Advanced Techniques for High Quality Multiresolution Volume Rendering

S. Guthe and W. Strasser
WSI/GRIS, University of Tuebingen

Abstract

We present several improvements for compression based multi-resolution rendering of very large volume data sets at interactive to real-time frame rates on standard PC hardware. The algorithm accepts scalar or multi-variant data sampled on a regular grid as input. The input data is converted into a compressed hierarchical wavelet representation in a pre-processing step. During rendering, the wavelet representation is decompressed on-the-fly and rendered using hardware texture mapping. The level-of-detail used for rendering is adapted to the estimated screen-space error. To increase the rendering performance additional visibility tests, such as empty space skipping and occlusion culling, are applied. Furthermore we discuss how to render the remaining multi-resolution blocks efficiently using modern graphics hardware. Using a prototype implementation of this algorithm we are able to perform a high quality interactive rendering of large data sets on a single off-the-shelf PC.

Key words: Viewing algorithms, Data compaction and compression

1. Introduction

Visualization of large volumetric data is a very common task in many areas in medicine, computational physics and various other disciplines. An important technique in interactively exploring these data sets is direct volume rendering. Texture mapping hardware can be used to implement this rendering efficiently [3]. Commodity graphics hardware is capable to perform this rendering technique in real-time on cheap off-the-shelf PCs [5, 11]. The visualization of large data sets (more that 256^3 voxel), however, is still infeasible unless massive parallel hardware is used [1]. Since most conventional texture based rendering approaches are brute-force methods, they require a rendering time and storage costs linear in the size of the data. These costs can be reduced dramatically by using a multi-resolution hierarchy. With this hierarchy the rendering resolution is adapted to the distance to the viewer, as proposed by Lamar et al. [10]. Various optimizations including compression and frequency adaptive level-of-detail selection have been proposed by Guthe et al. [6]. However there is still a wide area of optimizations to be applied. One would like to use the actual screen-space error for selecting a working set of blocks to be rendered in order to improve the image quality. Visibility tests such as block-based empty space skipping and occlusion culling promise an additional speedup.

Using these optimizations, we are able to render high quality walkthroughs of large data sets in real time on a conventional PC. We will demonstrate a real-time walkthrough of the visible human female RGB data set [14] at a resolution of 256^2 pixel. In addition to this we will also demonstrate a walkthrough at a resolution of 1280×960 pixel for the same data set at interactive frame rates.

2. Related Work

Visualizing large volume data sets is a well known problem in computer graphics. In the following section, we will give a brief overview of related work in the area of multi-resolution rendering and wavelet based compression and rendering.

Multi-resolution rendering: The idea of multi-resolution volume rendering algorithms is to provide a spatial hierarchy to adapt the local rendering resolution to the projection onto the screen: An octree or a similar spatial data structure is build for the data set in a pre-processing step. Each node of the spatial hierarchy contains a certain part of the volume within its bounding box at a specific resolution. During rendering, nodes are selected such that their resolution matches the display resolution to avoid loss of detail and aliasing. The technique was first proposed by Chamberlain et al. [4] in the context of rendering surface models. This technique was adapted to volume rendering by Lamar et al. [10]. To avoid discontinuity artifact between different levels-of-detail, Weiler et al. [17] propose an efficient extension to the algorithm that modifies the proxy geometry used for rendering. Guthe et al. [6] introduced a compression scheme to handle even larger data sets at interactive speed. Like Boada et al. [2], they also used a refined error criterion for the selection of octree nodes to speed up the rendering.

Wavelet Based Techniques: A very efficient compression scheme has been proposed by Nguyen et al. [13]. They use a blockwise compression: The volume is split into several small blocks of equal size that are then compressed individually. For rendering of large data sets with wavelet-based representations, two directions for post-classified volume data have been followed up: Firstly, several raycasting techniques were proposed that operate on a wavelet representation [7, 9, 15, 18]. However, raycasting of large data sets is not possible at interactive frame rates unless massive parallel hardware is used [1]. In contrast to this, Guthe et al. [6] use a block based wavelet compression and graphics hardware to render the data sets at interactive frame rates.

3. Multi-resolution Rendering Algorithm

In a preprocessing step the volume is converted and compressed into a level-of-detail hierarchy, similar to Guthe et al. [6].

In order to encode large amounts of zero values effectively, we construct an octree over the high pass coefficients of each block where a zero in an inner node marks all children as zero values. This approach is similar to the embedded zerotree [16]. After encoding this tree we only need to compress all non-zero coefficients. The compression of the remaining coefficients is done using a fixed huffman encoder [6]. Furthermore all leaf nodes that don't contain any data are completely removed from the octree.

To find the correct resolution for each block, the level-of-detail selection algorithm always starts with the root node and refines the octant of a single node that produces the highest error. This refinement stops if a maximum number of blocks is reached. In order to achieve the highest possible quality however, this view dependent refinement must not split any invisible blocks, but rather remove them from the list of the blocks to be rendered. This does not only include blocks outside the view frustrum, but also hidden or completely transparent blocks, even if they contain high frequencies.

While the transparency and the position outside the view frustrum are easily detected for during refinement, the amount of occlusion for each block can not be calculated easily. Therefore we did not include the occlusion test to

increase the quality, but rather to speed up the rendering only.

4. Error Estimation

To increase the rendering quality without any further impact on the performance of our algorithm, we will first discuss how to choose the optimal working set for rendering. To find this set we extend the simple greedy strategy by finding the block that produces the most error and replace it with higher resolution blocks. We therefore have to compare the rendered image using the current set against the image rendered using the blocks that map one voxel to a single pixel. As you can easily see this is even slower than rendering the image in full resolution in the first place, so we have to estimate the screen-space error in some way.

For estimating the screen-space error, we first have to define an appropriate error metric. The screen-space error we use is defined as the difference between the final color of a pixel using a low resolution block compared against the final color using the highest applicable resolution. This difference is the sum of the three color channels and the remaining transparency of each block. This error therefore depends on the current transfer function (color C and density D) in a natural way. In order to reduce the error, we are only interested in the maximum error produced by each block multiplied by its importance, i.e. size in screen-space. Since the maximum error per block err is the difference between the color err_C plus the difference between the opacity err_O, it can be calculated by

$$
\begin{aligned}
err_O &= \left| e^{-\int_0^d D\left(V_l(x)\right)\mathrm{d}x} - e^{-\int_0^d D\left(V_h(x)\right)\mathrm{d}x} \right|, \\
err_C &= \left| \int_0^d D\big(V_l(x)\big) C\big(V_l(x)\big) e^{-\int_0^x D\left(V_l(x)\right)\mathrm{d}t} \right. \\
&\qquad \left. - D\big(V_h(x)\big) C\big(V_h(x)\big) e^{-\int_0^x D\left(V_h(x)\right)\mathrm{d}t} \mathrm{d}x \right|, \\
err &= err_O + err_C .
\end{aligned}
$$

To calculate this error, both integrations have to be carried out for each pixel in the final image. Since this is equivalent to render the complete image twice using raycasting, we have to use a conservative estimation. Let us assume that we know each voxel in the high resolution blocks and the corresponding voxel value in the low resolution block. Since the integral is a linear operation, we can estimate the error as the maximum over all possible color and opacity combinations given by

$$
\begin{aligned}
err'_O &= \left| e^{-d\,D_l} - e^{-d\,D_h} \right|, \\
err'_C &= \left| C_l \left(1 - e^{-d\,D_l}\right) - C_h \left(1 - e^{-d\,D_h}\right) \right|, \\
err &\leq err'_O + err'_C .
\end{aligned}
$$

However, we would still have to decompress the corresponding child block. Storing the maximum deviation of each voxel value in the low resolution block, we no longer need the high resolution data and can simplify the estimation even more. Using C_{min}, C_{max}, O_{min} and O_{max} as the component wise minimum and maximum of the transfer function within the deviation we get.

$$
\begin{aligned}
err''_O &= e^{-d\,D_{min}} - e^{-d\,D_{max}} \\
err''_C &= (C_{max} - C_{min}) \left(1 - e^{-d\,D_{max}}\right) + C_{min}\, err''_O \\
err &\leq err''_O + err''_C
\end{aligned}
$$

With the maximum of err''_O at $d = \frac{\ln \frac{D_{max}}{D_{min}}}{D_{max} - D_{min}}$ and the maximum of err''_C at d being the diagonal of the block. If the value of d for the calculation of err''_O is larger than the diagonal of the block, eg. $D_{min} = 0$, we also use the diagonal for calculating this error.

Storing the values C_{min}, C_{max}, O_{min} and O_{max} into a two-dimensional table, we are now able to compute the maximum error for all voxel values present in the low resolution block, but the amount of memory we would need for that is still twice the size of the decompressed voxels. To store the deviation of all voxel values efficiently, we build a small histogram that contains the maximum positive and negative deviation for a small range of values. In practise a histogram with a fixed number of 8 entries showed up to be very efficient. The calculation of the estimated screen-space error is very costly since all possible combinations for each voxel value in the given range have to be considered. To speed up the calculation, we use another two-dimensional table that stores the minimum and maximum colors and opacities for a given range. Therefore only a single calculation has to be done for each interval in the given range of voxel values. On the other hand this again leads to a certain overestimation. Since the errors for a given resolution only change whenever the transfer function changes, we can speed up the calculation by storing the error produce by each octant prior to the multiplication with its size in screen-space with the decompressed block. The resulting space usage of all additional data is only 68 bytes per compressed block and an additional 40 bytes for each uncompresssed block.

5. Visibility Testing

After selecting the appropriate resolution for each block, the visible blocks have to be rendered in a front-to-back or back-to-front order. During this traversal each block can be tested wether it is completely transparent under the current transfer function. If so, it is skipped completely without any further processing. The reduced depth complexity can be seen in Figure 1. Using this technique, the rendering performance increases by about 35%. A per sample alpha test after applying the transfer function can be used to skip the alpha blending, however this only delivers a speedup of another 1%.

Up to now, we are still waiting more than 80% of the time for the graphics hardware, so we have to remove more invisible blocks, in this case the occluded ones. Taking occlusion into account on a per block basis is a bit more complicated than detecting transparency, especially if we use a back-to-front rendering order. Therefore the occlusion calculation is done prior to the rendering. Calculating the exact occlusion of each block is similar to render a complete image using raycasting. Although no shading of any kind has to be taken into account, this step is too slow to gain a speedup. However we don't need an exact occlusion information, but rather a number of blocks that are not visible. In order to approximate the occlusion, we assume a uniform opacity for each block, i.e. to be on the safe side, we use its minimum opacity. Then a software raycaster based on cell projection, similar to the one presented by Mora et al. [12], is used to calculate an occlusion map at a fixed resolution. During the cell projection there are some cases to consider. Blocks that have a minimum opacity below a certain threshold only have to be tested for their visibility since they don't change the occlusion map. This test can also be skipped until the first block with an opacity above this threshold has been processed. If a block is found to be occluded, we also won't need to update the occlusion map. Therefore only very few updates of the occlusion map are typically necessary. With our current implementation based on optimized SSE2 code, an occlusion map of 256^2 pixel showed up to be a good value for all possible cases, ranging from no occlusion to very high occlusion.

Although this is a very rough approximation, the depth complexity can be reduced, as seen in Figure 1. To reduce the traversal costs the block based empty space skipping is done during the selection of the level-of-detail. Each of these optimizations reduces the number of shaded samples by about 30%. The transfer function used for the example in Figure 1 is not an iso-surface transfer function, but rather a linear ramp.

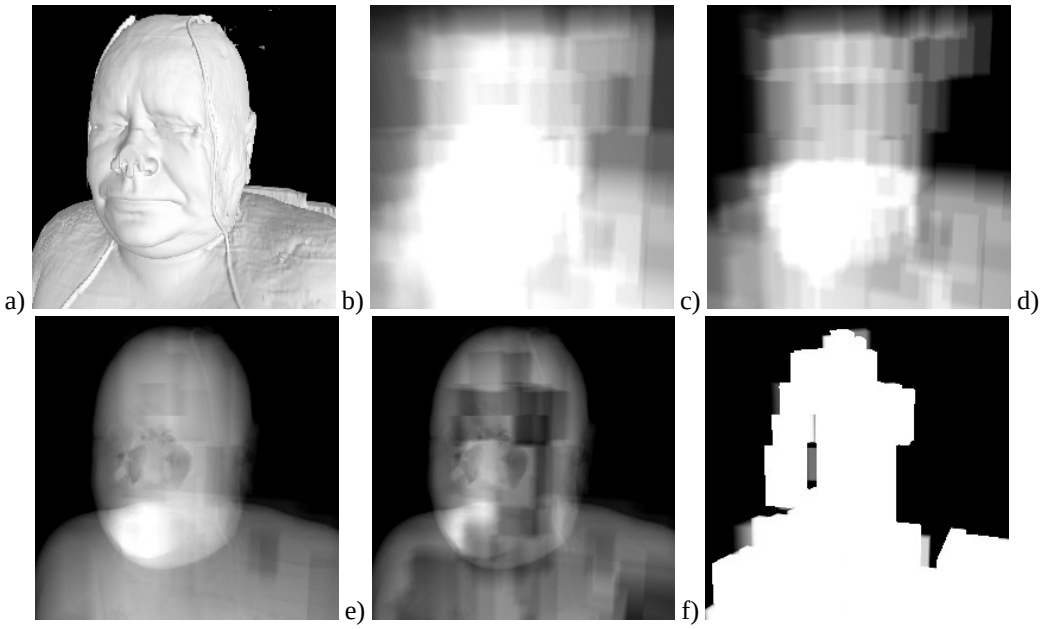

Fig. 1. Depth complexity for a given image (a) without optimizations (b), with block based empty space leaping (c), alpha test (d) and occlusion culling (e) with corresponding occlusion map (f). About 30% of all sample points have been removed with each optimization.

6. Rendering

Current graphics hardware supports a variety of different possibilities for rendering the selected blocks. But there are some performance issue whenever selecting a rendering scheme. Using 3D textures for the volume data in order to implement a high quality shader we have to keep in mind that the principle viewing direction should be aligned with the w-axis of the texture. Otherwise the rendering would be slowed down unnecessarily. However aligning all blocks to the principle viewing direction will result in uploading all textures whenever the direction changes. To circumvent this, we align the 3D textures to the principle direction of the vector pointing from the viewer to the center of the block. Therefore only few textures have to be re-aligned per frame. In addition to this these aligned textures can also be used for rendering with object aligned slices requiring only 2D textures.

Since our algorithm does not depend on a certain kind of rendering technique, we can choose from most of the existing approaches. The pre-integrated rendering presented by Engel et al. [5] produces the best image quality with our multi-resolution setting. However it also results in the lowest frame rate. The quality of simple post-classification rendering is a bit lower, but the loss of quality for reducing the number of textures is a lot worse. Both of these approaches also have to evaluate a per-pixel lighting calculation that further limits the frame rate. In contrast to this pre-classification algorithms only need do calculate the lighting on a per-voxel basis. Since the lighting only changes whenever the texture is replaced or the light source changes, it can be evaluated prior to the texture upload. Using a 3D texture to store the lit voxels already increases the frame rate to a highly interactive level, however using a 2D texture and object aligned slices even results in real-time rendering for a viewport of 256^2. Even an interactive navigation through the data set at a full-screen resolution of 1280×960 pixel is possible.

With the exception of object aligned slices, the proxy geometry used for rendering changes every frame. For block based rendering with 6144 individual 3D textures, the number of triangle easily exceeds 500k. In order to keep the transfer of data to the GPU as low as possible, we move all per-vertex computations to the vertex shader. Rendering with object aligned slices, the geometry only changes for textures that have been uploaded for the current frame. Either because they were re-aligned or a new level-of-detail was selected. For all blocks taken

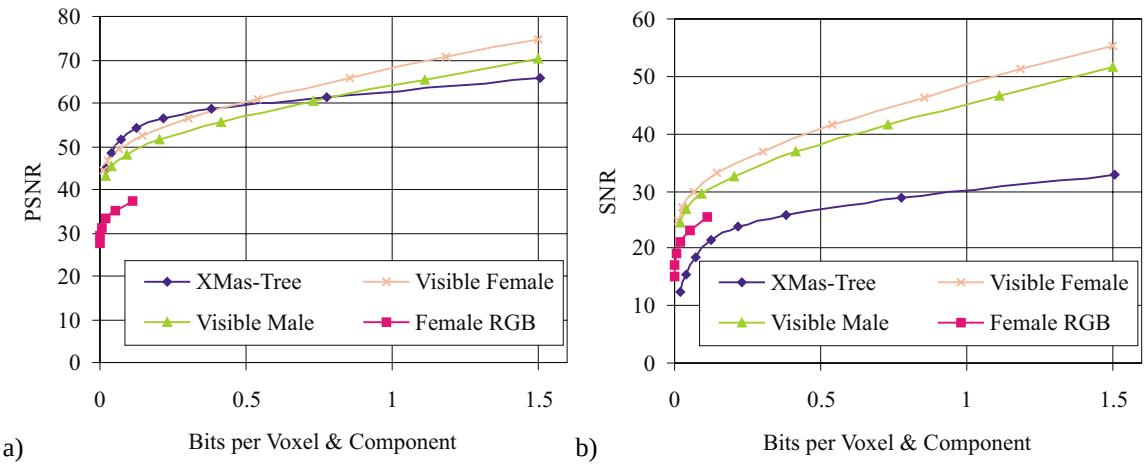

Fig. 2. Bits per voxel against PSNR (a) and SNR (b) for sample data sets.

from the previous frame, the geometry did not change and can be used again.

In general the best image quality at a certain frame rate can always be achieved using object aligned slices. Using a fixed number of textures however, pre-integrated rendering always produces the highest quality images. This is especially true for changing levels-of-detail during an interactive walk-through since the rendering is less prone to producing popping artifacts.

7. Results

In this section, we discuss the results obtained with a prototype implementation of our algorithm. The algorithm was implemented in C++ using using DirectX9 for rendering. All benchmarks were performed on a 2.6Ghz Pentium 4 PC with 2GB of main memory and an ATI Radeon 9700 Pro graphics adapter with 128MB of local video memory.

To evaluate our algorithm, we use four different data sets that are all too large to be visualized at interactive frame rates using conventional brute-force rendering approaches. The smallest data set is a computer tomography scan of a small Christmas tree [8] at a resolution of $512 \times 512 \times 999$ voxel with 12 bits per voxel and a spacing of $0.93mm \times 0.93mm \times 0.5mm$. The other two scalar data sets are the visible human male and female data sets [14]. The male data set has a resolution of $2048 \times 1216 \times 1877$ voxel and the female data set has $2048 \times 1216 \times 1734$ voxel. Both data sets have a spacing of $0.33mm \times 0.33mm \times 1.0mm$. The visible female cryosection RGB volume has a resolution of $2048 \times 1216 \times 5189$ voxel with 24 bits per voxel and a spacing of $0.33mm$.

7.1. *Compression Efficiency*

The compression ratios we achieve at a given peak-signal to noise ratio (PSNR) or signal to noise ratio (SNR) are among the best results for non-interactive compression techniques, such as the one proposed by Nguyen et al. [13]. The actual figures for the example data sets can be seen in Figure 2a and Figure 2b. The decompression speed is, due to various optimizations, at around 100MB per second. The degeneration of the final image due to compression can be seen in Figure 3.

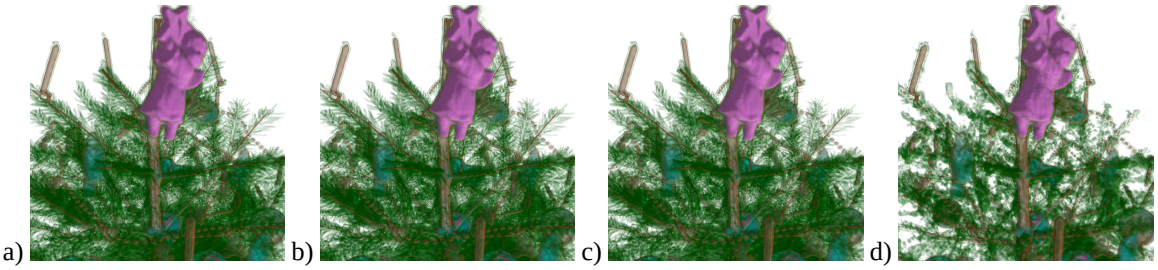

Fig. 3. Christmas tree compressed using lossless compression (a, comp. ratio 3.8:1), lossy compression at PSNR of 65.2 dB (b, comp. ratio 10.6:1), PSNR of 56.5 dB (c, comp. ratio 72.8:1) and PSNR of 51.4 dB (d, comp. ratio 224:1).

	pre-int.	post-cls.	pre-cls.	object
Christmas	9.1 fps	14.6 fps	22.8 fps	38.3 fps
Female	6.7 fps	9.5 fps	15.4 fps	28.7 fps
Male	5.3 fps	7.6 fps	11.6 fps	21.2 fps
Fem. RGB	n.a.	n.a.	10.9 fps	15.5 fps

Table 1
Frames per second for highest quality at 256^2 viewport (pre-integration, post-classification, pre-classification at a sampling distance of 0.7 voxel and pre-classified object aligned slices).

Fig. 4. Christmas tree data set rendered with pre-integration (a) and simple post-classification (b). Visible human female cryosection data set (RGB slices) rendered with post-classification with either view plane aligned slices (c) or object aligned slices (d).

7.2. *Interactive Walkthroughs*

The screen-space error estimation combined with the visibility tests allow for interactive navigation at a resolution of 256^2 with all rendering settings and 1280×960 pixels using the object aligned slices. The frame rates we could achieve for all three scalar test data sets are at least 5 frames per second. Changing the rendering to a simple post-classification increases the rendering performance by approximately 48% as seen in Table 1. This is because the volume has to be sampled only once per slice and the dependent lookup is only one-dimensional. Pre-classified rendering increases the frame rate to at least 7.5 frames per second. The average increase in frame rate is approximately 57%. Changing the rendering to object aligned slices and 2D textures the frame rate again increases significantly. The average speedup for this approach is approximately 70%. So we are well in the area of real-time rendering for the low resolution window. Rendering at a resolution of 1280×960 however is a lot slower. Since the object aligned slices needs the least amount fill rate, we will render the data sets in full screen mode using this approach. Beside the increased frame rate, both the simple post-classification rendering and the object aligned slices reduce the quality of the final image. Although the introduced artifacts are clearly visible, as seen in Figure 4, they are easier to be tolerated than a reduced amount of detail.

	640×480	800×600	1280×960
Christmas	19.9 fps	19.5 fps	13.9 fps
Female	12.4 fps	11.7 fps	9.7 fps
Male	11.5 fps	10.9 fps	7.5 fps
Fem. RGB	10.1 fps	9.4 fps	7.0 fps

Table 2
Frames per second for object aligned slices at different full screen resolutions.

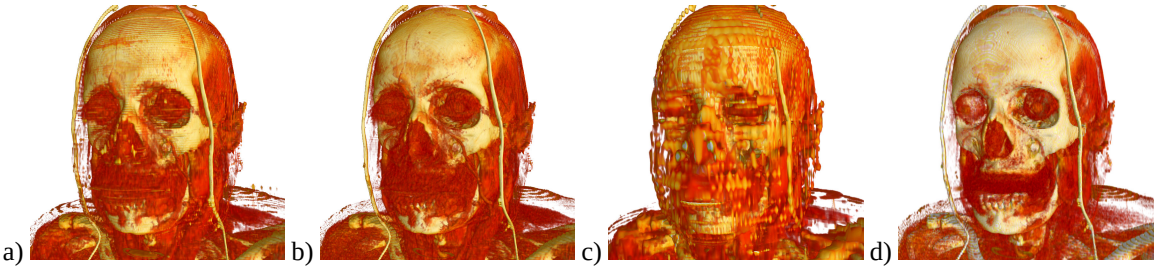

Fig. 5. Frequency based level-of-detail selection using pre-integration (a and c) against screen-space error driven selection with all optimizations using pre-integration (b) or object aligned pre-classified slices (d). (a) and (b) render at approximately 4 frames per second, (c) and (d) render at 10 frames per second at 800^2 pixels.

Using object aligned slices, we are able to render all data sets in full screen on a single off-the-shelf PC, the exact timing for different resolutions can be seen in Table 2. Even for the Visible human female RGB data set we were still able to achieve a frame rate of 7 frames per second which is still interactive.

The timings for post-classified volume rendering are divided into the following tasks. 8% of the rendering time is consumed by the decompression of blocks and the calculation of the corresponding screen-space error. The texture generation and uploading needs about 7%. The overhead for the occlusion culling is about 1% on the average if no occlusion was found. The geometry setup and transfer to the graphics card consumes about 16% of the rendering time. In contrast to this the textures rarely change and therefore need less bandwidth. Finally the actual rendering takes about 68% of the total time. This is the additional time that the GPU needs for rendering. This also explains the higher frame rates for using pre-classification in Table 1.

7.3. *Error Estimation*

The increased image quality for our estimated screen-space error can hardly be measured, but can clearly be seen in the following examples.

We compare the pre-integrated rendering of the approach presented by Guthe et al. [6] with our screen-space error optimization combined with the advanced visibility tests. For a comparison we use images that render at approximately the same frame rate. First we take a look at a closeup view of the Visible human female data set. Using pre-integrated rendering we already gain a higher image quality and a lot of detail has been added to the image (compare Figure 5a and b). Switching to object aligned slices using pre-classification, the increase in image quality is clearly becoming visible. (compare Figure 5c and d).

Using the Visible human male data set for another comparison, the increase in image quality is as dramatic as in the previous example. The added amount of detail for pre-integrated rendering is again quiet noticeable (compare Figure 6a and b). For object aligned slices there is a huge increase in image quality. While small details in Figure 6c are not visible at all, the image in Figure 6d shows the same details as the pre-integrated rendering (see Figure 6b). Although the image quality is not as high, all details are still preserved at a higher frame rate.

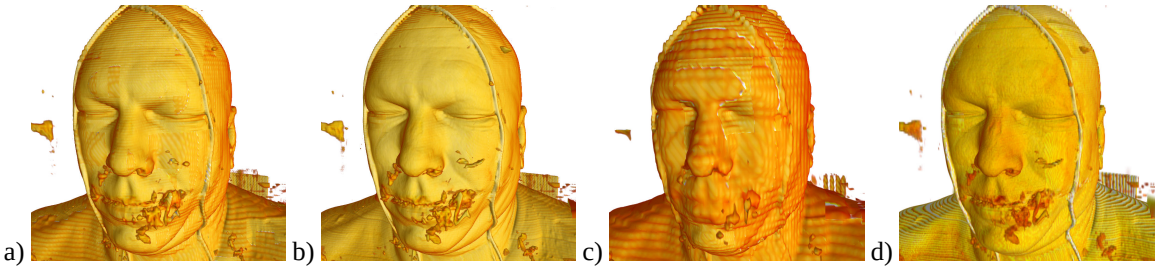

Fig. 6. Frequency based level-of-detail selection using pre-integration (a and c) against screen-space error driven selection with all optimizations using pre-integration (b) or object aligned pre-classified slices (d). (a) and (b) render at approximately 3.5 frames per second, (c) and (d) render at 9 frames per second at 800^2 pixels.

Beside these improvements in image quality, the calculation of the screen-space error does not consume any additional time during rendering. The overall rendering performance even increases using the estimated screen-space error, since large regions are rendered using blocks of very low resolution.

8. Conclusions and Future Work

We presented a rendering algorithm for visualizing very large volume data sets. The algorithm uses a hierarchical wavelet representation to store large data sets in main memory. An error metric that minimizes the screen-space error is used to determine the correct level-of-detail. Empty space skipping and occlusion culling are used to speed up rendering. We are able to render multi-variant data sets of approximately 40 GB at 14 frames per second. Smaller data sets can be rendered even at more than 30 frames per second at a very high quality setting.

We believe that the compressed representation will be useful in an out-of-core scenario, too. Including a simplified occlusion test in the level-of-detail selection could increase the rendering quality. Since the estimated screen-space error is a lot larger than the actual one, we would also like to consider other ways of estimating the screen-space error, that may or may not be conservative.

9. Acknowledgements

The authors would like to thank Michael Wand and Gunter Knittel for valuable discussions on the topic of projective error estimation and Michael Dogget for supplying the Radeon 9700. This work has been funded by the SFB grant 382 of the German Research Council (DFG).

References

[1] C. Bajaj, I. Ihm, G. Koo, and S. Park. Parallel Ray Casting of Visible Human on Distributed Memory Architectures. In *Data Visualization*, Eurographics, pages 269–276, May 1999.

[2] I. Boada, I. Navazo, and R. Scopigno. Multiresolution Volume Visualization with a Texture-Based Octree. In *The Visual Computer*, volume 17(3), pages 185–197. Springer, 2001.

[3] B. Cabral, N. Cam, and J. Foran. Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. In *Workshop on Volume Visualization*, pages 91–98, October 1994.

[4] B. Chamberlain, T. DeRose, D. Lischinski, D. Salesin, and J. Snyder. Fast Rendering of Complex Environments Using a Spatial Hierarchy. In *Graphics Interface*, pages 132–141, May 1996.

[5] K. Engel, M. Kraus, and T. Ertl. High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, August 2001.

[6] S. Guthe, M. Wand, J. Gonser, and W. Straßer. Interactive Rendering of Large Volume Data Sets. In *IEEE Visualization*, pages 53–60, October 2002.

[7] I. Ihm and S. Park. Wavelet-Based 3D Compression Scheme for Very Large Volume Data. In *Graphics Interface*, pages 107–116, June 1998.

[8] A. Kanitsar, T. Theußl, L. Mroz, M. Šrámek, A. V. Bartrolí, B. Csébfalvi, J. Hladůvka, D. Fleischmann, M. Knapp, R. Wegenkittl, P. Felkel, S. Röttger, S. Guthe, W. Purgathofer, and E. Gröller. Christmas Tree Case Study: Computed Tomography as a Tool for Mastering Real World Objects with Applications in Computer Graphics. In *IEEE Visualization*, pages 489–492, October 2002.

[9] T. Kim and Y. Shin. An Efficient Wavelet-based Compression Method for Volume Rendering. In *Pacific Graphics*, pages 147–157, October 1999.

[10] E. C. LaMar, B. Hamann, and K. I. Joy. Multiresolution Techniques for Interactive Texture-Based Volume Visualization. In *IEEE Visualization*, pages 355–362, October 1999.

[11] M. Meißner, S. Guthe, and W. Straßer. Interactive Lighting Models and Pre-Integration for Volume Rendering on PC Graphics Accelerators. In *Graphics Interface*, pages 209–218, May 2002.

[12] B. Mora, J.-P. Jessel, and R. Caubet. A New Object-Order Ray-Casting Algorithm. In *IEEE Visualization*, pages 203–210, October 2002.

[13] K. G. Nguyen and D. Saupe. Rapid High Quality Compression of Volume Data for Visualization. *Computer Graphics Forum*, 20(3), 2001.

[14] The National Library of Medicine. The Visible Human Project. *http://www.nlm.nih.gov/research/visible/visible_human.html*, 1987.

[15] F. Rodler. Wavelet based 3D Compression with Fast Random Access for Very Large Volume Data. In *Pacific Graphics*, pages 108–117, October 1999.

[16] J. M. Shapiro. An Embedded Hierarchical Image Coder Using Zerotrees of Wavelet Coefficients. In *IEEE Data Compression Conference*, pages 214–233, April 1993.

[17] M. Weiler, R. Westermann, C. Hansen, K. Zimmermann, and T. Ertl. Level-of-Detail Volume Rendering via 3D Textures. In *IEEE/SIGGRAPH Symposium on Volume Visualization*, pages 7–13, October 2000.

[18] R. Westermann. A Multiresolution Framework for Volume Rendering. In *IEEE/SIGGRAPH Symposium on Volume Visualization*, pages 51–58, October 1994.