Hardware-Accelerated High-Quality Filtering on PC Hardware

Markus Hadwiger* Thomas Theußl† Helwig Hauser* Eduard Gröller†

* VRVis Research Center
Donau-City-Strasse 1, 1220 Vienna, Austria

† Institute of Computer Graphics and Algorithms
Vienna University of Technology
Karlsplatz 13/186, 1030 Vienna, Austria

Abstract

We describe a method for exploiting commodity 3D graphics hardware in order to achieve hardware-accelerated high-quality filtering with arbitrary filter kernels. Our approach is based on reordering the evaluation of the filter convolution sum to accommodate the way the hardware works. We exploit multiple rendering passes together with the capability of current graphics hardware to index into several textures at the same time (multi-texturing). The method we present is applicable in one, two, and three dimensions. The cases we have been most interested in up to now are two-dimensional reconstruction of object-aligned slices through volumetric data, and three-dimensional reconstruction of arbitrarily oriented slices. As a fundamental building block, the basic algorithm can be used in order to directly render an entire volume by blending a stack of slices reconstructed with high quality on top of each other. However, it is important to emphasize that our approach has no fundamental restrictions with regard to the filters that can be employed. Thus, it could also be used for more general filtering tasks than reconstruction, e.g., image processing.

1 Introduction

A fundamental problem in computer graphics is how to reconstruct images and volumes from sampled data. The process of determining the original continuous data – or at least a sufficiently accurate approximation – from discrete input data is usually called function or signal reconstruction. In volume visualization, the input data is commonly given at evenly spaced discrete locations in three-space. In theory, the original volumetric data can be reconstructed entirely, provided certain conditions are honored (cf. sampling theorem [13]). In reality, of course, reconstruction is always a trade-off between performance and quality. This is especially true for hardware implementations. Reconstruction in graphics hardware is usually done by using simple linear interpolation. This is fast, but introduces significant reconstruction artifacts. On the other hand, a lot of research in the last few years has been devoted to improving reconstruction by using high-order reconstruction filters [6, 10, 11, 12, 15]. Among the investigated filters are piecewise cubic functions, as well as windowed ideal reconstruction functions (windowed sinc filters). However, these filters were usually deemed to be too slow to be used in practice.

In this paper, we will show how to exploit consumer 3D graphics hardware for accelerating high-order reconstruction of volumetric data. The presented approach works in one, two, and three dimensions, respectively. Up to now, we have used our method for reconstruction of images and slices in two dimensions, and reconstruction of oblique slices through volumetric data. An interesting application of such slices is to use them for direct volume rendering. Standard texture mapping hardware can be exploited for volume rendering by blending a stack of texture-mapped slices on top of each other [1]. These slices can be either viewport-aligned, which requires 3D texture mapping hardware [7, 17], or object-aligned, where 2D texture mapping hardware suffices [14]. Our high-quality filtering approach can be used to considerably improve reconstruction quality of the individual slices in both of these cases, thus increasing the quality of the entire rendered volume.

As reconstruction kernels we have used bicubic and tricubic B-splines and Catmull-Rom splines,

* {Hadwiger, Hauser}@VRVis.at, http://www.VRVis.at/vis/
† {theussl, groeller}@cg.tuwien.ac.at, www.cg.tuwien.ac.at

as well as windowed sinc filters using Kaiser and Blackman windows.

The structure of the paper is as follows. After discussing related work in section 2, we describe our method for hardware-accelerated high-order filtering in section 3. Section 4 presents results, and section 5 summarizes what we have presented, tries to draw some general conclusions, and wraps up with future work.

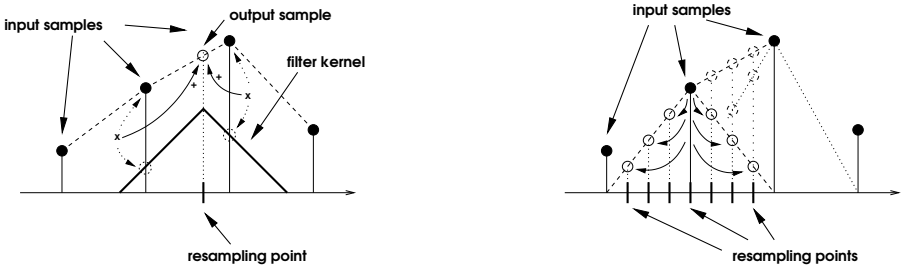
2 Related work

There is a vast amount of literature on filter analysis and design. Keys [5] derived a family of cardinal splines for function reconstruction, and showed, using a Taylor series expansion, that among these the Catmull-Rom spline is numerically most accurate. Mitchell and Netravali [9] derived another family of cubic splines quite popular in computer graphics, the BC-splines. Marschner and Lobb [6] compared linear interpolation, cubic splines, and windowed sincs. They concluded that linear interpolation is certainly the cheapest option and will likely remain the method of choice for time critical applications. Cubic splines perform quite well, especially those BC-splines that fulfill $2C + B = 1$ which include the Catmull-Rom spline. Windowed sincs can provide arbitrarily good reconstruction, while being much more expensive. Möller et al. [10] provide a general framework for analyzing filters in spatial domain again using a Taylor series expansion of the convolution sum. They use this framework to analyze the cardinal splines [10], affirming that the Catmull-Rom splines are numerically most accurate, and the BC-splines [11], affirming that the filters with parameters fulfilling $2C + B = 1$ are numerically most accurate. Since numerical considerations alone may not always be appropriate, they also show how to use their framework to design accurate and smooth reconstruction filters [12]. Turkowsky [16] used windowed ideal reconstruction filters for image resampling tasks. Theußl et al. [15] used the framework developed by Möller et al. [10] to assess the quality of windowed reconstruction filters and to derive optimal values for the parameters of Kaiser and Gaussian windows.

The idea of using a reordered evaluation of the filter convolution sum that is employed by our approach is also used in all splatting-based techniques [18]. However, apart from this basic similarity our method performs reconstruction in a way that is significantly different from splatting, e.g., we are not rendering kernel footprints.

Meißner et al. [8] have done a thorough comparison of the four prevalent approaches to volume rendering, one of them being the use of texture mapping hardware. Hardware-accelerated texture mapping has been used to accelerate volume rendering for quite some time. 3D textures can be used effectively for rendering volumes at interactive frame rates, as shown by Cabral et al. [1], and Cullip and Neumann [2], for example. These early approaches all stored a preshaded volume in a 3D texture, which had to be recomputed whenever the viewing parameters, lighting conditions, or transfer function were changed. Westermann and Ertl [17] introduced several approaches how to accelerate volume rendering with graphics hardware that is capable of three-dimensional texture mapping and supports a color matrix. They were able to render monochrome images of shaded isosurfaces at interactive rates – without explicitly extracting any geometry – and performed shading on-the-fly, without requiring the volume texture to be updated. Meißner et al. [7] have extended this approach for semi-transparent volume rendering, and are able to apply both classification and colored shading at run-time without changing the volume texture itself. Dachille et al. [3] used a mixed-mode approach between applying a transfer function and shading in software, and rendering the resulting texture volume in hardware. Many of the approaches presented earlier have been adapted to standard PC graphics hardware later on, e.g., by Rezk-Salama et al. [14]. They are only using two-dimensional textures instead of three-dimensional ones, exploiting the multi-texturing and multi-stage rasterization capabilites of NVIDIA GeForce graphics cards. The high-quality filtering approach we are proposing can be used in conjunction with many of the methods previously described.

The OpenGL imaging subset (introduced in OpenGL 1.2) offers convolution capabilities, but these are primarily intended for image processing and unfortunately cannot be used for the kind of reconstruction we desire. That is, these features cannot be used for reconstructing a signal at arbitrary locations. Hopf and Ertl [4] have shown how to achieve 3D convolution by building upon two-dimensional convolution in OpenGL.



(a) (b)

Figure 1: Gathering vs. distribution of input sample contributions (tent filter): (a) Gathering all contributions to a single output sample (b) Distributing a single input sample's contribution.

3 Hardware-Accelerated High-Order Filtering

This section presents our approach for filtering input data by convolving it with an arbitrary filter kernel stored in multiple texture maps, exploiting low-cost 3D graphics hardware. The basic principle is most easily illustrated in the case of one-dimensional input data (sections 3.1 and 3.2), and can also be applied in two (section 3.3), as well as three dimensions (sections 3.4 and 3.5).

3.1 Basic principle

Since we want to be able to employ arbitrary filter kernels for reconstruction, we have to evaluate the well-known filter convolution sum:

$$g(x) = (f * h)(x) = \sum_{i=\lfloor x \rfloor - m + 1}^{\lfloor x \rfloor + m} f[i]h(x - i) \quad (1)$$

This equation describes a convolution of the discrete input samples $f[i]$ with a continuous reconstruction filter $h(x)$. If the function to be reconstructed was band-limited and sampled properly, using the sinc function as reconstruction filter would result in perfect reconstruction. However, the sinc function is impracticable because of its infinite extent. Therefore, in practice finite approximations to the sinc filter are used, which yield results of varying quality. In equation 1, the (finite) half-width of the filter kernel is denoted by m.

In order to be able to exploit standard graphics hardware for performing this computation, we do not use the evaluation order usually employed, i.e., in software-based filtering. The convolution sum is commonly evaluated in its entirety for a single output sample at a time. That is, all the contributions of neighboring input samples (their values multiplied by the corresponding filter values) are *gathered* and added up in order to calculate the final value of a certain output sample. This "gathering" of contributions is illustrated in figure 1(a). This figure uses a simple tent filter as an example. It shows how a single output sample is calculated by adding up two contributions. The first contribution is gathered from the neighboring input sample on the left-hand side, and the second one is gathered from the input sample on the right-hand side. In the case of this example, the convolution results in linear interpolation, due to the tent filter employed. For generating the desired output data in its entirety, this is done for all corresponding resampling points (output sample locations).

Our method uses a different evaluation order. Instead of focusing on a single output sample at any one time, we calculate the contribution of a single input sample to all corresponding output sample locations (resampling points) first. That is, we *distribute* the contribution of an input sample to its neighboring output samples, instead of the other way around. This "distribution" of contributions is shown in figure 1(b). In this case, the final value of a single output sample is only available when all corresponding contributions of input samples have been distributed to it.

We evaluate the convolution sum in the order outlined above, since the distribution of the contributions of a single relative input sample can be done in hardware for all output samples (pixels) simulta-

neously. The final result is gradually built up over multiple rendering passes. In the example of a one-dimensional tent filter (like in figure 1), there are two relative input sample locations. One could be called the "left-hand neighbor," the other the "right-hand neighbor." In the first pass, the contribution of all respective left-hand neighbors is calculated. The second pass then adds the contribution of all right-hand neighbors. Note that the number of passes depends on the filter kernel used, see also below.

Thus, the same part of the filter convolution sum is added to the previous result for each pixel at the same time, yielding the final result after all parts have been added up. From this point of view, the graph in figure 1(b) depicts both rendering passes that are necessary for reconstruction with a one-dimensional tent filter, but only with respect to the contribution of a single input sample. The contributions distributed simultaneously in a single pass are depicted in figures 2 and 3, respectively. In the first pass, shown in figure 2, the contributions of all relative left-hand neighbors are distributed. Consequently, the second pass, shown in figure 3, distributes the contributions of all relative right-hand neighbors. Adding up the distributed contributions of these two passes yields the final result for *all* resampling points (i.e., linearly interpolated output values in this example).

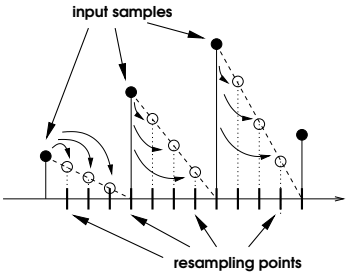


Figure 2: Distributing the contributions of all "left-hand" neighbors; tent filter.

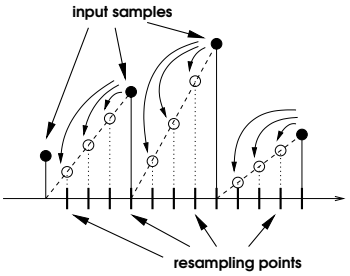


Figure 3: Distributing the contributions of all "right-hand" neighbors; tent filter.

3.2 Accommodating graphics hardware

The rationale for the approach described in the previous section is that, at the pixel or fragment level, graphics hardware basically operates by applying simple, identical operations to a lot of pixels simultaneously—or at least in very rapid succession, which for all conceptual purposes can be viewed as fully parallel operation from the outside. Naturally, these operations have access to a very limited number of inputs. However, in general filtering based on a convolution of an input signal with a filter kernel, a potentially unbounded number of inputs needs to be available for the calculation of a single output value. Therefore, in order to exploit graphics hardware and accommodate the way it works, our method reorders the evaluation order of the filter convolution sum.

Section 3.1 has already shown how we do this reordering, and we are now going to describe in detail how this basic principle can be employed in order to achieve high-order reconstruction in hardware. The following facts are crucial to the operation of our method, and together form the reason why it works. If we look at filtering and the convolution of an input signal with a filter kernel at a single output sample location, we can observe that – even for a kernel of arbitrary width – each segment from one integer location of the kernel to the next is only relevant for exactly one input sample. In the case of a tent filter (which has width two) this can be seen in figure 1(a), by imagining the filter kernel sliding from the second input sample shown to the third input sample.

From now on we will be calling a filter segment of width one from one integer location in a filter kernel to the next a *filter tile*. Consequently, a one-dimensional tent filter has two filter tiles, corresponding to the fact that it has width two. Looking again at figure 1(a), we can now say that each filter tile covers exactly one input sample, regardless of where the kernel is actually positioned. If we now imagine all output sample locations (resampling points) between two given input samples simultaneously, instead of concentrating on a single

output sample, we can further observe that:

- this area has width one, which is exactly the width of a single filter tile,
- all output samples in this area get a non-zero contribution from each filter tile exactly once,
- as many input samples yield a non-zero contribution as there are filter tiles in the filter kernel, and
- this number is, as defined above, the width of the filter kernel.

Now, instead of imagining the filter kernel being centered at the "current" output sample location, we note that an identical mapping of input samples to filter values can be achieved by replicating a single filter tile mirrored in all dimensions (in the case of more than one dimension) repeatedly over the output sample grid, see figure 4. The scale of this mapping is chosen so that the size of a single tile corresponds to the width from one input sample to the next. Note that the need for mirroring filter tiles has nothing to do with the fact that a filter kernel is usually mirrored in the filter convolution sum (see equation 1). Mirroring is necessary to compensate for the fact that, instead of "sliding" the filter kernel, individual filter tiles are positioned at (and replicated to) fixed locations. This becomes clear when looking at one input sample location after the other from left to right, together with the values in a non-moving filter tile located above. For correct correspondences the tile has to be mirrored. Figure 4 shows how this works in the case of a one-dimensional cubic Catmull-Rom spline, which has width four and thus consists of four filter tiles.

We calculate the contribution of a single specific filter tile to all output samples in a single pass. The input samples used in a single pass correspond to a specific relative input sample location or offset with regard to the output sample locations. That is, in one pass the input samples with relative offset zero are used for *all* output samples, then the samples with offset one in the next pass, and so on. The number of passes necessary is equal to the number of filter tiles the filter kernel used consists of. Note that the subdivision of the filter kernel into its tiles is crucial to our method and necessary in order to attain a correct mapping between locations in the input data and the filter kernel, and to achieve a consistent evaluation order of passes everywhere.

We employ multi-texturing with (at least) two textures and retrieve input samples from the first

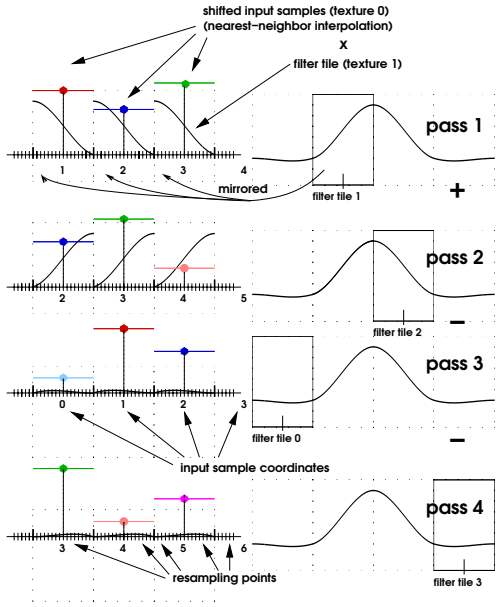


Figure 4: Catmull-Rom spline of width four used for reconstruction of a one-dimensional function in four passes

texture, and filter kernel values from the second texture. Actually, due to the fact that only a single filter tile is needed during a single rendering pass, all tiles are stored and downloaded to the graphics hardware as separate textures. The required replication of tiles over the output sample grid is easily achieved by configuring the hardware to automatically extend the texture domain beyond $[0, 1] \times [0, 1]$ by simply repeating the texture. In order to fetch input samples in unmodified form, nearest-neighbor interpolation has to be used for the input texture. The textures containing the filter tiles are sampled using the hardware-native linear interpolation. If a given hardware architecture is able to support $2n$ textures at the same time, the number of passes can be reduced by n. That is, with two-texture multi-texturing four passes are needed for filtering with a cubic kernel in one dimension, whereas with four-texture multi-texturing only two passes are needed, etc. Our approach is not limited to symmetric filter kernels, although symmetry can be exploited in order to save texture memory for the filter tile textures. It is also not limited to separable filter kernels (in two and three dimensions, respectively). How-

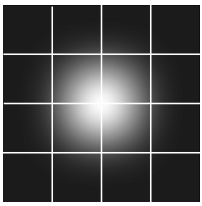


Figure 5: Bicubic B-spline filter kernel; filter tiles separated by white lines.

ever, some filter kernels, e.g., non-separable ones, may need additional passes if they contain both positive and negative areas. In this case, every tile containing both positive and negative values needs to be split in two, correspondingly requiring two passes instead of one.

Additionally, the algorithm is identical for orthogonal and perspective projections of the resulting images. Basically, we are reconstructing at single locations in object space, which can be viewed as happening before projection. Thus, we are independent from the projection used. Note that we are not considering area-averaging filters, since we are assuming that magnification is desired instead of minification. This is in the vein of graphics hardware using bilinear interpolation for magnification, and other approaches, usually mip-mapping, to deal with minification.

3.3 Reconstruction of object-aligned slices

The basic algorithm outlined in the previous section for one dimension can easily be applied in two dimensions. For each output pixel and pass, our method takes two inputs. Unmodified (i.e., unfiltered) image values, and filter kernel values. That is, two 2D textures are used simultaneously. One texture contains the entire source slice, and the other texture contains the filter tile needed in the current pass. Figure 5 shows an example of a two-dimensional filter kernel texture. All sixteen tiles are shown, where in reality only three tiles would actually be downloaded to the hardware, exploiting symmetry. In addition to using the appropriate filter tile, in each pass an appropriate offset has to be applied to the texture coordinates of the texture containing the input slice. As explained in the previous section, each pass corresponds to a specific relative location of an input sample. Thus, the slice texture coordinates have to be offset and scaled in order to match the point-sampled input image grid with the grid of replicated filter tiles.

3.4 Reconstruction of oblique slices

When planar slices through 3D volumetric data are allowed to be located and oriented arbitrarily, three-dimensional filtering has to be performed although the result is still two-dimensional. On graphics hardware, this is usually done by trilinearly interpolating within a 3D texture. Our method can also be applied in this case in order to improve reconstruction quality considerably. The conceptually straightforward extension of the 2D approach described in the previous section, simultaneously using two 2D textures, achieves the equivalent for three-dimensional reconstruction by simultaneously using two 3D textures. The first 3D texture contains the input volume in its entirety, whereas the second 3D texture contains the current filter tile. In the case of a cubic filter kernel for tricubic filtering, 64 passes need to be performed on two-texture multi-texturing hardware. If such a kernel is symmetric, we download four 3D textures for the filter tiles, reusing them for the remaining 60. Due to the high memory consumption of 3D textures, it is especially important that the filter kernel need not be downloaded to the graphics hardware in its entirety if it is symmetric.

In addition to the application outlined in this section, our method applied to three-dimensional reconstruction can also be used for high-quality filtering of solid textures.

3.5 Volume rendering

Our technique can also be used for direct volume rendering by using either one of the two major approaches exploiting texture mapping hardware, i.e., blending either viewport-aligned [17] or object-aligned [14] slices on top of each other. Furthermore, our approach can also be used to reconstruct gradients with high quality, in addition to reconstructing density values. This is possible in combination with hardware-accelerated methods that store gradients in the RGB components of a texture [7, 17]. Note that it is usually necessary to read back slices if multiple rendering passes are needed for a single slice, in order to prevent reconstruction of slices from interfering with each other.

	Bicubic B-spline	Bicubic Catmull-Rom	Tricubic B-spline
kernel width	4	4	4
slice orientation	object-aligned	object-aligned	oblique
number of passes	16	16	64
slice/volume resolution	256x256	256x256	128x128x128
render resolution	500x500	500x500	500x500
NVIDIA GeForce 2 [fps]	12.6	12.6	-
NVIDIA GeForce 3 (hq) [fps]	9.4	9.4	4.5
NVIDIA GeForce 3 (fast) [fps]	-	-	7.2
ATI Radeon [fps]	10.1	-	1.2

Table 1: Frame rates for different scenarios.

4 Results

In our work, we are focusing on widely available low-cost PC graphics hardware. Currently, the two premier graphics accelerators in this field are the NVIDIA GeForce 3 and the ATI Radeon. The graphics API we are using is OpenGL. The GeForce 3 supports multi-texturing with four 2D or 3D textures at the same time and offers the capability to subtract from the contents of the frame buffer. The Radeon supports up to three simultaneous 2D textures, as well as one 3D plus one 2D texture at the same time. Unfortunately, it does not allow to subtract from the frame buffer which is necessary for filter kernels containing negative values. As filter kernels we have used a cubic B-spline and a cubic Catmull-Rom spline of width four. We have also tested windowed sinc filter kernels with a Blackman and a Kaiser window.

We have reconstructed object-aligned planar slices on both a GeForce 2 and a GeForce 3, as well as the Radeon. Figure 6 (colorplate) shows slices reconstructed with different filters together with magnified regions to highlight the differences. The Blackman windowed sinc and Catmull-Rom kernels can only be used on the GeForce, due to the fact that the Radeon does not support frame buffer subtraction. We have tested our approach for reconstructing arbitrarily oriented slices on the Radeon and the GeForce 3. On the Radeon we emulated the missing second 3D texture in software, which prevents interactive performance. On the GeForce 3 we were able to achieve frame rates up to eight fps. On this platform we also implemented two different approaches differing in performance and quality by exploiting register combiners in different ways. Table 1 shows some timing results of our test implementation. We have used a Pentium 3/733, 512MB of RAM, with the three graphics cards described above. Note that the timings do not depend on the shape of the filter kernel per se, only on its width.

An important consideration in practice is that all rendering algorithms employing multiple passes are prone to artifacts due to limited frame buffer precision and range and thus special care has to be taken to choose a pass ordering avoiding to exceed the $[0, 1]$ range for intermediate results. Also, in order to be able to allow negative values in filter tiles it is necessary to store absolute values in the corresponding textures and subtract from the frame buffer instead of adding to it.

5 Conclusions and future work

We have presented a general approach for high-quality filtering that is able to exploit hardware acceleration for reconstruction with arbitrary filter kernels. Conceptually, the method is not constrained to a certain dimensionality of the data, or the shape of the filter kernel. In practice, limiting factors are the number of rendering passes and the precision of the frame buffer. Our method is quite general and can be used for a lot of applications. With regard to volume visualization, the reconstruction of object-aligned, as well as oblique slices through volumetric data is especially interesting. Reconstruction of slices can also be used for direct volume rendering. We are exploiting commodity graphics hardware, multi-texturing, and multiple rendering passes. The number of passes is a major factor determining the resulting performance. Therefore, future hardware that supports high num-

bers of textures at the same time – not only 2D textures, but also 3D textures – will make the application of our method more feasible for real-time visualization, even if many slices need to be rendered. In the future we would like to experiment with additional filter kernels and apply our approach to a slice-based volume renderer, as well as image processing.

Please see http://www.VRVis.at/vis/research/hq-hw-reco/ for high-resolution images and the most up-to-date information on this ongoing work.

6 Acknowledgments

Parts of this work have been carried out as part of the basic research on visualization at the VRVis Research Center (http://www.VRVis.at/vis/), which is funded in part by an Austrian research program called Kplus.

References

[1] B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proc. of IEEE Symposium on Volume Visualization*, pages 91–98, 1994.

[2] T. J. Cullip and U. Neumann. Accelerating volume reconstruction with 3D texture mapping hardware. Technical Report TR93-027, Department of Computer Science, University of North Carolina, Chapel Hill, 1993.

[3] F. Dachille, K. Kreeger, B. Chen, I. Bittner, and A. Kaufman. High-quality volume rendering using texture mapping hardware. In *Proc. of Eurographics/SIGGRAPH Graphics Hardware Workshop 1998*, 1998.

[4] M. Hopf and T. Ertl. Accelerating 3D convolution using graphics hardware. In *Proc. of IEEE Vis '99*, pages 471–474, 1999.

[5] R. G. Keys. Cubic convolution interpolation for digital image processing. *IEEE Trans. Acoustics, Speech, and Signal Processing*, ASSP-29(6):1153–1160, December 1981.

[6] S. R. Marschner and R. J. Lobb. An evaluation of reconstruction filters for volume rendering. In *Proc. of IEEE Vis '94*, pages 100–107, 1994.

[7] M. Meißner, U. Hoffmann, and W. Straßer. Enabling classification and shading for 3D texture mapping based volume rendering. In *Proc. of IEEE Vis '99*, pages 207–214, 1999.

[8] M. Meißner, J. Huang, D. Bartz, K. Müller, and R. Crawfis. A practical evaluation of four popular volume rendering algorithms. In *Proc. of IEEE Symposium on Volume Visualization*, pages 81–90, 2000.

[9] D. P. Mitchell and A. N. Netravali. Reconstruction filters in computer graphics. In *Proc. of SIGGRAPH '88*, pages 221–228, 1988.

[10] T. Möller, R. Machiraju, K. Müller, and R. Yagel. Classification and local error estimation of interpolation and derivative filters for volume rendering. In *Proc. of IEEE Symposium on Volume Visualization*, pages 71–78, 1996.

[11] T. Möller, R. Machiraju, K. Müller, and R. Yagel. Evaluation and Design of Filters Using a Taylor Series Expansion. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):184–199, 1997.

[12] T. Möller, K. Müller, Y. Kurzion, Raghu Machiraju, and Roni Yagel. Design of accurate and smooth filters for function and derivative reconstruction. In *Proc. of IEEE Symposium on Volume Visualization*, pages 143–151, 1998.

[13] A. V. Oppenheim and R. W. Schafer. *Digital Signal Processing*. Prentice Hall, Englewood Cliffs, 1975.

[14] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive volume rendering on standard PC graphics hardware using multi-textures and multi-stage rasterization. In *Proc. of Eurographics/SIGGRAPH Graphics Hardware Workshop 2000*, 2000.

[15] T. Theußl, H. Hauser, and M. E. Gröller. Mastering windows: Improving reconstruction. In *Proc. of IEEE Symposium on Volume Visualization*, pages 101–108, 2000.

[16] K. Turkowski. Filters for common resampling tasks. In A. Glassner, editor, *Graphics Gems I*, pages 147–165. Academic Press, 1990.

[17] R. Westermann and T. Ertl. Efficiently using graphics hardware in volume rendering applications. In *Proc. of SIGGRAPH '98*, pages 169–178, 1998.

[18] L. Westover. Footprint evaluation for volume rendering. In *Proc. of SIGGRAPH '90*, pages 367–376, 1990.

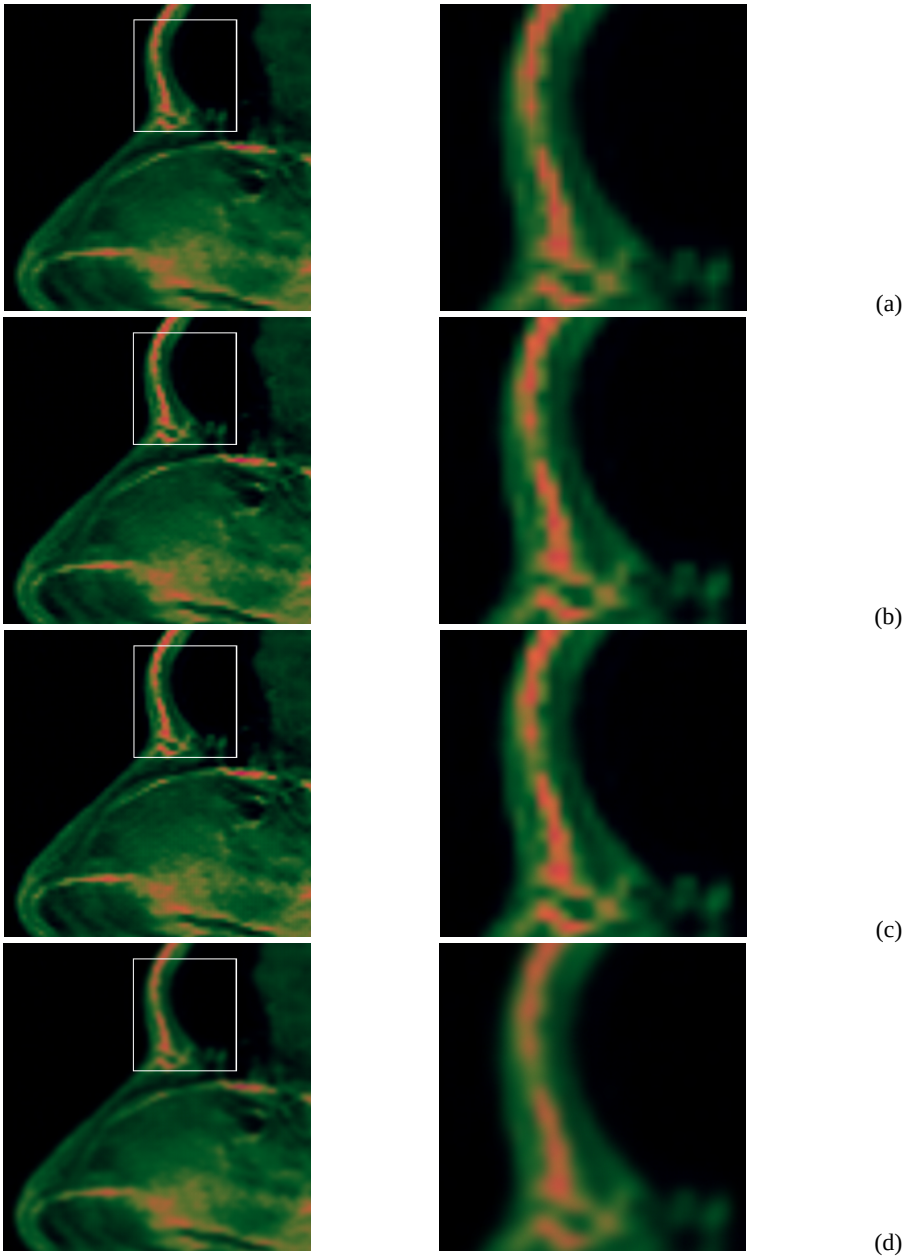


Figure 6: Slice from MR data set, reconstructed using different filters (right images show marked area of left image enlarged): (a) Bilinear filter; (b) Blackman windowed sinc; (c) Bicubic Catmull-Rom spline; (d) Bicubic B-spline (smoothing filter).