procedure should be processed in the next iteration step in order to keep the shape. Therefore, for one subiteration step, only those simple and not end voxels will be collected to go through the following sequential deletion procedure. In this way, the skeleton will be forced to be close to the central axis of the original object. Each subiteration step has two phases. The fist phase is to collect all simple and not end voxels in its corresponding direction. The second phase will re-check whether these collected voxels are the simple and not end points when deleting them sequentially. Figure 10 shows the result of a different example.

4.3 Find anchor voxels

This code provides a number of different options to calculate skeletons. There is the possibility to combine the advantage of topology preserving thinning with methods based on the distance transform. Two ways are implemented to find *anchor points* in the provided code as preprocessing steps. One option calculates anchor points based on distance transform.

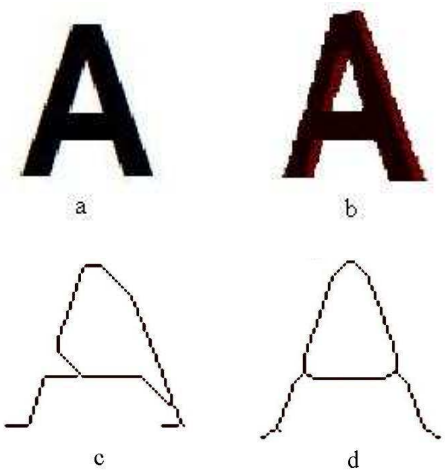

Figure 9: (a) 2D image (b) 3D object constructed by a. (c) Skeleton generated without considering the condition that the voxel is a simple and not end point (d) Skeleton generated with the added condition

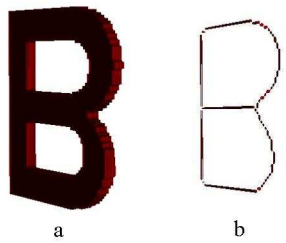

Figure 10: 3D object and the skeleton of this object

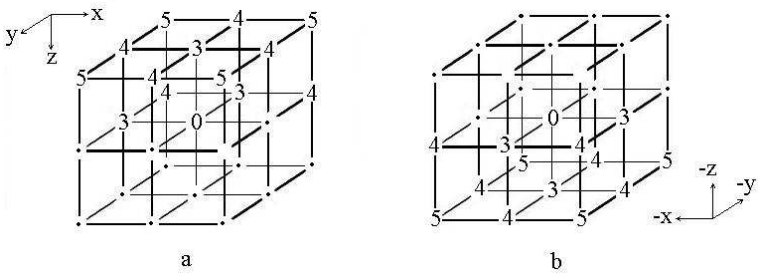

Figure 11: (a) Forward (3, 4, 5)-chamfer mask (b) Backward (3, 4, 5)-chamfer mask

Definition 2: *Let* $\langle I \rangle \subseteq I$ *and* $\langle \bar{I} \rangle \subseteq I$. *For any grid metric* d_α, *the* d_α *distance transform of* I *associates with every voxel* p *of* $\langle I \rangle \subseteq I$ *the* d_α *distance from* p *to* $\langle \bar{I} \rangle \subseteq I$.

K.Palagyi uses the (3, 4, 5)-chamfer distance transform to find *anchor points*. Two masks are created according to the distance weights for all 26 neighbors (see Figure 11). (3, 4, 5)-chamfer distance transform consists of a forward scan and a backward scan. In a forward scan distance values are calculated applying the first mask (see Figure 11/a). It starts from the top left corner of the top slice, then along the direction shown in Figure 11/a. A backward scan follows using the second mask in the opposite direction shown in Figure 11/b. Within the minimum distance field generated by (3, 4, 5)-chamfer distance transform, the voxels with maximum values in its local neighborhoods are anchor voxels.

The second option uses the *shrink table* for 2D images to generate the *anchor points* in a preprocessing step.

1. Find the lowest and the highest image slice with *object voxels*, say m and n, and then shrink them to get the voxels as the *anchor points*.

2. Start from m and n, find the first slice in upward and downward direction

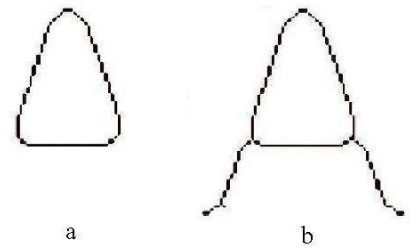

Figure 12: Use *anchor points* generated by either chamfer distance transform or shrink. (a) Skeleton generated without checking the end point condition (b) Skeleton generated by checking the end point condition

respectively, say m1 and n1, which would contain only one *object voxel* after shrinking. The results of the shrinking procedures per slice i ($i \in [m, m1], i \in [n, n1]$) are *anchor points*.

If anchor points were calculated in a preprocessing step based on 2D slices then an additional thinning is used to reduce the number of voxels for the following procedure. A thinning will be applied on the original image with the conditions that a *object voxel* p can be deleted if it is not an *anchor point* and p is simple and not an end point. Because the result is not a thin skeleton, a normal thinning follows.

5 Conclusion

We tested the 3D sequential 6-subiteration thinning algorithm for simple symmetric 3D objects. The program provides a large number of different combinations of preprocessing steps. For complex medical images further testing of all given preprocessing options is required to remove noise.

We changed the actual thinning procedure as the central part of the program based on the propositions in this report. We used the characterization of non simple points based on the grid cell model and we changed the test for voxels to be simple or not. We identified the same simple points as the original version of the program. The running time of the algorithm was 4 times faster.

6 Acknowledgment

We thank K.Palagyi for providing the program code of the sequential 3D 6-subiteration thinning algorithm.

7 Reference

[1] G. Klette, "A Comparative Discussion of Distance Transformations and Simple Deformations in Digital Image Processing", Machine Graphics & Vision, 12: 235-256, 2003.

[2] G. Klette, "Simple Points in 2D and 3D Binary Images", In: CAIP 2003, LNCS 2756, pages 57-64, Springer Berlin, 2003.

[3] K. Palagyi, E. Sorantin, E. Balogh, A. Kuba, C. Halmai, B. Erdohelyi and K. Hausegger, "A Sequential 3D Thinning Algorithm and Its Medical Applications", IPMI 2001, LNCS 2082, pages 409-415, Springer Berlin, 2001.

[4] K. Palagyi, A. Kuba, "A 3D 6-subiteration thinning algorithm for extracting medial lines", Pattern Recognition Letters, 19: 613-627, 1998.

[5] T. Y. Kong, "Topology-Preserving Deletion of 1's from 2-, 3- and 4-Dimensional Binary Images", DGCI 1997, pages 3-18, Springer Berlin, 1997.

[6] C. J. Gau and T. Y. Kong, "Minimal Non-Simple Sets in 4D Binary Images", Graphical Models, 65: 112-130, 2003.