OpenThinning: Fast 3D Thinning based on Local Neighborhood Lookups

Tobias Post* University of Kaiserslautern
Christina Gillmann† University of Kaiserslautern
Thomas Wischgoll‡ Wright State University
Hans Hagen§ University of Kaiserslautern

Abstract

3D Thinning is an often required image processing task in order to perform shape analysis in various applications. For researchers in these domains, a fast, flexible and easy to access implementation is required. Open source solutions, as the Insights Segmentation and Registration Toolkit (ITK), are often used for image processing and visualization tasks, due to their wide range of provided algorithms. Unfortunately, ITK's thinning implementation is computational expensive and allows solely one specific thinning approach. Therefore, this work presents OpenThinning, an open source thinning solution for 3D image data. The implemented algorithm evaluates a moving local neighborhood to find deletable voxels, according to different sets of criteria. In order to reduce the computational effort, all possible local neighborhood setting outputs are stored in a lookup table. To show the effectiveness of OpenThinning, the implementation is compared to the performance of the ITK library.

Keywords: 3D-Thinning, Open Source, Lookup Table

1 Introduction

Image processing tasks occur in various applications as medicine and mechanical engineering [7]. To understand the shape of different objects, centerlines are a common tool. They offer a geometric and topological representation, which allows further examinations [11]. To obtain a centerline, thinning is often the first step, where voxels are successively deleted until a skeleton remains [15]. Unfortunately, the properties of a skeleton are not unique and alter depending on the use case in different application [5].

Independent from the application, skeletons need to fulfill specific requirements to be suitable for further examinations. First, skeletons require a one pixel thickness, to be analyzed properly. Additionally, interesting structures of an image are usually segmented, resulting in a discrete binary mask, which needs to be handled by the thinning algorithm. Furthermore, the thinning result requires the ability to capture the geometry of the thinned objects as well as possible. This means, that the skeleton needs to preserve the length of an object and it needs to be located in the object's center. Moreover, the topological properties of the object have to be retained. This means that, connected components or branches need to be presented by a skeleton with the same properties. Finally, as thinning often needs to be computed for various objects, a fast computation is required. Depending on the application, these list of properties needs to be extended.

As thinning does not require a user input, an open source solution, that can be easily added to existing projects, is desirable. Although multiple solutions for open source 3D-Thinning exist [3, 6, 9, 4], they lack the ability to perform a fast thinning independent from the required properties of the output skeleton. In contrast to that, Lee et al. [12] presented a family of thinning algorithms, that successively removes voxel of an object according to a defined deletion scheme. One of their presented algorithms is available in the free ITK [10] library. Although this is an open source solution, the ITK implementation can be computationally slow and does not offer the possibility to alter the requirements of the tinning result.

Therefore, this paper presents OpenThinning [13], a fast and robust implementation of thinning algorithms, presented by Lee et al. (Section 2). The algorithms remove successively voxels from the input object by evaluating a moving local neighborhood until the desired skeleton remains. In order to achieve a fast computation, all possible neighborhood settings are stored in a lookup table. To provide skeleton outputs with different properties, the system offers three distinct lookup tables. To show the effectiveness of the OpenThinning solution, this paper presents a runtime comparison with the widely used ITK library and demonstrates skeletons obtained with the OpenThinning solution (Section3).

*tpost@rhrk.uni-kl.de
†c_gillma@cs.uni-kl.de
‡thomas.wischgoll@wright.edu
§hagen@cs.uni-kl.de

2 Open Thinning

In order to incorporate a free, flexible and fast thinning in other programming frameworks, this section presents the OpenThinning solution [13]. The free available implementation offers a parallel thinning approaches that evaluate a moving local neighborhood encoded in a lookup table.

2.1 Local Neighborhood Lookup Tables

In order to determine deletable voxels, a moving local neighborhood, as shown in Lee et al.'s approach [12], is used. Therefore, the work states a family of thinning algorithms that can be utilized to obtain skeletons fulfilling different properties from an binary image data. In each thinning algorithm, a list of criteria (see Table 1) is used to evaluate the moving local neighborhood. Their three sets of used criteria and the different thinning results will be explained in the following.

Simple thinning (T_1), requiring criteria $C_1 - C_4$, results in a skeleton that preserves the object's geometry and topology, while neglecting line-like branches.

Instead, the implementation of a **medial axis thinning** (T_2), which additionally preserved the length of an object, requires criteria $C_1 - C_5$. In addition to the properties of the simple thinning algorithm, condition C_5 avoids a shortening of line-like structures, which outputs a medial axis.

To achieve a **medial surface thinning** (T_3) as a thinning result, criteria $C_1 - C_4$ and C_6 are required. Criteria C_6 verifies, if the observed voxel is embedded in the medial surface or its border. If this is the case, the voxel is not allowed to be deleted, thereby remaining the medial surface of the structure.

In the algorithms presented by Lee et al. the moving local neighborhood determines, if a voxel is deletable, by checking the applied criteria according to their order in Table 1. This evaluation is performed until either a criteria is not fulfilled (voxel will not be deleted) or all criteria are fulfilled (voxel will be deleted). Depending on the criteria, that causes the computation to stop, the runtime for a local neighborhood evaluation varies. As a result, the runtime of the approach depends on the presence of slow cases in the input object, whereas a dependency of the image size is preferred.

As the evaluation output of the moving local neighborhood remains the same in the entire computation of the thinning algorithm, their result can be computed in advance and stored in a lookup table. Based on that, the algorithm does not need to recheck all criteria again during its runtime. Instead it can request the output of the current neighborhood setting in the lookup table and therefore requiring an constant time consumption for each voxel evaluation.

To achieve a standardized indexing scheme for arbitrary lookup tables, the presented method of Post et al. [14] is used as shown in Figure 2. The table index is calculated by stringing together the filled (value 1) and not filled voxels (value 0) from the local neighborhood, except for the current voxel value itself. As this value needs to be checked in the thinning procedure as well, this would not lead to a reduction of time consumption. In total, the resulting lookup table holds 2^{26} entries, storing all possible settings of a voxel's local neighborhood. This results in a 8MiB file, that can be loaded in less then one second. In the OpenThinning solution, three different lookup tables are available encoding the presented thinning criteria by Lee et al. . Each of the provided tables can be used as a basis for the thinning algorithm. Besides the presented lookup tables, users of OpenThinning can create and use their own tables if they match the presented index scheme of the lookup table.

2.2 Algorithm

The OpenThinning solution provides an algorithm, that can be utilized to read a dataset in combination with a lookup table. Based on this input, OpenThinning utilizes an extended version of the presented algorithm by Post et al. [14], which is shown in the pseudocode of Figure 1.

The procedure forms a parallel thinning approach which means, that multiple voxels are deleted in one iteration. Each iteration consists of 6 subcycles, that differ in the direction, the thinning is performed.

At the beginning of the procedure, the border voxels of the dataset are duplicated. This avoids special cases for border voxels, that can slow down the computation. In the algorithm, the neighborhood evaluation is performed solely for voxels of the original volume. In each subcycle iteration all voxels and their local neighborhoods are evaluated using the lookup table. If the current voxel can be deleted, it is stored in a list of possible candidates. The voxels can not be deleted directly as this would result in an incorrect amount of removed voxels in one direction. The result would be a skeleton that is not located in the center of the input structure.

After the list of potential candidates is complete, the algorithm iteratively rechecks the potential candidates. If the lookup table still encodes a deletion, the candidate voxel is finally deleted by setting its value to unfilled. For the rechecks of the remaining potential candidates, the updated image with the previous deleted voxels is considered. The procedure is repeated until no voxel can be deleted in none of the six subcycles. The resulting image holds the skeleton output of the input dataset according to the used lookup table.

3 Results and Discussion

In order to show the effectiveness of OpenThinning, this section presents a comparative study to the widely utilized ITK implementation. The standard implementation, that is provided in ITK Version 4.8 [10, 1] can be applied to a 3D dataset. As the implemented method performs a single thinning procedure in each layer of the three-dimensional input image, it can not be guaranteed, that the connection between the layers is preserved. Instead, the implementation of the ITK Journal, which uses Lee's medial axis thinning, is considered for a comparative study [8]. To obtain fair results, the algorithm was uncoupled from the ITK framework to provide an equal datastructure for the tested approaches. In order to complete the study, the simple- and medial axis approaches of Lee et al. where added to the ITK implementation.

The resulting skeletons can be seen in Figure 3. Therefore, four datasets (box cross, hollow cube, engine [2] and vessels) are thinned with the three thinning algorithms of the OpenThinning solution as well as the ITK implementation. As the results show (see Table 4), the OpenThinning solution is able to perform thinning tasks up to twice as fast as the approach of Lee et al. . The factor of speedup is highly depending of the relative amount of hard to evaluate neighborhoods, where Lee et al's approach needs to check various criteria and the lookup table approach only requires a constant evaluation time.

The implemented thinning algorithm in OpenThinning outputs a correct result, that fulfills the criteria encoded in the lookup tables. The system is designed flexible, as arbitrary lookup tables can be loaded, if they provide the described file format. OpenThinning contains three lookup tables for simple thinning, medial axis thinning [13] and medial surface thinning. As Lee et al. showed, their criteria are able to delete the maximum number of voxels in each iteration correctly. Their implementation in combination with the presented lookup table approach lead to a minimal computational effort, if a parallel algorithm design is not considered.

4 Conclusion

This paper presents OpenThinning, an open source thinning solution using a local moving neighborhood evaluation implemented as lookup tables. The approach is available in an repository for easy access. The system provides three different lookup tables, that can be used to thin binary images and obtain skeletons with different properties. Through the presented lookup table approach and the implemented thinning procedure of Lee et al., which requires a minimal number of iterations, the underlying computation is accelerated.

As future work, a graphical interface is planned, that allows users to drag and drop their input files into the openThinning solution.

References

[1] itk::binarythinningimagefilter. http://www.itk.org/Doxygen/. Accessed: 2016-02-03.

[2] Volovis.org. "http://volvis.org". Accessed: 2016-03-09.

[3] I. Arganda-Carreras. Imagej. "http://imagej.net/Skeletonize3D", July 2016.

[4] D. Bloomberg. Laptonica. "http://www.leptonica.com/", July 2016.

[5] N. D. Cornea, D. Silver, and P. Min. Curve-skeleton properties, applications, and algorithms. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):530–548, May 2007.

[6] GitHub. Zhang-suen thinning. "https://github.com/bsdnoobz/zhang-suen-thinning", July 2016.

[7] R. C. Gonzalez and R. E. Woods. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., 2006.

[8] H. Homann. Implementation of a 3d thinning algorithm. http://hdl.handle.net/1926/1292, 10 2007.

[9] http://freesourcecode.net. Stentiford thinning algorithm in matlab. "http://freesourcecode.net/", July 2016.

[10] H. J. Johnson, M. McCormick, L. Ibáñez, and T. I. S. Consortium. *The ITK Software Guide*. Kitware, Inc., third edition, 2013. *In press*.

[11] C. Kirbas and F. Quek. Vessel extraction techniques and algorithms: a survey. In *Bioinformatics and Bioengineering, 2003. Proceedings. Third IEEE Symposium on*, pages 238–245, 2003.

[12] T.-C. Lee, R. L. Kashyap, and C.-N. Chu. Building skeleton models via 3-d medial surface/axis thinning algorithms. *CVGIP: Graph. Models Image Process.*, 56(6):462–478, 1994.

[13] T. Post and C. Gillmann. Openthinning. "https://github.com/christinagillmann/OpenThinning", July 2016.

[14] T. Post, C. Gillmann, T. Wischgoll, and H. Hagen. Fast 3d thinning of medical image data based on local neighborhood lookups. The Eurographics Association, 2016.

[15] B. Preim and C. P. Botha. *Visual Computing for Medicine: Theory, Algorithms, and Applications*. Morgan Kaufmann Publishers Inc., 2 edition, 2013.

```
function THINNING( $Image$ , $LookupTable$ )
    repeat
        $modified \leftarrow false$
        for $d \in \{up, down, right, left, forward, backward\}$ do
            $Candidates \leftarrow \emptyset$
            for $v \in Voxels$ do
                if $\begin{pmatrix} Image(v) & = & 1 & \wedge \\ Image(v-d) & = & 0 & \wedge \\ LookupTable(v) & = & 1 & \end{pmatrix}$ then
                    $Candidates \leftarrow Candidates \cup \{v\}$
                end if
            end for
            for $c \in Candidates$ do
                if $LookupTable(c) = 1$ then
                    $Image(c) \leftarrow 0$
                    $modified \leftarrow true$
                end if
            end for
        end for
    until $\neg modified$
end function
```

Figure 1: Extended pseudocode for thinning, using arbitrary lookup tables, as shown in the work of Post et al. [14]. As a new feature, OpenThinning offers an implementation of this code as well as three different lookup tables, that can be used.

ACKNOWLEDGEMENTS

This research was funded by the ”German Research Foundation” (DFG) within the IRTG 2057 ”Physical Modeling for Virtual Manufacturing Systems and Processes”. We would like to thank Ghassan Kassabs research team for providing their datasets used in this study.

#	Formula	Usage	Intuition
C_1	$f(v) = 1$	T_1, T_2, T_3	The current voxel has to be filled.
C_2	$f(v-d) = 0$	T_1, T_2, T_3	Considering the current direction of thinning, the predecessor of the voxel is not allowed to be filled.
C_3	$E(v) = 0$	T_1, T_2, T_3	The Euler characteristics of a point needs to be unchanged. This means, that the geometric properties of the thinned object are preserved
C_4	S(v) = 1	T_1, T_2, T_3	The considered point needs to be a simple point. This means, that removing this point, remains the objects topology.
C_5	$\lvert\{n : n \in N_{26}(v) \wedge f(n) = 1\}\rvert > 1$	T_2	The number of neighbors in a local neighborhood of a point needs to be higher than one. This prevents shortening line-like structures and therefore keep their length.
C_6	$\neg\forall i \in \{1...8\} : Index(N_i^2(v)) \in \{153, 165, 170, 195, 204, 240\} \vee \lvert N_i^2(v)\rvert \leq 3$	T_3	The voxel is not allowed to be a medial surface point or the edge of a medial surface while considering the octants in the voxel neighborhood, as shown in Lee et al’s work. The method presents a lookup table deciding weather a voxel belongs to the medial surface or not. The list of matching cases (identified by their number) was extended to obtain correct results.

Table 1: Thinning criteria of the presented lookup tables and their intuition. The different use of these criteria alters the output of the thinning result.

			Medial Surface Thinning				Medial Axis Thinning				Simple Thinning			
Dataset	Size	Voxel Ratio [%]	# Iterations	Voxel Ratio [%]	ITK Thinning [s]	LUT Thinning [s]	# Iterations	Voxel Ratio [%]	ITK Thinning [s]	LUT Thinning [s]	# Iterations	Voxel Ratio [%]	ITK Thinning [s]	LUT Thinning [s]
Box Cross	512 x 512 x 512	23.14	150	0.493	330.6	195.4	125	0.001	275.5	168.9	138	0.000 0007	335.3	194.3
Hollow Cube	513 x 513 x 513	15.15	41	0.442	95.1	58.3	38	0.017	85.2	52.2	38	0.004	81.8	52.2
Engine	256 x 256 x 128	12.60	16	1.309	3.24	1.78	24	0.125	3.98	2.32	23	0.027	3.53	2.14
Vessels	512 x 512 x 199	0.4997	11	0.070	8.02	5.09	7	0.016	4.98	3.29	104	0.002	72.9	45.6

Table 2: Comparison of the OpenThinning approach with the ITK journal implementation. The ITK solution for medial axis thinning was decoupled from the general ITK framework to allow a fair comparison to the OpenThinning solution. It was possible to alter the implementation of the used criteria to allow simple- and medial surface thinning as presented by Lee et al. . As a result, all algorithms can be compared to the OpenThinning solution. The results show, that the lookup table approach implemented in OpenThinning is able to perform thinning tasks up to twice as fast as the implementation of Lee et al.'s approach.

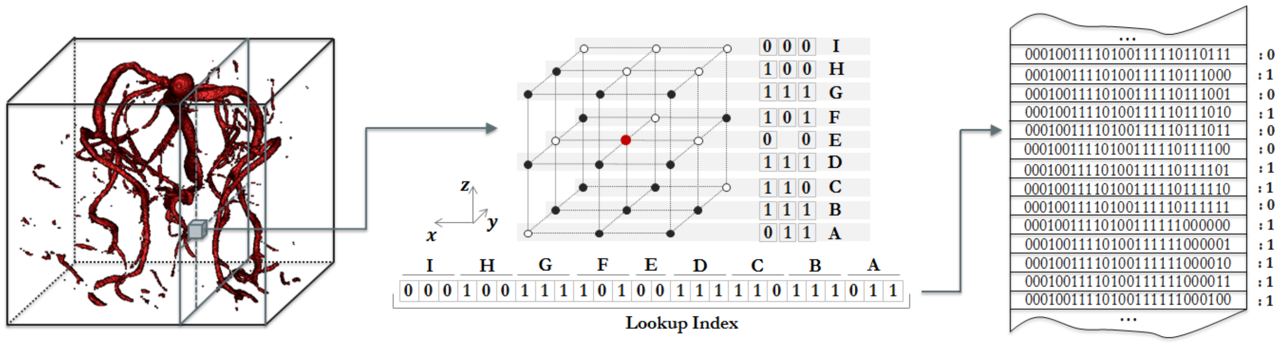

Figure 2: Scheme for neighborhood encoding with example output as shown in the work of Post et al. [14]. The moving local neighborhood evaluates the single voxels of the volume by encoding its neighborhood setting to a lookup index. This index can be used to determine weather a voxel needs to be deleted or not.

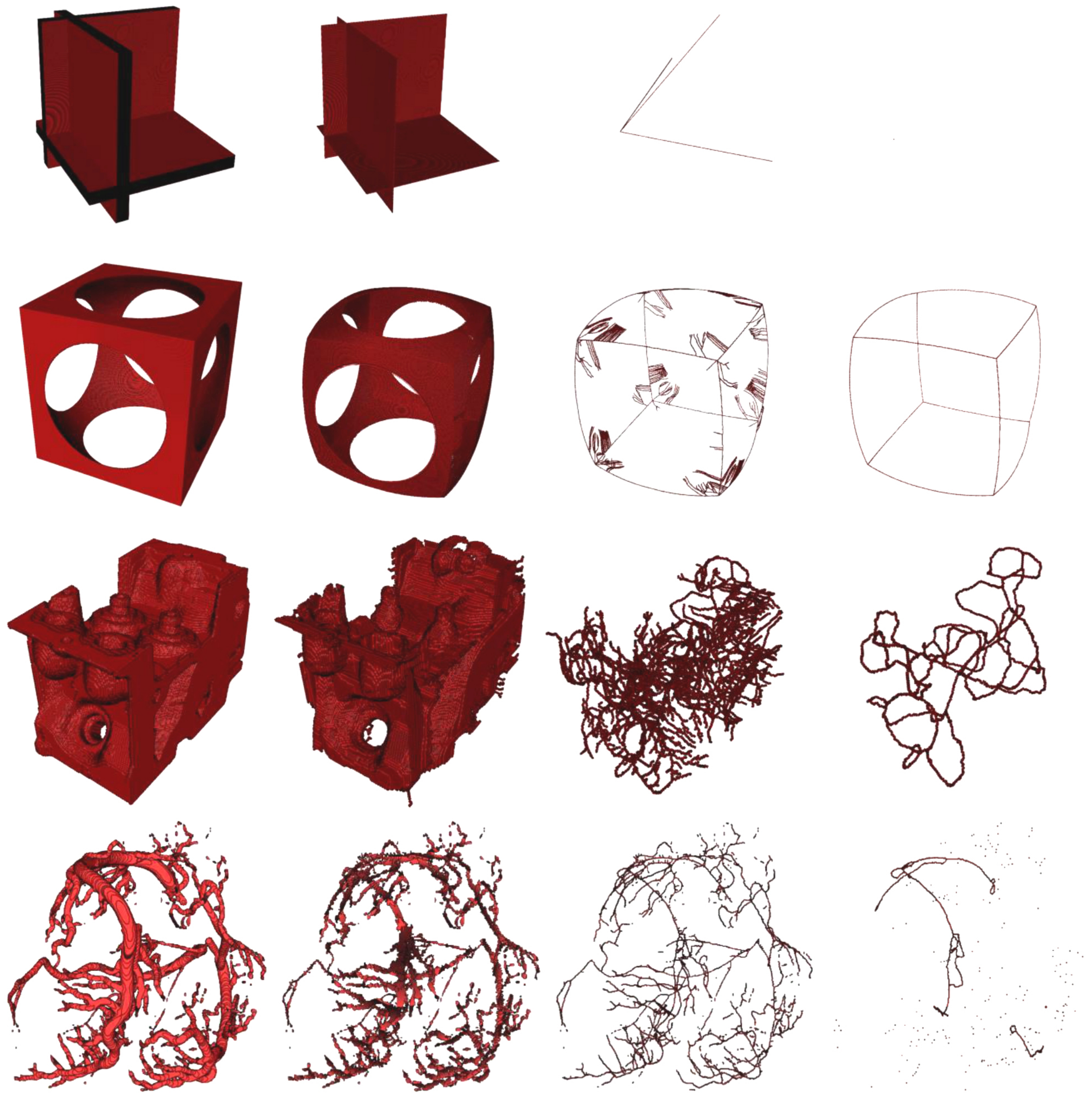

Figure 3: Example datasets and their thinning results. From top to bottom: Box cross dataset, hollow cube, machine engine and vessels. Left to right: original object, medial surface thinning, medial axis thinning and simple thinning. The results show, how the different encoded criteria output altering thinning results. According to the special needs in different applications, the table can be selected. As these examples show, medial surface thinning is beneficial for plate lite structures (e.g. engine), wheres medial axis thinning shows good results for tube like structures (e.g. vessels).