# Filtering Video Volumes Using the Graphics Hardware

Andreas Langs and Matthias Biedermann

Universität Koblenz-Landau,
Universitätsstrasse 1, 56070 Koblenz, Germany
{alangs,mbmann}@uni-koblenz.de

**Abstract.** Denoising video is an important task, especially for videos captured in dim lighting environments. The filtering of video in a volumetric manner with time as the third dimension can improve the results significantly. In this work a 3D bilateral filter for edge preserving smoothing of video sequences exploiting commodity graphics hardware is presented. A hardware friendly streaming concept has been implemented to allow the processing of video sequences of arbitrary length. The clear advantage of time-based filtering compared to frame-by-frame filtering is presented as well as solutions to current limitations for volume filtering on graphics hardware. In addition, a significant speedup over a CPU based implementation is shown.

**Keywords:** Non-Linear Filtering, Graphics Hardware, Video Processing.

## 1 Introduction

Data acquired by any type of analog-digital converter, such as CCD and CMOS image sensors or sensors in CT/MRI scanners contains noise. In the case of commodity hardware, this can be observed especially in video sequences or images captured in dim lighting conditions. As a result, further processing, like segmentation, is difficult for these datasets.

To enhance the distorted data, filtering is a common first step in the workflow. There are three different types of operations that transfer a pixel of a source dataset or image into the corresponding pixel in the destination dataset: point operations where the destination pixel depends only on the source pixel; local operations where the destination pixel is dependent both on the source pixel and its distinct neighborhood; and finally global operations, where the value of the destination pixel is influenced by all pixels of the source image. These three operations can either be linear or nonlinear, where linear filters describe a convolution of the source image with a given filter kernel. A typical example is the Gaussian filter that resembles a linear local operation with a filter mask consisting of weights that have Gaussian distribution. Another example is the median filter as a nonlinear local operation where the destination pixel's value becomes the median of the source pixel and its neighbourhood. In order to enhance the

B.K. Ersbøll and K.S. Pedersen (Eds.): SCIA 2007, LNCS 4522, pp. 878–887, 2007.

distorted data we would like to smoothen the homogeneous regions while maintaining edges, thus performing an edge preserving smoothing. For reasons to be shown, we use the well-known bilateral filter [1]. Our target datasets are video sequences, i.e., a sequence of individual frames. Thus, we have the possibility of filtering each frame separately with a two-dimensional kernel. Trying to eliminate the noise as much as possible using this bilateral filter usually results in a comic style [2], however, which is caused by the evenly coloured areas with sharp edges in the resulting image. By choosing the parameters of the filter in a way that we get a more natural look, we cannot denoise the data sufficiently.

In our approach we regard the video sequence as a volume and use the bilateral filter in all three dimensions. Areas in the video sequence that are not changing or moving from one frame to the next are then homogeneous regions in time. Thus, we can choose the parameters of the bilateral filter in a way that we limit the influence of each frame and compensate for the lack of denoising with the now available temporal dimension. We are also able to eliminate the noise without introducing the comic look, but preserving a more natural appearence instead.

Representing video sequences as volumes and operations thereon is no new idea, as shown in [3], or [4], but advanced filtering is rarely used due to its high computational costs. To perform filtering on reasonably sized video sequences in an acceptable amount of time, we utilize recent graphic processing units (GPU) found on commodity hardware. Using the GPU for general purpose computations (GPGPU) is currently becoming more and more popular, motivated by the exponential growth of performance, exceeding the already fast progress of CPUs [5]. This way, we are able to perform high quality volumetric filtering of video sequences at PAL resolution in realtime.

The paper is organized as follows: In section 2.1 the underlying system used for data and shader handling and graphics hardware concepts are introduced, followed by implementation details of the GPU based bilateral filter. Subsequently, we explain our approach to filtering video sequences of arbitrary length (section 2.3), including a discussion of current limitations and workarounds. We conclude with quality and performance comparisons in section 3 and finally give an outlook on improvements and filtering in other domains in section 4.

## 2 Approach

### 2.1 Basic Concepts and Setup

Filtering images with custom filter kernels on the GPU has become possible quite recently with the introduction of programmable parts of the 3D rendering pipeline. It basically resembles a stream processor when performing general purpose computation in the graphics hardware. Input and output are streams (textures) on which kernels (shader programs) are performing various, user defineable operations. Important properties of these operations are data independency and data locality, which are both key factors for the high performance of graphics processors. This means that an output stream depends only on the input stream

data, and no additional data besides the stream itself can be transferred between kernels. Imposing these programming restrictions, the GPU is able to compute several kernels in parallel. Although there are some limited exceptions from the mentioned stream processing concept, not all computation problems can be efficiently ported to the GPU without changing the algorithm completely.

In our implementation we used the Plexus framework that has been developed at the University of Koblenz-Landau during a collaborative project. With this graph-based tool we are able to easily model data flow between various nodes, called "devices", representing all kinds of functionality. The type of data flowing can be of an arbitrary nature, although the devices have to be capable of handling the appropriate data, of course. Plexus provides some basic functionality to transfer data between CPU and GPU, along with devices that are able to provide a stream of images from a video file, webcam, etc. By using this framework we have been able to adapt our approach quickly to various data sources and use it in different configurations.

The basic data type for representing 2D image data on the CPU in Plexus is "Image". These "Images" can be uploaded to the GPU where they become a "Texture2D" (for reading) or a "GPUStream" (for reading or writing), which are roughly equivalent. In addition, for the GPU there are data types like "Texture3D" and "FlatVolume", which will be described in more detail in section 2.3. All data types that have been mentioned before can have different numbers of channels and bit-depths per channel. For filtering video sequences, we used three channels (RGB) with eight bits each, which is sufficient for common video material and is also natively supported by a wide range of GPUs.

Concerning the architecture of GPUs, we only use one of currently three user programmable parts of the graphics pipeline: the fragment processing step. The vertex and geometry shader (the latter introduced with Shader Model 4.0 [6]) are not used in our approach, as is obvious for pure pixel data such as video frames. Therefore, we simply use the graphics API to draw a quadrilateral in size of the input data (i.e., video frame resolution) to initiate data processing. In our implementation the source video frame is the input data to be processed by the initial shader, i.e., the first render pass. The one-dimensional bilateral filter is then applied to this texture resulting in one filtered pixel value, which is written to the framebuffer (or some equivalent offscreen target). In our application, this result is used in subsequent processing steps to realize three-dimensional filtering.

### 2.2 Bilateral Filtering on the GPU

As image denoising is a common research topic, various filter types have been proposed. Among these approaches anisotropic diffusion and wavelet based filters are well studied and widely used, mainly due to their edge preserving properties. In the context of GPGPU, however, anisotropic diffusion as an iterative solution is not well suited for the GPU. Wavelet based filters on the other hand require several conversions to the wavelet domain and thus introduce a considerable overhead.

Barash has shown in [7] that bilateral filtering resembles anisotropic diffusion by using adaptive smoothing as link between the two approaches. Therefore, we are able to implement bilateral filtering as a more GPU-friendly algorithm because of its "local operation" nature, while maintaining their common basis. That is, it is possible to implement the bilateral filter, without significant changes, as a shader program using the OpenGL Shading Language.

The bilateral filter itself was first introduced in 1998 by C. Tomasi und R. Manduchi [1]. Its basic idea is to combine the domain and the range of an image. The domain describes the spatial location or closeness of two pixels, while the range describes their similarity, or the distance of the pixel values. In traditional filters only a pixel's location in the spatial domain is taken into account, resulting in less influence for more distant pixels, for example. The bilateral filter additionally takes the similarity into account.

The bilateral filter is defined as:

$$k\left(x\right) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} c\left(\xi, x\right) s\left(f\left(\xi\right), f\left(x\right)\right) d\xi$$

with the spatial closeness $c(\xi, x)$ between the neighborhood center $x$ and a nearby point $\xi$, and the similarity $s(f(\xi), f(x))$ between $x$ and $\xi$. The pixel value at $x$ is therefore a combination of the domain $c$ and the range $s$.

A simple and important case of the distance function in the domain is the gaussian weighted euclidean distance:

$$c\left(\xi, x\right) = e^{-\frac{1}{2}\left(\frac{d(\xi, x)}{\sigma_d}\right)^2}$$

with the euclidean distance function:

$$d\left(\xi, x\right) = d\left(\xi - x\right) = \left\|\xi - x\right\|$$

The distance function in the range is defined analog to $c$:

$$s\left(\xi, x\right) = e^{-\frac{1}{2}\left(\frac{\delta(f(\xi), f(x))}{\sigma_r}\right)^2}$$

with an appropriate distance measure for the intensity values $\phi$ and $f$:

$$\delta\left(\phi, f\right) = \delta\left(\phi - f\right) = \left\|\phi - f\right\|$$

In the simplest case, this can be the absolute pixel value. With parameters $\sigma_d$ and $\sigma_r$, the filter result can be influenced. The bilateral filter is inherently non-linear because of its utilization of the image's range, and is therefore not directly separable.

In our implementation, however, we use the separated bilateral filter introduced by Pham et al. [8]. As described in their work, it is not a true separation of the original bilateral filter, thus leading to approximated rather than equal results. However, for the purpose of filtering video sequences the introduced error is negligible, especially considering the potential increase in performance. In our

approach we convolve the source frame with the separated, two-dimensional bilateral filter, thus applying the one-dimensional kernel twice. The internal video volume is then built from these filtered slices and finally reused several times for the convolution in the third dimension. The whole procedure is described in more detail in the following section.

### 2.3 Streaming Approach

Filtering video data in a volumetric manner cannot be done directly for sequences of reasonable lengths as the entire data usually does not fit into the graphics memory. Therefore, we have implemented a streaming approach to allow for videos of arbitrary length. When choosing the parameters for the kernel so that $n$ previous and $n$ subsequent frames are used to filter the current frame, we have to keep $2n+1$ frames in the graphics cards memory at once. In every filtering step we upload the next video frame to the GPU, which then is processed separately in $x$ and $y$ direction using the separated bilateral filter. This two-dimensionally filtered slice is subsequently added to the video volume, thus replacing the oldest slice in the volume. The last step is to generate the processed video frame by filtering the video volume in $z$ direction. The final video frame is then displayed and/or downloaded into the CPUs memory, to be written to a file or processed further. The whole process is depicted in the figure below:

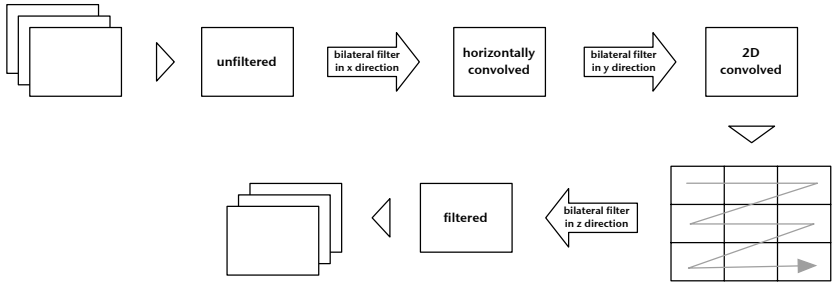


**Fig. 1.** Processing steps of our separable bilateral filtering approach

To minimize data transfers during copying to and from the GPU, and inside the GPU's memory, we use a ring buffer storage scheme for the video volume. Thus, it is possible to constantly add slices to the volume until the maximum number of slices is reached.

However, using this approach we encountered a severe limitation especially on NVIDIA graphics cards: the maximum size of 3D textures being $512^3$ pixels, whereas only their most recent hardware (available since November 2006) is capable of larger 3D textures ($2048^3$). With this constraint we were not able to filter videos with the desired PAL resolution of $720 \times 576$ pixels using traditional 3D textures in our original implementation. To circumvent this limitation we used a 3D data representation known as "flat volume", introduced by Harris et al. [9].

The idea behind this technique is to place every slice of the volume side by side, resulting in a large 2D texture. This concept is also shown in figure 1, and exploits the graphics hardware's support for 2D texture sizes up to $4096^2$ (latest hardware: $8192^2$) pixels. We can then use a simple address translation to convert a pixel position in 3D $(x, y, z)$ to the according pixel position in the 2D flat volume $(x, y)$. As an additional benefit of using the "flat volume" we also measured a significant performance increase as described in section 3.2.

## 3 Results

### 3.1 Comparison 3D/2D Filtering

To assess the quality of our filtering – especially the difference between the 2D and 3D version – we did a *full reference* comparison. Therefore, we took a video sequence without noise (our reference sequence) and added synthetic noise to it by adding an evenly distributed random value between -50 and 50 to the R, G, and B channel of every pixel independently. This is motivated by the nature of noise of commodity sensors, especially when used in dim lighting conditions. We then compared the video sequences with two different measures by performing a frame-by-frame comparison and averaged the measured error values for the entire sequence. The first video sequence that has been compared is an outdoor video sequence captured in daylight without noticeable noise. The second video is a synthetic animation showing a typical pre-video countdown, combined with a testbar screen for broadcasting, thus being completely noise free.

The two measures we used are the SSIM (Structural Similarity Index Measure) introduced by Wang et al. [10] and the MSE (Mean squared error). The SSIM is a measure specially designed to compare two images in a perception based manner, whereas the MSE measure is purely mathematically based and straight forward, but is by its nature not capable of measuring the quality difference from an observer's point of view.

After computing an error value for the distorted video sequence in comparison to the original sequence, we are able to compare the distorted video sequence filtered with the 2D and 3D bilateral filter, respectively, with the reference sequence. The results can be seen in table 1. By the method's definition, a value of 1 means completely equal frames with the SSIM: the lower the value, the more

**Table 1.** Quality comparison

| | outdoor | | synthetic | |
|---|---|---|---|---|
| | SSIM | MSE | SSIM | MSE |
| unfiltered | 0,393 | 669,6 | 0,311 | 626,3 |
| filtered with "2D bilateral filter" (GPU-BF 2D) | 0,737 | 135,0 | 0,720 | 125,7 |
| filtered with "3D bilateral filter" (GPU-BF 3D/flat3D) | 0,844 | 87,3 | 0,890 | 73,9 |
| filtered with AfterEffects "Remove grain" | 0,829 | 89,7 | 0,908 | 68,3 |
| filtered with "Neat Video" | 0,909 | 53,9 | 0,954 | 48,1 |

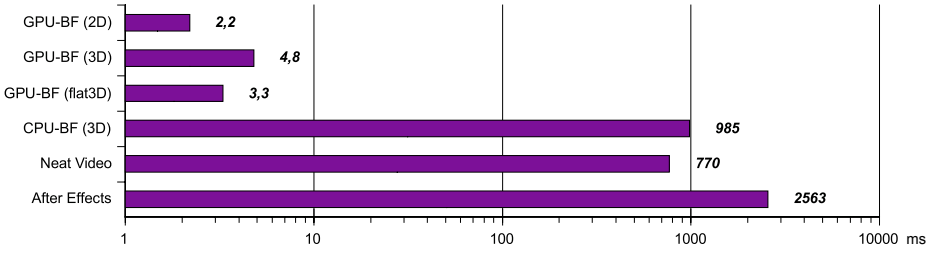


**Fig. 2.** Performance comparison (time per frame)

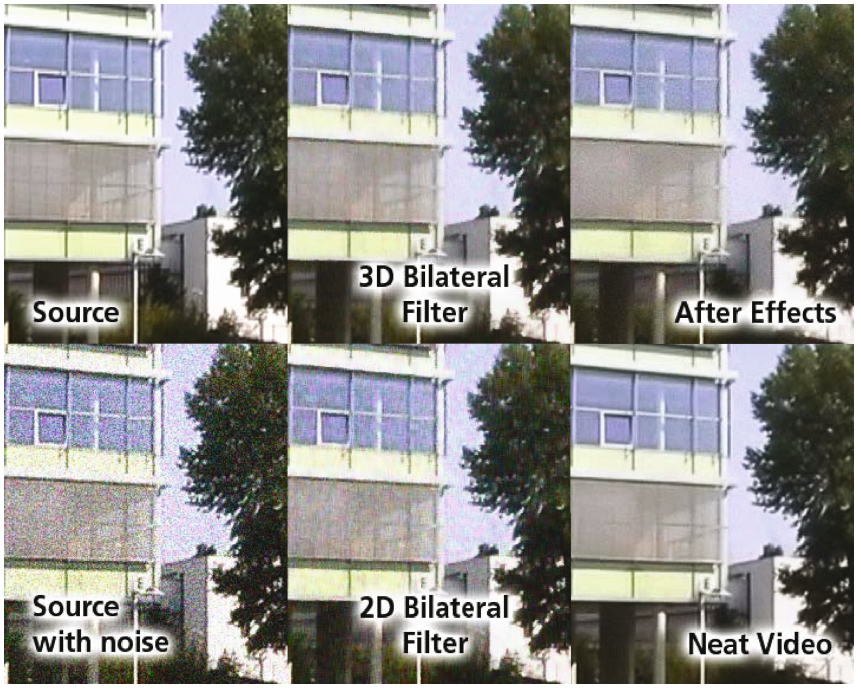


**Fig. 3.** Example "outdoor" for quality comparison

different the compared images are. In contrast, value 0 corresponds to no differences with the MSE. We also included the results of two commercial products: "Adobe AfterEffects 7.0" (i.e., "Remove grain" filter) and the relatively new "Neat Video Pro 1.5", which are both widely used and designed for this field of application. The filter methods used by these two programs are not disclosed. A visual comparison of the results is shown in figure 3.

As can be seen there is no significant difference of the filter quality between the two very different types of video sequences: both error metrics indicate the same relationship. The quality of 3D filtering surpasses the quality of the 2D

filter, but is comparable to the quality of the "Adobe AfterEffects" filter. In contrast, the visual result of "Neat Video" is better than ours.

We also measured the time needed for filtering a single frame. Our application runs on a commodity Windows PC that features following components: Intel CoreDuo 6400 (2.13 GHz), 2 GB RAM, NVIDIA GeForce 8800 GTX 768 MB PCIe 16x. The video sequence "outdoor", used for comparison, has a resolution of $720 \times 576$ and is 1085 frames long. The denoted time per frame is an average over all frames of the filtering only, that is excluding the time needed for reading or writing the video file or uploading the data to the GPU. The results are given in figure 2. They obviously show that using the GPU for filtering yields a significant performance improvement, approximately three orders of magnitude faster than comparable functionality in commercial products.

### 3.2 Performance 3D/Flat Volume

The processing time for our implementation using the flat volume storage concept has a performance gain of approximately 25% on current hardware with respect to full 3D volume textures, as can be seen from figure 2. On previous graphics hardware the difference using the flat volume was even higher, up to a factor of two. This indicates some trend for improved and complete 3D texture processing in graphics hardware, but due to the very new hardware a detailed evaluation could not be taken into account in this work yet.

By using the flat volume textures, we were also able to filter PAL resolution video sequences even on older graphics hardware (esp. NVIDIA) in their original resolution. As the architecture of commodity graphics cards is mainly driven by the computer games industry, features like true 3D texture functionality have not yet been optimized to the same level as their 2D counterparts. This can also be experienced with API functions for 3D texture processing like OpenGL function calls which have not been hardware accelerated or are not available at all, until recently, as described before. Therefore, if no advanced texture access (e.g., trilinear filtering) is needed, flat volume textures are an appropriate alternative to native 3D textures. Despite the additional overhead for the address translation, they offer a considerable performance increase, especially with older graphics hardware.

## 4 Conclusion and Future Work

In this paper we have described that using a 3D bilateral filter for noise reduction on video sequences results in higher quality than with frame-by-frame 2D bilateral filtering. In figure 4 we give another real world example with comparable performance results, which depicts a video frame acquired in a very dark room, lit only indirectly by a projection screen. In addition, we have shown that using modern commodity graphics hardware clearly decreases processing time for this rather costly volumetric filter operation. In order to exploit their capabilities even better, we argued that using flat volumes for three-dimensional data

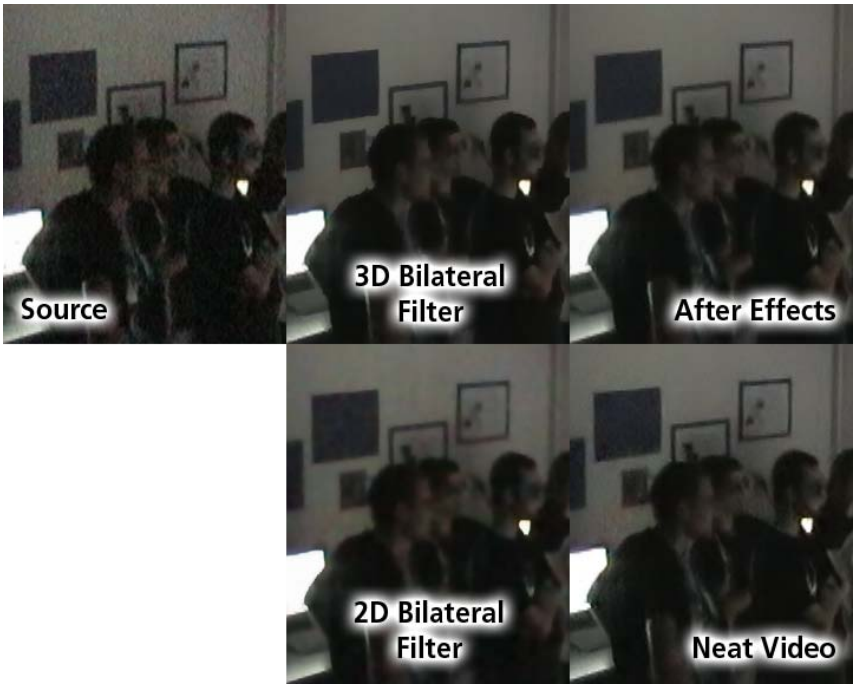


**Fig. 4.** Real world example in dim lighting conditions with natural noise

offers advantages in comparison to true 3D textures for a wide range of graphics hardware.

Based on this application, it is also possible to perform fast filtering of real 3D datasets, like medical volumes to enhance the results of subsequent processing steps like segmentation. However, the size of the volume is then directly limited by the available memory on the graphics card, so that other approaches like bricking have to be employed.

One natural disadvantage of smoothing is the loss of high frequencies which can be seen in some regions of the example material. To compensate for this, we would like to investigate other types of smoothing filters or sequences of filter operations. Therefore, we are planning to further generalize the concepts to other kinds of computations amenable to the stream processing nature of current graphics hardware.

The dramatic decrease in filtering time by using the GPU has been shown with video data of 8 bit per channel. It would be interesting to see how well our approach works on data with 16 bit per channel or even other representations, e.g., true high dynamic range data. As current graphics hardware is heavily optimized for floating point data, an equally, if not greater performance increase should be possible.

## Acknowledgements

We would like to thank Tobias Ritschel, who has provided the Plexus framework developed at the University of Koblenz-Landau as a basis for our work, as well as Oliver Abert and Thorsten Grosch for helpful discussions and suggestions. Also, we would like to thank the reviewers for their helpful and motivating comments.

## References

1. Tomasi, C., Manduchi, R.: Bilateral Filtering for Gray and Color Images. ICCV: 839-846 (1998)
2. Fischer, J., Bartz, D., Straßer, W.: Stylized Augmented Reality for Improved Immersion. In: Proceedings of IEEE Virtual Reality (VR 2005), pp. 195–202 (2005)
3. Hájek, J.: Timespace Reconstruction of Videosequences, 6th Central European Seminar on Computer Graphics (CESCG) (2002)
4. Daniel, G., Chen, M.: Visualising Video Sequences Using Direct Volume Rendering, Vision, Video, and Graphics, VVG 2003, University of Bath, UK, July 10-11th, 2003. In: Proceedings (2003)
5. Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A.E., Purcell, T.J.: A Survey of General-Purpose Computation on Graphics Hardware. Computer Graphics Forum, vol. 26, (to appear) (2007)
6. Blythe, D.: The Direct3D 10 system. ACM Trans. Graph. 25(3), 724–734 (2006)
7. Barash, D.: Bilateral Filtering and Anisotropic Diffusion: Towards a Unified Viewpoint, Scale-Space '01. In: Proceedings of the Third International Conference on Scale-Space and Morphology in Computer Vision, pp. 273–280. Springer, Heidelberg (2001)
8. Pham, T.Q., van Vliet, L.J.: Separable bilateral filtering for fast video preprocessing. In: ICME 2005: IEEE International Conference on Multimedia & Expo, IEEE Computer Society Press, Los Alamitos (2005)
9. Harris, M.J., Baxter, W.V., Scheuermann, T., Lastra, A.: Simulation of cloud dynamics on graphics hardware. HWWS '03: In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware Eurographics Association
10. Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image Quality Assessment: From Error Visibility to Structural Similarity. IEEE Trans. Image Processing, vol. 13 (2004)
11. Viola, I., Kanitsar, A., Gröller, M.E.: Hardware-Based Nonlinear Filtering and Segmentation using High-Level Shading Languages. In: Proceedings of IEEE Visualization, IEEE, New York (2003)