Accelerating 3D Convolution using Graphics Hardware

Matthias Hopf Thomas Ertl

Visualization and Interactive Systems Group, IfI, University of Stuttgart

Abstract

Many volume filtering operations used for image enhancement, data processing or feature detection can be written in terms of three-dimensional convolutions. It is not possible to yield interactive frame rates on todays hardware when applying such convolutions on volume data using software filter routines. As modern graphics workstations have the ability to render two-dimensional convoluted images to the frame buffer, this feature can be used to accelerate the process significantly. This way generic 3D convolution can be added as a powerful tool in interactive volume visualization toolkits.

Keywords: Convolution, Hardware Acceleration, Volume Visualization

1 Introduction

Direct volume rendering is a very important technique for visualizing three dimensional data. Several fundamental different methods have been introduced [2, 4, 5, 6, 8, 12]. Hardware-based volume texturing [9, 14] is one of the most prominent variants for interactive visualization due to the high frame rates that can be achieved with this technique.

The basic principle of texture based volume rendering is depicted in Figure 1. 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 image plane with respect to the current viewing direction (“volume slicing”).

Filtering on the other hand is a major part of the visualization pipeline. It is broadly used for improving images, reducing noise, and enhancing detail structure. Volume rendering can benefit from filter operations, as low pass filters reduce the noise e.g. of sampled medical volume images and high pass filters can be used for edge extraction, visualizing prominent data features. Multiscale approaches as [13] regularly use disjunct filtering and downsampling steps and can benefit from any speedups of the filtering process.

{hopf,ertl}@informatik.uni-stuttgart.de

Institut für Informatik
Breitwiesenstr. 20 – 22
70565 Stuttgart
Germany

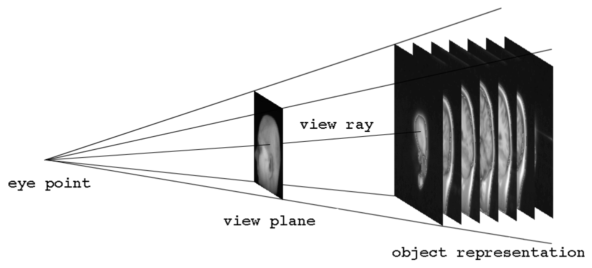

Figure 1: Texture based volume rendering

Filters can be classified as linear or non-linear. Discrete linear filters can be written as convolutions with filter kernels that completely specify the filtering operation. Non-linear filters, as for instance morphological operators, were recently used for volume analysis and visualization [7]. Segmentation and classification depend heavily on filtering operations as well. Bro-Nielson [1] already thought about using convolution hardware for accelerating the registration process.

For texture based volume rendering the data set has to be loaded into special texture memory, which can be addressed by the graphics pipe very fast. The loading process itself is relatively slow, taking several seconds for big data sets even on the fastest available graphics workstations. As the data set has to be reloaded after a filter operation has been performed in software, interactive filtering will benefit a lot from convolution algorithms that directly work on the texture hardware. Additionally, we will show in the following that computing the convolution with graphics hardware is much faster than software solutions.

2 3D Convolution

The general three-dimensional discrete convolution can be written as

$$\tilde{f}(x,y,z) = \sum_{i_1,i_2,i_3} k(i_1,i_2,i_3) \cdot f(x+i_1, y+i_2, z+i_3)$$

with f being the input data function and k being the filter kernel, resulting in the convoluted data $\tilde{f}$.

In the following examination we will assume without loss of generality that $k(i_1,i_2,i_3)$ is given for $0 \leq i_1,i_2,i_3 < n$ and vanishes outside this interval. Also, we will assume that the input data function vanishes for (x,y,z) outside the interval $[0,m)^3$.

In the special case of $k(i_1,i_2,i_3) = \bar{k}_1(i_1) \cdot \bar{k}_2(i_2) \cdot \bar{k}_3(i_3)$ the

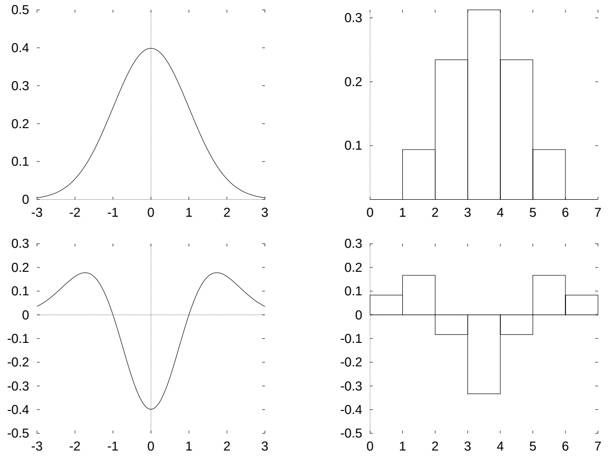

Figure 2: The Gauß filter function and its second derivative

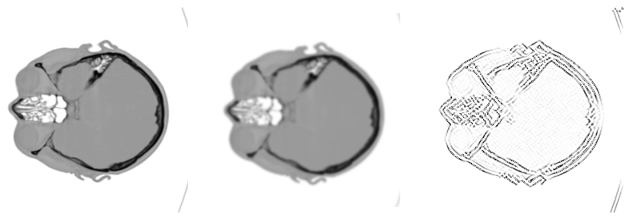

Figure 3: Example image; original, filtered with Gauß, and filtered with second derivative

kernel is called separable. In this case the number of operations necessary for the convolution can be reduced down to $O(m^3n)$, from $O(m^3n^3)$ in the non-separable case:

$$\tilde{f}_1(x,y,z) = \sum_{i_1} \bar{k}_1(i_1) \cdot f(x+i_1,y,z) \quad (1)$$

$$\tilde{f}_2(x,y,z) = \sum_{i_2} \bar{k}_2(i_2) \cdot \tilde{f}_1(x,y+i_2,z) \quad (2)$$

$$\tilde{f}(x,y,z) = \sum_{i_3} \bar{k}_3(i_3) \cdot \tilde{f}_2(x,y,z+i_3) \quad (3)$$

Of course special care has to be taken near the boundaries of the input data function, as convolution routines are generally written on a very low language level for speed purposes.

Figure 2 shows two well known convolution filters, the Gauß filter and its second derivative, both in their continuous and discrete forms. They can be used for noise reduction and edge detection, respectively. An example image that was filtered with these kernels can be seen in Figure 3.

3 Hardware Acceleration

In order to accelerate the convolution process, special purpose hardware can be used. On systems that have built-in Digital Signal Processors (DSPs), for example for multimedia acceleration, a specialized convolution subroutine could be downloaded to the signal processor. On the other hand, most times these DSPs are not well documented or the run-time system can not be modified by the user. In general they are not faster than the main processor as well. Additionally, there exists a wide range of different DSP systems, all of which are incompatible to each other.

The approach we have implemented in our system is to combine a 2D and a 1D convolution kernel in order to calculate three-dimensional separable convolutions. Several vendors of the graphics API *OpenGL* — as for example Silicon Graphics Inc. [10] — have included extensions for one- and two-dimensional filtering. In contrast to most implementations that emulate these extensions only in software, the SGI graphics pipes *MXE* and *BasicReality* calculate the convolutions on-board, boosting performance by an order of magnitude already for reasonably sized filters. The *CRM* graphics system of the O_2 is capable of rendering convolutions in hardware as well, but it does not support volume textures, which are crucial for the algorithm as well.

Recall that the volume data is already stored in texture memory for visualization using texture hardware. Now (1) and (2) are combined to one 2D convolution that is to be applied to every plane of the volume data perpendicular to the z-axis. Therefore, plane by plane is drawn by rendering textured triangle strips (two triangles per strip) into the frame buffer as it is sketched in Figure 4. The texture coordinates assigned to the vertices of the triangle strips are specified in such way that no interpolation of the texture is necessary (see Figure 5). In order to increase the potential speedup and to avoid rounding problems, nearest neighbor interpolation is activated during the rendering process. Each plane is then read back with the OpenGL routine `glCopyTexSubImage3DEXT`, which replaces one plane of the active volume texture orthogonal to the z-axis by data that is read directly from the frame buffer. While transfering the data to the texture memory, the separable 2D convolution filter is activated using `glSeparableFilter2DEXT`. After this first pass, the volume texture contains data filtered along the x- and y-axes.

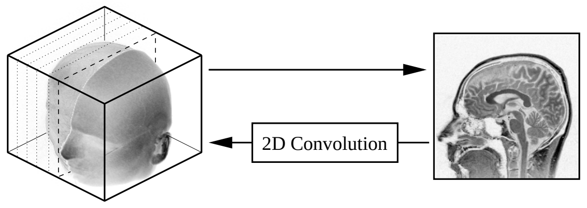

Figure 4: The first pass of the hardware filtering algorithm

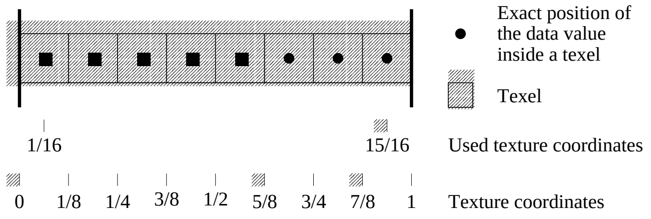

Figure 5: Texture coordinates used for exact texel hits

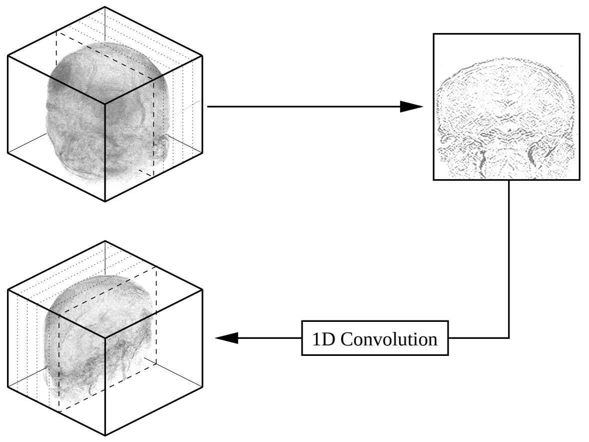

Figure 6: The second pass

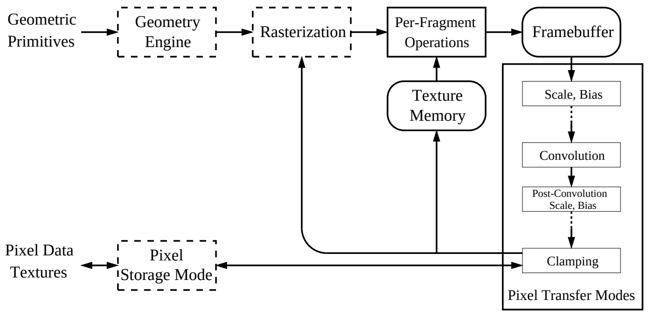

Figure 7: The OpenGL graphics pipeline

Applying the convolution to the third axis is more complicated and depicted in Figure 6. In this second pass planes are rendered perpendicular to one of the other axes. Assume that the y-axis has been chosen. As `glCopyTexSubImage3DEXT` can not write planes orthogonal to any other axis than the z-axis to the texture memory, they have to be transfered to a second volume texture. OpenGL's texture objects are used by calling `glBindTexture` for switching between them, which implies only a very small overhead. While copying the data from the frame buffer to the texture memory, a one-dimensional convolution filter is activated. As we are dealing with two-dimensional image data, we specify a 2D convolution filter with `glConvolutionFilter2DEXT`, using a filter kernel that is exactly one pixel wide.

After this second pass the convoluted volume data has been mirrored at the plane $y - z = 0$. For texture based volume renderers this does not impose any restrictions, as they only have to swap coordinates during texture coordinate calculations. When data order is crucial for the application, the algorithm of pass two can be used for both passes, thus restoring the data order in the second pass. However, the textured planes have to be drawn two times perpendicular to the y-axis. Cache misses are much more likely in this case in respect to planes rendered orthogonal to the z-axis. This can increase the convolution times on big volumes by up to 50% even on fast graphics hardware.

Figure 7 depicts the relevant part of the OpenGL pipeline. It reveals, that pixel fragments read from the frame buffer are clamped to $[0, 1)$ before they can be written back to the frame buffer or into the texture memory. Filter kernels with negative coefficients can compute negative intermediary values during the two-dimensional convolution pass, which will not contribute to the final 1D convolution. Additionally, no negative results can be stored in the output volume. These values are especially needed when the filter kernel is not symmetrical.

The strategy for avoiding these effects depends on the particular application. For edge detection the absolute maxima of the filtered volume data are of interest. In this case, calculating the absolute value can be performed in hardware as well, further reducing necessary computations on the CPU. In most other cases post-convolution scaling and bias can be used to map the expected results to the interval $[0, 1)$ just before the clamping takes place. OpenGL provides the `GL_POST_CONVOLUTION_c_SCALE_EXT` and `GL_POST_CONVOLUTION_c_BIAS_EXT` parameters, which are applied to pixel color values after convolution and before clamping as depicted in Figure 7.

4 Results

The used data sets are presented on the color plate. Figures 8 to 12 show slices of a head data set of size 128^3. Figure 8 reveals the unfiltered data set, whereas Figure 9 and 10 present slices of the software and hardware low pass filtered volume data, respectively. Here, a Gauß filter kernel of size 5^3 has been used. Almost no differences can be detected. Figure 11 and 12 depict the results for high pass filtering using the second derivative of the Gauß filter, again computed in software and in hardware. The hardware convoluted volume displays noticeable artifacts that occur due to the already mentioned clamping step in the OpenGL pipeline. By using post-convolution scaling and bias the artifacts disappear completely.

The Figures 13 to 18 picture another data set that has been used for testing the implementation. They have been visualized with the hardware based volume rendering toolkit TiVOR [11], again with the first picture being rendered with the original data set. While the noise reduction effect of the Gauß filter is rather bothering in Figure 14 by smearing volume details, it has remarkable positive effects on ISO surface generation (compare Figures 17 and 18). Please remark that the ISO surfaces are rendered in real-time using a hardware accelerated volume rendering approach described in [14].

Noise interferes with high pass filters, which can be seen in Figure 15. Using a high pass filter on the already low pass filtered data set reveals by far more and better separable details (see Figure 16) compared to the directly filtered volume.

We have tested the speed our implementation against a well tuned software convolution filter. Unsurprisingly, the software convolution is almost completely memory bound. Even extremely fast workstations as the Octane are limited by the main memory band-

Filter size	2^3	3^3	5^3	7^3
`head`[†]	0.33/0.72	0.33/1.02	0.33/1.56	0.48/2.0
`angio`[‡]	2.5/6.0	2.5/8.7	2.5/14.7	3.7/21.3

† Data set created by computer tomography, 128^3

‡ Data set created by MR angiography, 256^3

All times were measured on a Silicon Graphics Onyx2 equipped with a BasicReality graphics pipe. The system has two R10000/195 MHz processors and 640 MB main memory.

Table 1: Convolution times in seconds using hardware / software

width, as today's caches are far too small for the values needed for convolution along the z-axis. High end machines as the Onyx2 perform huge 3D convolutions three times faster than the Octane, even when equipped with slower CPUs. Standard PCs cannot cope with the memory bandwidth of the Onyx2 system, and multiprocessor options will not accelerate the process because it is not CPU bound.

Table 1 shows convolution times for different data sets and filter sizes, using software and hardware convolution. All times have been measured on an Onyx2 equipped with a BasicReality graphics pipe. The maximum filter size supported by the graphics system is 7^2. Therefor, the maximum 3D convolution that can be performed in hardware on this system is 7^3. Noteworthy is the fact that the BasicReality graphics system is optimized for filter kernels of size 5^2. Convolutions with smaller kernels need exactly the same computation time. Filters of size 6^2 and 7^2 share their timing results as well. The x and y coordinates of the volume are swapped during the hardware based convolution process, which is a side effect of the presented 3D convolution algorithm.

5 Conclusion

As several of today's graphics workstation vendors have added two-dimensional convolution to their OpenGL pipeline, using this capability for accelerating 3D convolution is an almost straightforward approach. We have determined that by using the implemented algorithm three-dimensional convolution can be performed even on big data sets with nearly interactive rates. First promising approaches of accelerating wavelet decomposition and reconstruction have been investigated as well [3].

As all intermediary data is transfered to the frame buffer, clamping can swallow negative values that result from the two-dimensional convolution as well as final negative results. Thus this approach is currently most useful for symmetrical filter kernels. By using post-convolution scaling and bias extensions these problems can be easily overcome.

Non-separable convolutions are not possible right now with this algorithm. However, by applying several two-dimensional filter kernels and blending convoluted images in the frame buffer the use of non-separable 3D kernels will be a possibility for the future as well.

References

[1] M Bro-Nielson. *Medical Image Registration and Surgery Simulation*. PhD thesis, IMM, Technical University of Denmark, 1996.

[2] 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.

[3] M. Hopf and T. Ertl. Hardware Based Wavelet Transformations. In *Erlangen Workshop '99 on Vision, Modeling and Visualization*, Erlangen, November 1999. IEEE. accepted for publication, November 17 – 19.

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

[5] 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.

[6] 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.

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

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

[9] Silicon Graphics Inc., Mountain View, California. *Volume Rendering using RE2 Technology*, 1994.

[10] Silicon Graphics Inc., Mountain View, California. *OpenGL on Silicon Graphics Systems*, 1996.

[11] O. Sommer, A. Dietz, R. Westermann, and T. Ertl. An Interactive Visualization and Navigation Tool for Medical Volume Data. In N. M. Thalmann and V. Skala, editors, *WSCG '98, The Sixth International Conference in Central Europe on Computer Graphics and Visualization '98*, volume II, pages 362–371, Plzen, Czech Republic, February 1998. University of West Bohemia Press.

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

[13] R. Westermann and T. Ertl. A Multiscale Approach to Integrated Volume Segmentation and Rendering. In *Computer Graphics Forum 16(3) (Proc. EUROGRAPHICS '97)*, number 3, pages 117–129. Blackwell, 1997.

[14] R. Westermann and T. Ertl. Efficiently Using Graphics Hardware in Volume Rendering Applications. In *Computer Graphics Proceedings*, Annual Conference Series, pages 169–177, Orlando, Florida, 1998. ACM SIGGRAPH.

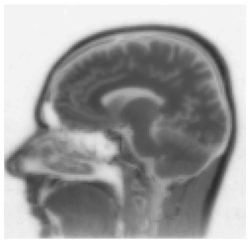
Figure 8: The unfiltered head data set

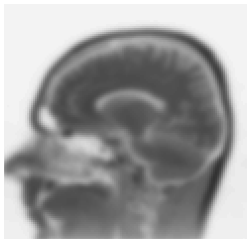
Figure 9: Head, low pass filtered in software

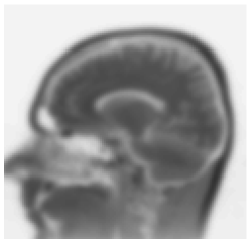
Figure 10: Head, low pass filtered in hardware

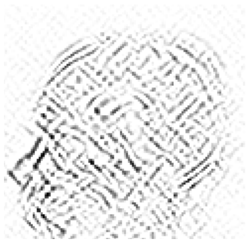
Figure 11: Head, high pass filtered in software

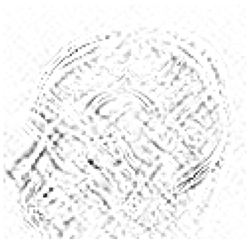
Figure 12: Head, high pass filtered in hardware

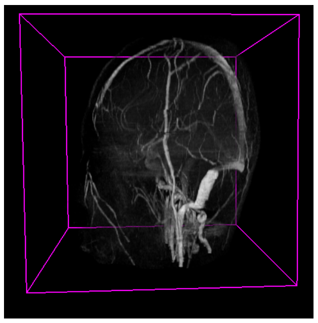
Figure 13: The original angiography data set

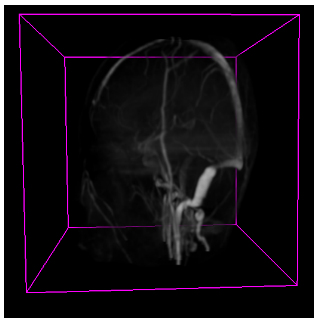
Figure 14: The Gauß filtered data set

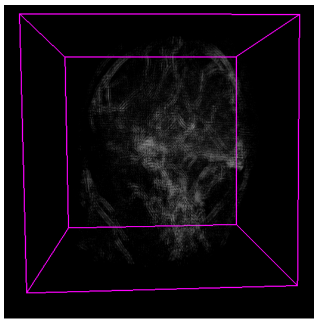
Figure 15: Data, after direct filtering with Gauß' second derivative

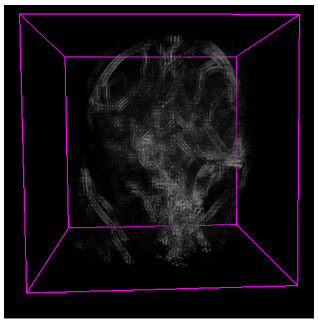
Figure 16: First low pass, then high pass filtered data

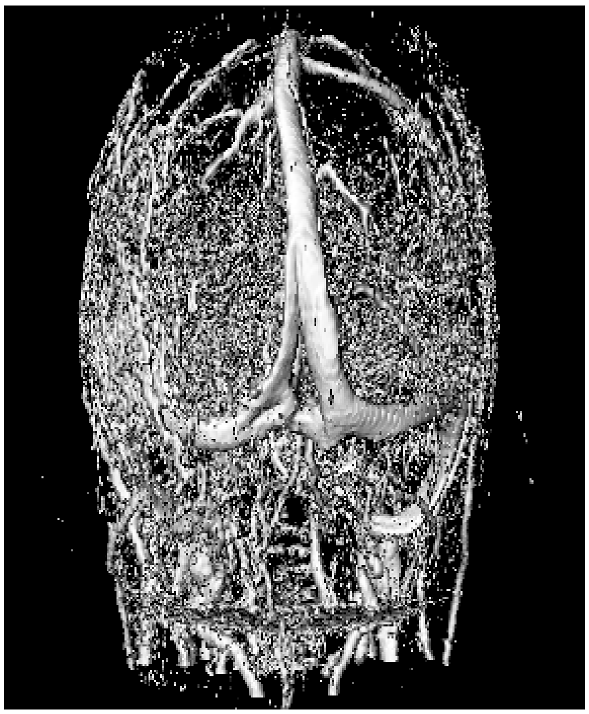
Figure 17: ISO surfaces on the original angiography data set

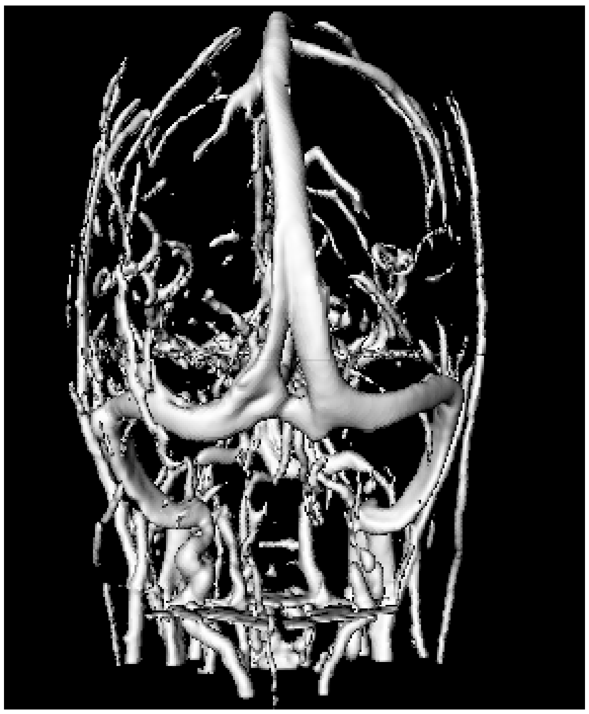
Figure 18: ISO surfaces on the Gauß filtered data set