Accelerating Morphological Analysis with Graphics Hardware

Matthias Hopf Thomas Ertl

{hopf,ertl}@informatik.uni-stuttgart.de
http://wwwvis.informatik.uni-stuttgart.de/

Visualization and Interactive Systems Group, University of Stuttgart

Abstract

Direct volume rendering is a common means of visualizing three-dimensional data nowadays. It is, however, still a very time consuming process to create informative and visual appealing images. Semi-automatic volume analysis procedures as morphological operators are a promising approach to improve the overall visualization cycle. However, these operators need quite some time for computation, reducing their usefulness for interactive visualization.

Modern graphics hardware, on the other hand, has all necessary functions for doing hardware based morphological filtering. As the problem is mainly memory bandwidth bound, a solution based on graphics hardware can significantly reduce computation time in the filtering step, as the graphics hardware typically has much broader and faster memory paths.

When using graphics hardware for mathematical computations, accuracy is usually quite a problem. However, morphological operators map so well onto the graphics pipeline, that no loss of accuracy occurs at all.

1 Introduction

Direct volume rendering is a very important technique for visualizing three-dimensional data. Several fundamentally different methods have been introduced [1, 6, 7, 8, 10, 15]. Hardware-based volume texturing [11, 16] is one of the most prominent variants for interactive visualization due to the high frame rates that can be achieved with this technique.

Before a given data set is rendered, several filtering and mapping steps can be applied to the data in the visualization pipeline (Figure 1) in order to expose prominent features more precisely. Even with the broad knowledge about filtering techniques we have today, for volume rendering this only means adapting the mapping step in most cases. The proper adjustment of the according functions is one of the major problems for the practical use of volume visualization. This is commonly accomplished by the interactive generation of a more or less complicated transfer function. More complex approaches for the lighting model are beyond the capabilities even of modern graphics hardware and therefore not very useful for interactive volume rendering.

Alternatively, the volume data can be preprocessed in order to reveal certain features more properly with relatively simple transfer functions. This approach requires fewer input parameters from the user during analysis and rendering, simplifying the visualization cycle. One promising approach for semi-automatic frequency based volume analysis are morphological operators. Despite all improvements in processor technology one serious drawback of this approach remains. Almost all operators working on three-dimensional data are computational complex, reducing their usefulness for interactive visualization.

The disadvantage of wavelet transformation and other filters, that are based on linear combinations of the input data, is the fact that the

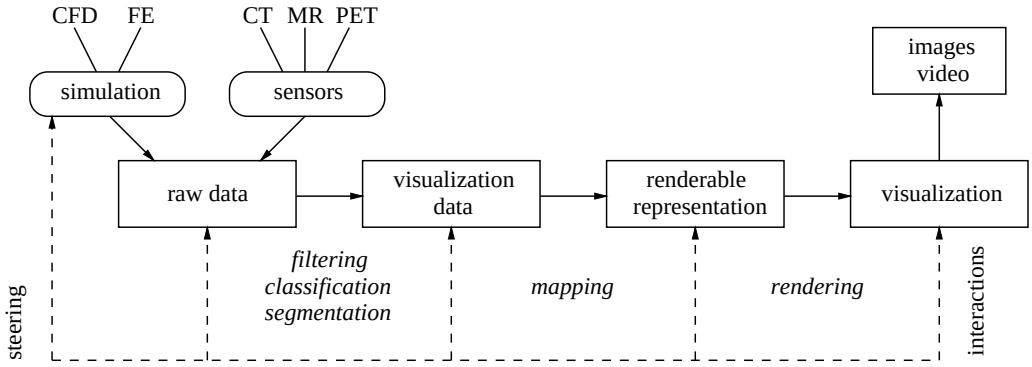


Figure 1: The visualization pipeline.

analyzed structures become distorted. This can be seen especially on lower frequency scales, accomplished by large filter kernels. One class of special non-linear filters that do not flatten the contours of the original data set are the morphological operators. This filter type computes the minimum respectively maximum of pixel values within a given scope. Before the actual maxima are calculated, the values of the so-called structuring element are added to the pixel values. [12] gives a more thorough introduction to gray-scale morphology, while we will give only a short overview.

Morphological operators are mainly used in pattern recognition to perform some kind of presegmentation filtering. Often they are used in combination with region growing methods [2]. Recently they have gained interest in visualization groups for processing raw data in order to reveal certain structures. One approach uses structuring elements of different sizes in order to do some kind of a hierarchical analysis, determining areas of different spatial frequencies of the data set [9]. The filtering process takes two minutes and more for big data sets, despite the efforts to parallelize the operations.

As we have already shown the feasibility of implementing graphics hardware accelerated three-dimensional separable filters [3] and wavelet decomposition and reconstruction [4, 5], we will now address morphological operators. With the ability of almost all graphics accelerators to use a minimum/maximum blending function while rendering textured triangles to the frame buffer, these operators can be mapped perfectly onto the hardware pixel path, accelerating the time consuming filtering steps while retaining the volume data in the texture memory of the graphics hardware for the following visualization step. That way the volume data does not have to be reloaded for consecutive visualization steps, as it is the case with software based approaches.

2 Morphological Operators

Morphological operators are a special form of non-linear filters that are capable to separate or to combine different objects in an image with a minimal distortion of the contours. At first, these operators are defined on binary images only. Let X and Y be two binary data sets of dimension d. Then the erosion $Z := X \ominus Y$ is defined as

$$Z_i = \begin{cases} 0: & \exists j:\ Y_j = 1 \ \wedge\ X_{i-j} = 0 \\ 1: & \text{otherwise} \end{cases} .$$

The other basic operator, the dilation operator $Z := X \oplus Y$ is defined as the dual operator to the erosion:

$$Z_i = \begin{cases} 1: & \exists j:\ Y_j = 1 \ \wedge\ X_{i+j} = 1 \\ 0: & \text{otherwise} \end{cases} .$$

As one can see in Figure 2, the erosion operator cuts away parts of the boundary of the analyzed image X. The amount that is removed is defined by the structuring element Y, which is typically much smaller than the input image. The dilation operator on the other hand enlarges the input data.

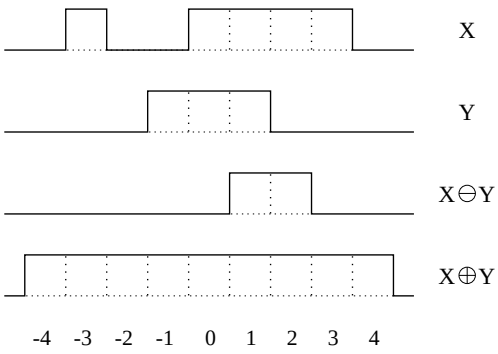


Figure 2: Results of the binary erosion and dilation operators.

By combining these basic operators to more complex ones we get the so-called opening and closing operators. The opening operator breaks up small bridges between connected regions while the closing operator tends to fill small gaps in solid components. The opening operator is defined as

$$X \bigcirc Y := (X \ominus Y) \oplus Y$$

and the closing operator is defined as

$$X \circ Y := (X \oplus Y) \ominus Y$$

Gray-scale morphological operators, which are used in our algorithms, are defined by transforming the gray-scale data to an binary data set with an extra dimension, representing the gray level. This lifting into the extra dimension can be done implicitly by defining according dilation and erosion operators for gray-scale data. This way we get the dilation operator $Z := X \oplus Y$ as

$$Z_i = \max_j \{X_{i-j} + Y_j\} \tag{1}$$

and the erosion $Z := X \ominus Y$ accordingly as

$$Z_i = \min_j \{X_{i+j} - Y_j\} \quad . \tag{2}$$

In the binary forms of the operators, the parts of the structuring element Y with $Y_j = 0$ have no effect at all, and thus we will call them neutral elements. In gray-scale morphology this corresponds to parts of the structuring element with $Y_j = N := -\infty$, because in the way they are used they are invariants to the maximum and minimum operators. In the following we will assume that all structuring elements have only a finite size, i. e. all values outside a given domain are equal to N.

When we remember that all input data as well as the structuring element is bound for problems related to image and volume data sets to a fixed domain $[0,m]$, we can define $\bar{Y}_j = m - Y_j$, and using this we can shift the range of Eq. (1) and (2) to $[0,m]$ as well by using

$$Z_i = \max_j \{X_{i-j} - \bar{Y}_j\} \tag{3}$$

for the dilation and

$$Z_i = \min_j \{X_{i+j} + \bar{Y}_j\} \tag{4}$$

for the erosion operator, provided, that there exists a j_m with $Y_{j_\mathrm{m}} = m$. This requirement assures, that $X_{i-j_\mathrm{m}} - \bar{Y}_{j_\mathrm{m}} \geq 0$, because $\bar{Y}_{j_\mathrm{m}} = 0$. Additionally, $X_{i-j} - \bar{Y}_j \leq m$ holds for all i, j, because $\bar{Y}_j \geq 0$. The neutral element in this case is $\bar{N} := m$, because $X_j - m \leq 0$ and is thus not contributing to the maximum value in Eq. (3). Equivalent inequalities hold for the erosion operator as well.

As the range of the morphological operators can be shifted to the domain range and only integer operations are needed during computation, it is clear, that these operators can be implemented using graphics hardware without precision loss, when the equations can be mapped onto the graphics pipeline.

The most problematic aspect of morphologic volume analysis are the high computational costs of dilation and erosion operators, when they are invoked for volume data. For structuring elements of size n, $2n^3$ data values have to be addressed per voxel. The general approach to cope with this efficiency problem is to decompose a large structuring element into several smaller ones. This is possible because the dilation and erasion operators obey the following relations as described in [17]:

$$\begin{aligned} X \ominus (Y \oplus Z) &= (X \ominus Y) \ominus Z \\ X \oplus (Y \oplus Z) &= (X \oplus Y) \oplus Z \end{aligned}$$

When a decomposition of a large structuring element into several smaller ones can be found,

the morphological operation with the large element can be accomplished by the consecutive application of the smaller ones:

$$\begin{aligned} S &= H_1 \oplus H_2 \oplus \cdots \oplus H_n \quad , \\ X \ominus S &= (((X \ominus H_1) \ominus H_2) \cdots) \ominus H_n \quad , \\ X \oplus S &= (((X \oplus H_1) \oplus H_2) \cdots) \oplus H_n \quad . \end{aligned}$$

With this technique a diamond shaped structured element of the form

$$Y_{j_1,\ldots,j_d} = c_1 \cdot \left(c_2 - \sum_k |j_k| \right)$$

with constants c_1 and c_2 can be decomposed into several smaller diamond shaped structures. For instance, if we decompose a structuring element of size n into several smaller elements of size 3, only $54n$ data values have to be addressed per voxel for one basic morphological operation.

But there is still the better possibility to decompose the structuring element into several one-dimensional elements in a tensor product like fashion: $Y = ((Y^1 \oplus Y^2) \cdots) \oplus Y^d$ with

$$Y^i_{j_1,\ldots,j_d} = \begin{cases} c_1 \cdot (\frac{c_2}{d} - |j_i|) & j_k = 0 \ \ \forall\, k \neq i \\ N & \text{otherwise} \end{cases} \quad (5)$$

for which an example is depicted in Figure 3.

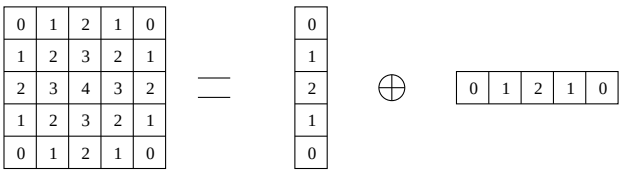


Figure 3: Example decomposition of a structuring element of size 5.

The structuring elements of the range shifted operators Eq. (3) and (4) can be decomposed similarly.

That way the computational complexity can be reduced to $6n$ addressed data cells per voxel. This is the approach we will continue to investigate in this paper, though the basic algorithms are in no way restricted to this type of decomposition. For more information about structuring element decomposition we refer to [17].

3 Hardware Environment

The basic principle of texture based volume rendering is depicted in Figure 4. According to the sampling theorem a 3D view of the volume is generated by drawing an adequate number of equidistant, semi-transparent polygons parallel to the viewing plane ("volume slicing").

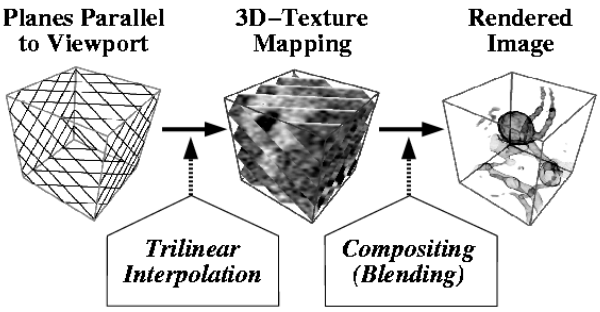


Figure 4: Texture based volume rendering.

Most PC based graphics cards do not support 3D textures. On these cards a slicing approach is popular for volume rendering, that renders textured planes perpendicular to the volume axes, very much the way shear-warp algorithms work. With this method only a set of 2D textures is necessary for volume visualization. OpenGL 1.1 supports so-called texture objects, which allows switching between several texture maps in almost no time, as long as all images fit into the texture memory of the graphics card. Without loss of generality we assume, that the texture planes contain data information perpendicular to the z axis.

We have implemented both approaches for our algorithm, but a serious bug in partial 3D texture reloading on a number of machines effectively disables the possible use right now. Therefore, we will concentrate on the 2D texture solution in our paper.

In order to map morphological operators onto the graphics pipeline (see Figure 5), we first have to find out where minimum and maximum operators are supported. It turns out, that only the per-fragment operations can perform minimum and maximum blending in the frame buffer, when the according OpenGL extension is supported by the graphics driver.

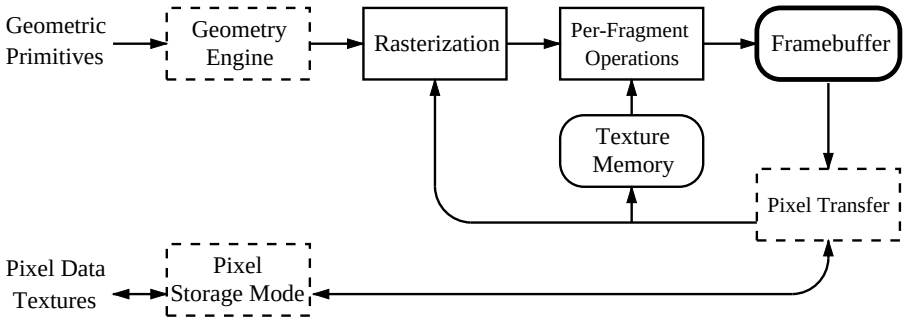


Figure 5: The OpenGL graphics pipeline.

4 Hardware Based Operators

As the basic algorithmic overview in Figure 6 shows, we can use the structure of the structuring element decomposition Eq. (5) to sweep over the volume in three passes. Note that our approach can also be applied to morphological filters, that cannot be decomposed into several one-dimensional filters. However, the necessary computation time usually prohibits the usage of such operations. Nevertheless, the possible speedup would be even higher in these cases.

In the first two passes we perform the filtering along the x and y axes. These steps only access data from within one single texture, which contains data perpendicular to the z axis, as defined in the previous section. In the third pass we filter along the z axis. As the data along this axis is spread in multiple textures, triangles textured with several different images have to be rendered. In order to minimize texture binding changes, we combine the first two passes.

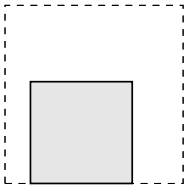
In every pass textured triangles that are translated by several pixels along the filtering axis are rendered into the frame buffer. There the incoming fragments are blended with the pixel data that is already contained in the buffer. By using the minmax blending extension the compositing step effectively calculates the maximum or minimum value of all texels within the filter scope. After a one-dimensional filter operation has been performed, the resulting image is read back from the frame buffer into texture memory.

In order to perform the dilation Eq. (3), we have to subtract a per polygon constant value from the fragments after texturing and before blending. This can be accomplished on Silicon Graphics systems by using the texture color lookup table with different lookup tables that only contain linear ramps, shifted by the according structuring element values. The lookup table has to be changed for every rendered polygon, therefore we accelerate the loading process by using predefined lookup tables contained in display lists. On NVidia systems, the same effect can be accomplished with the paletted texture extension, or even better with the register combiners extension. Unluckily, the copy operation from the frame buffer into texture memory is currently not accelerated at all, yielding uninteresting filter rates. However, with recent advances in the driver technology, especially for linux systems, we are confident, that future graphics chips and driver versions from NVidia will clearly outperform the current filter rates, that have been measured on an SGI Octane system.

In order to save texture memory, which is a scarce resource, we want to do the filtering in place, replacing the previously used textures. In the first pass this is accomplished implicitly, because we perform the filter operations for all pixels along the filter axis in one step. Therefore, we do not need the original data any more after the operator was invoked.

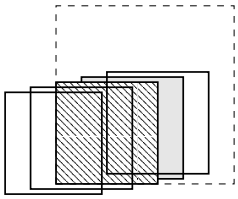
In the second pass we cannot perform the filter operations for all pixels along the filter axis in one step, as this involves several textures, and we cannot copy back frame buffer contents to multiple textures in one step. Therefore, we either have to use a second set of textures, which we wanted to avoid, or we have to store the resulting data to parts of the volume we will not address any more. As we are dealing with usually small or at least finite filters, we can see that we

1. Pass: For all z:
   - Clear frame buffer, select texture z
   - For all j: $\bar{Y}_j \neq \bar{N}$
     - Render textured polygons, shifted by (j,0), biased by $\bar{Y}_j$



(Example for j=1)

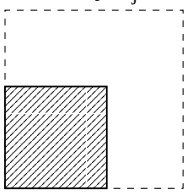
     - Blend results in the frame buffer to calculate minimum/maximum



(The shift along the y axis in this diagram is only used to clearify the situation)

   - Copy indicated region back to texture z
   - For all j: $\bar{Y}_j \neq \bar{N}$
     - Do the same, this time shifted by (0,j)

2. Pass: For all z:
   - Clear frame buffer
   - For all j: $\bar{Y}_j \neq \bar{N}$
     - Render polygons textured with texture z+j, biased by $\bar{Y}_j$



     - Blend results in the frame buffer to calculate minimum/maximum
   - Copy indicated region back to texture $z - j_{neg}$

Figure 6: The basic algorithmic structure.

need j_{neg} spare textures, with

$$j_{\text{neg}} = \max\left\{ \min_j \left\{ j : \; Y_j \neq N \right\}, \; 0 \right\}$$

being the number of coefficients the structuring element extends to negative indices. Figure 7 shows an example, where a structuring element of size 3 is used, with the filter coefficients $\bar{Y}_{-1}$, $\bar{Y}_0$ and $\bar{Y}_1$. If no precautions had been taken, the textures that are needed in the next step would be overwritten by values from the frame buffer, which is indicated in the figure by dashed lines.

We can optimize the basic algorithm even more by removing the frame buffer clear command. When we render the first polygon of the inner loop with blending disabled, it will implicitly clear the relevant part of the frame buffer, and rendering polygons without blending is faster anyway. We just have to be sure, that the first polygon rendered is the one related to $\bar{Y}_0$, because it is drawn exactly at the position of the image to be read back after the filtering is complete (compare Figure 6).

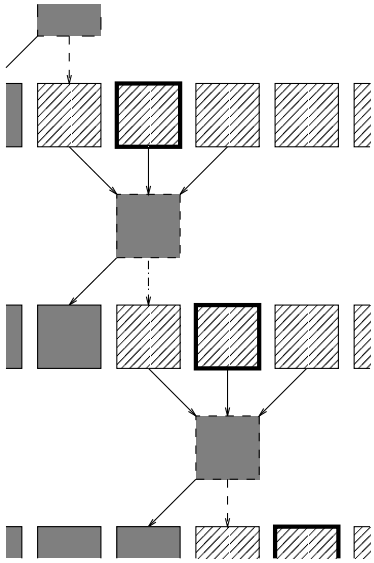


Figure 7: Handling texture copy overlaps.

For many applications of morphological operators, the same data set has to be filtered several times, using different operators of different sizes. When using the graphics hardware approach, it is clear that we can use the color channels to perform three or four operators in one filtering step, provided, that we use the biggest morphological operator as the basic filter, combine it with the other operators and fill the remaining empty slots with the neutral element $\bar{N}$.

5 Results

Table 1 reveals that hardware based morphological operators are much faster than well tuned software implementations. Compared with [9] they easily beat multiprocessor implementations as well. Admittedly, the times measured there include two basic morphological operators with four different elements and the additional overhead for difference signal creation, thresholding

Data set	256^3				$512^2 \times 154$			
System	ix86[1]	mips[2]	OpenGL[3]	Speedup[4]	ix86[1]	mips[2]	OpenGL[3]	Speedup[4]
3^3	13.4	27.3	5.7	4.8×	33.3	67.7	11.8	5.7×
7^3	24.0	32.7	10.1	3.2×	58.8	101.5	19.5	5.2×
15^3	43.5	46.3	18.8	2.5×	104.2	172.3	34.3	5.0×
25^3	62.8	90.6	29.5	3.1×	149.9	251.2	54.5	4.6×

[1] Measured on a standard PC with Pentium III, 500 MHz
[2] Measured on a SGI Octane with R10000, 250MHz
[3] Measured on a SGI Octane MXI
[4] Speedup on SGI Octane, hardware vs. software based

Note that the hardware accelerated OpenGL method can additionally perform three different morphological operations in one filtering step on the MXI graphics system.

Table 1: Times in seconds per basic operator.

and transfer function invocation. But with hardware based operators, three or even four (when RGBA texture lookup tables are supported) filters can be applied in one cycle, reducing comparable times even more. By performing three different morphological operations in one cycle, we get filtering speedups of 15 times and more for large data sets and small to mid-sized filters. Note that all methods work on 8 bit data structures in order to minimize memory consumption and to speed up data access.

The Intel based systems seem to be superior to this kind of image processing, and modern graphics cards for PC systems have high fill rates that can perform morphological operators extremely fast. The performance of the software based method is not processor architecture depended, though, as a much faster Athlon processor produced inferior results. The kind and size of caches and the main board chip set have a much higher influence on the results than the processor speed. On the hardware accelerated side, the most promising systems today that support the necessary OpenGL extensions are based on the NVidia GForce256 chip and its successors.

Unfortunately, despite recent improvements in NVidias drivers, copying data from the frame buffer to texture memory is still not accelerated, ruining the otherwise fine rendering rates that come from the high fill rates of GForce256 DDR graphics cards. Therefore, we have desisted from printing test results of this system.

Mathematical computations in the frame buffer are usually susceptible to accuracy loss due to the limited frame buffer depth [5, 13, 14]. As morphological operators only need integer operations and the range of the results does not exceed the domain, no precision loss occurs at all. The correctness of our implementation has been verified by several tests, of course, yielding the desired results. Figures 8 to 11 show sample result slices of the opening and closing operator working on a dataset of size 256^3, using a filter of size 7^3.

6 Conclusion

We have introduced a OpenGL hardware based algorithm for morphological operators, that utilizes the high fill rates of modern graphics hardware for accelerating time consuming 3D filtering up to 15 times and more for large data sets. Because of the structure of the operators, no loss of precision occurs at all, making this method attractive for all analysis performing morphological operations.

The approach that has been taken can be used for non-decomposed morphological operators and for non-separable linear filters as well, although these kinds of filters are not as often

used as the ones we have already addressed. These and other types of filtering systems are subject of future work.

PC based graphics systems almost catched up with high end workstations in the last few years. However, the drivers are still very much optimized for gaming purposes, and several features that would yield very interesting options for scientific visualization are still not accelerated. But we are confident that this will change in the very near future.

References

[1] R. A. Crawfis and N. L. Max. Texture Splats for 3D Scalar and Vector Field Visualization. In G. M. Nielson and Bergeron D., editors, *Visualization 93*, pages 261–265, Los Alamitos, CA, 1993. IEEE Computer Society, IEEE Computer Society Press.

[2] K.-H. Hoehne and W. A. Hanson. Interactive 3D Segmentation of MRI and CT Volumes using Morphological Operations. *Journal of Computer Assisted Tomography*, 16(2):285–294, 1992.

[3] M. Hopf and T. Ertl. Accelerating 3D Convolution using Graphics Hardware. In D. Ebert, M. Gross, and B. Hamann, editors, *Visualization '99*, pages 471–474, San Francisco, CA, 1999. IEEE Computer Society, IEEE Computer Society Press.

[4] M. Hopf and T. Ertl. Hardware Based Wavelet Transformations. In B. Girod, H. Niemann, and H.-P. Seidel, editors, *Vision, Modeling, and Visualization '99*, pages 317–328, Erlangen, Germany, November 1999. SFB 603, Graduate Research Center, IEEE, and GI, Infix Press.

[5] M. Hopf and T. Ertl. Hardware Accelerated Wavelet Transformations. In *Symposium on Visualization VisSym '00*. EG/IEEE TCVG, IEEE Computer Society Press, May 2000.

[6] A. Kaufman. *Volume Visualization*. IEEE Computer Society Press, 1991.

[7] P. Lacroute and M. Levoy. Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. In *Computer Graphics Proceedings*, Annual Conference Series, pages 451–457, Los Angeles, California, July 1994. ACM SIGGRAPH, Addison-Wesley Publishing Company, Inc.

[8] L. Lippert, M. H. Gross, and C. Kurmann. Compression Domain Volume Rendering for Distributed Environments. In D. Fellner and L. Szirmay-Kalos, editors, *EUROGRAPHICS '97*, volume 14, pages C95–C107. Eurographics Association, Blackwell Publishers, 1997.

[9] C. Lürig and T. Ertl. Hierarchical Volume Analysis and Visualization Based on Morphological Operators. In *Proc. IEEE Visualization '98*, pages 335–341, 1998.

[10] P. Schröder and G. Stoll. Data Parallel Volume Rendering as Line Drawing. In *1992 Workshop on Volume Visualization*. ACM SIGGRAPH, October 1992.

[11] SGI. Volume Rendering using RE2 Technology. Technical report, SGI, 1994.

[12] S. Steinberg. Grayscale Morphology. In *Computer Vision, Graphics and Image Processing*, pages 333–355, 1986.

[13] C. Teitzel, M. Hopf, and T. Ertl. Volume Visualization on Sparse Grids. *Computing and Visualization in Science*, (2):47–59, 1999.

[14] C. Teitzel, M. Hopf, and T. Ertl. Scientific Visualization on Sparse Grids. In H. Hagen, G. M. Nielson, and F. Post, editors, *Proceedings of Scientific Visualization - Dagstuhl '97*, pages 284–295, Heidelberg, 2000. IEEE Computer Society, IEEE Computer Society Press.

[15] T. Totsuka and M. Levoy. Frequency Domain Volume Rendering. *Computer Graphics*, 27(4):271–78, August 1993.

[16] R. Westermann and T. Ertl. Efficiently Using Graphics Hardware in Volume Rendering Applications. *Computer Graphics (SIGGRAPH '98)*, 32(4):169–179, 1998.

[17] X. Zhuang and R. Haralick. Morphological Structuring Element Decomposition. In *Computer Vision, Graphics and Image Processing*, pages 370–382, 1986.

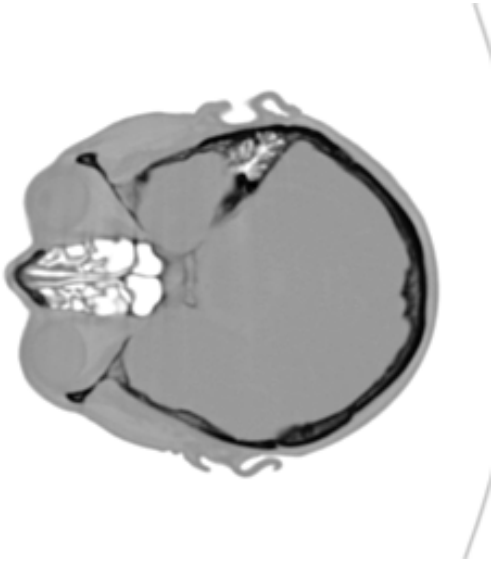


Figure 8: The original data set.

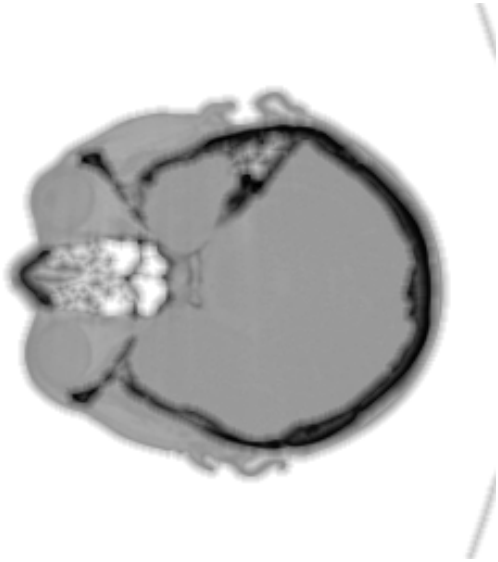


Figure 9: The opened data set.

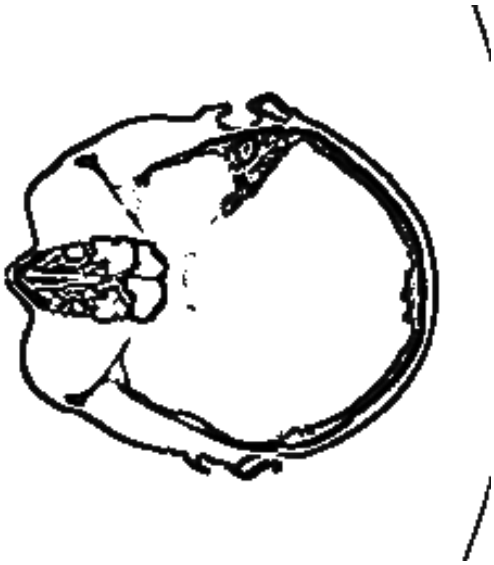


Figure 10: Difference to closed data set.

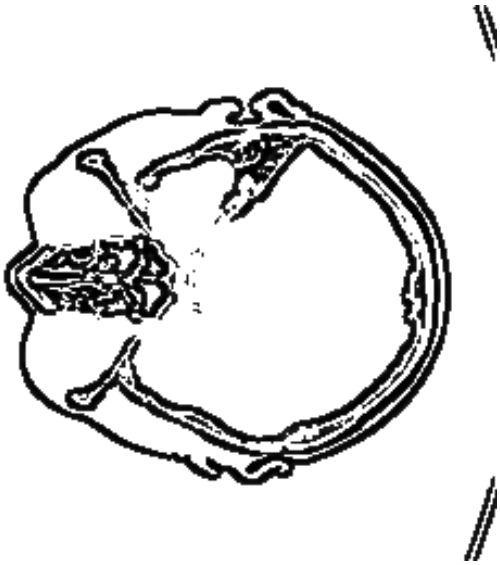


Figure 11: Difference to opened data set.