Level-Of-Detail Volume Rendering via 3D Textures

Manfred Weiler*, Rüdiger Westermann*
Chuck Hansen†, Kurt Zimmerman†, Thomas Ertl*

*Visualization and Interactive Systems Group
University of Stuttgart

†Scientific Computing and Imaging Institute
University of Utah

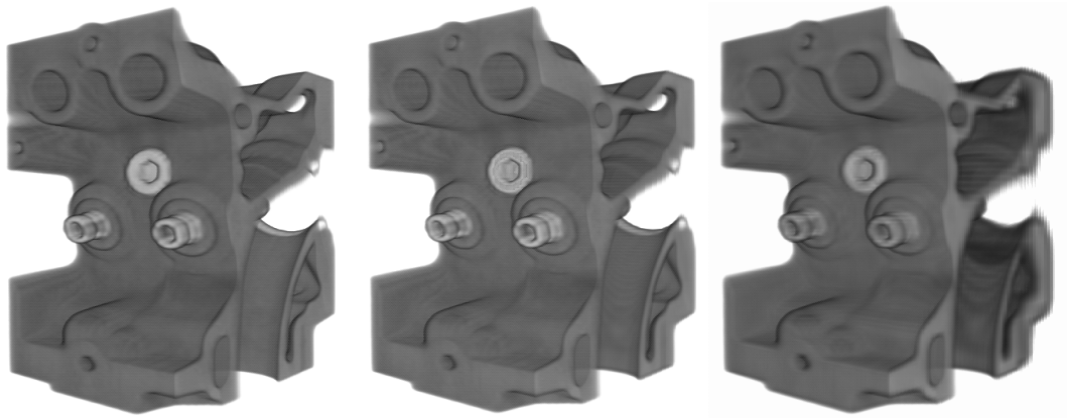

Figure 1: *All images show 3D texture based volume rendering of the engine data set. On the left, the data set is displayed with full resolution. In the middle, two different levels of detail are used with lower resolution in the back. This reduces texture memory consumption to 57%. On the right, the adaptive representation using four levels of detail from front-to-back requires only 29% of the original texture memory. Here, the data set was rendered with only 32% of the originally needed number of texture lookups.*

Abstract

In this paper we present an adaptive approach to volume rendering via 3D textures at arbitrary levels of detail. The algorithm has been designed to enable interactive exploration of large-scale data sets while providing user-adjustable resolution levels. A texture map hierarchy is constructed in a way that minimizes the amount of texture memory with respect to the power-of-two restriction imposed by OpenGL implementations. In addition, our hierarchical level-of-detail representation guarantees consistent interpolation between different resolution levels. Special attention has been paid to the fixing of rendering artifacts that are introduced by non-corrected opacities at level transitions. By adapting the sample slice distance with regard to the desired level-of-detail, the number of texture lookups is reduced significantly.

*Universität Stuttgart, IfI, Abt. VIS, Breitwiesenstr. 20-22, 70565 Stuttgart, Germany; E-mail: Manfred.Weiler@informatik.uni-stuttgart.de.

1 Introduction

Volume data sets most commonly occur in two fields: imaging and computational science. 3D imaging devices continue to increase the resolution of their sampled volumes with the current generation approaching data volumes of 1024^3 samples. Similarly, computational science continues to increase the mesh resolutions for large scale simulation thereby increasing the size of the data to be visualized. The challenge is to provide interactive visualization of these large 3D scalar fields while avoiding rendering artifacts.

Hardware assisted volume rendering can provide interactive visualization of 3D scalar fields[2, 3, 8, 10]. The ability to interact with transfer functions and viewpoint orientation provides powerful visual cues that would be difficult to reproduce in batch-mode volume rendering. 3D texture mapping hardware that is now also available on PC graphics adapters [6] is the most prevalent choice for hardware assisted volume rendering although dedicated volume processors have recently been introduced[10]. While 3D texture mapping is a powerful tool that allows one to investigate volumes easily, either by direct volume rendering, or by visualizing these functions on intermediate surfaces, the limited amount of texture memory is a serious constraint. Methods exist for performing volume rendering where the entire data volume resides in texture memory[2, 3]. If one exceeds the limits of physical texture memory, some graphics libraries allow for the paging of textures[14, 5]. However, such brute-force methods for dealing with volumes whose size exceeds physical texture memory severely hamper the interactivity of the rendering.

Fortunately, most data sets have large regions that don't contain interesting data. This paves the way for the application of multiresolution representations of the volume data. Such representations allow regions of interest to be rendered at higher resolutions than other parts of the data set, allowing interactive rendering of data whose uniform grid is much larger than texture memory. A second benefit is the reduction of trilinear interpolations. Since coarser levels provide a filtered, more compact, representation of the original data, resampling based upon the level-of-detail is desirable because it reduces the number of expensive trilinear interpolations. The problem with multiresolution methods is the introduction of rendering artifacts when adjoining regions differ in the level-of-detail.

This paper describes a method for hierarchically subdividing the data. Care is taken to insure interpolation consistency between levels while maintaining a minimal amount of data replication. Even with the interpolation consistency, rendering artifacts can occur at the boundaries between levels. We present a description of these errors and describe a solution which provides continuity at level boundaries. In the next section, we briefly describe related work. In section 3, we describe the level-of-detail representation. Section 4 provides insight into the rendering artifacts previously seen in multiresolution volume rendering. We conclude with some results and areas of future research.

2 Related work

As described in the introduction, hardware assisted volume rendering has been a reality for the past several years[1, 2, 10]. While dedicated hardware is now commercially available[10], 3D texture mapping approaches are the most prevalent due the vast numbers of machines, mainly SGIs, upon which this OpenGL extension runs. Fundamentally, these systems resample volume data, represented as a 3D texture, onto a sampling surface. The most common surface is a plane that can be aligned with the data, aligned orthogonal to the viewing direction, or aligned in other configurations (such as spherical shells). The ability to leverage the embedded trilinear interpolation hardware is at the core of this acceleration technique.

This capability was first described by Cullip and Neumann[3]. They discussed the necessary sampling schemes as well as axis aligned and viewpoint aligned sampling planes. Further development of this idea, as well as the extension to more advanced medical imaging, was described by Cabral et al.[2]. They demonstrated that both interactive volume reconstruction and interactive volume rendering was possible with hardware providing 3D texture acceleration. Further improvements providing non-polygonal surface rendering and effective handling of arbitrary clipping geometries have been proposed in [12].

The work most similar to ours was reported by LaMar et al.[7]. They described a multiresolution approach to interactive volume rendering. The use of multiresolution for importance based volume rendering is well motivated by LaMar et al. Their multiresolution hierarchy filtered the volume to create levels-of-detail in an octree, but they did not explicitly address the avoidance of interpolation errors in their multiresolution model. They investigated several slicing methods and concluded that spherical shells were an approach to deal with rendering artifacts. Their solution did not guarantee continuity between levels but attempted to reduce the visual artifacts through spherical sampling.

Our work differs from theirs in that we develop a multiresolution hierarchy that allows consistent interpolation between levels. We also address the rendering artifacts that still persist when rendering two differing but adjacent levels. Our multiresolution hierarchy is not strictly based upon an octree. Our work uses slicing planes which are parallel to the image plane rather than spherical shells for rendering efficiency. It should be noted that LaMar's method will allow a much wider field-of-view than our method, but since this is not the common viewing frustum, this is not a severe limitation. In the next section, we describe our multiresolution hierarchy followed by a description and solution to the artifact problem.

3 Level-of-detail texture representation

Since in practical applications the size of the volume data sets is likely to exceed the amount of available texture memory, the data is usually split into subvolumes or bricks that are small enough to fit into texture memory. Each brick can be rendered separately in back-to-front or front-to-back order, but since for every frame all bricks have to be reloaded, the rendering performance decreases considerably.

In order to overcome this limitation we propose a level-of-detail texture representation that provides an alternative for interactive rendering of large-scale data sets. The goal of this hierarchical representation is twofold: to convert the volume data into a multiresolution representation that entirely fits into the limited texture memory *and* to minimize texture lookups by adaptively resampling the data with respect to the selected level-of-detail.

Each brick locally stores approximations of the original data at an ever coarser resolution. These copies are then used to adaptively render arbitrary regions with reduced detail size. Even if the multiscale representation does not entirely fit into texture memory, texture loading will be reduced. In addition a considerable number of rasterization operations can be saved by choosing the sampling frequency according to the detail size present within each copy. In summary, by taking advantage of a level-of-detail representation as described, our goal is improved rendering performance.

In the remainder of this section let us assume that we initially decompose the data set into a number of bricks. After constructing the texture hierarchy these bricks can be rendered at arbitrary resolutions. As will be outlined below, some additional expense is required to guarantee continuous transitions between adjacent bricks at different levels. Criteria that are applied to determine the number of initially selected bricks and the resolution level that is to be used for rendering will be discussed in subsection 3.4.

3.1 Texture decomposition

Prior to the rendering, the volume data set is subdivided into multiple bricks of smaller size, which get assigned the chunk of texture that is necessary to render the brick at the original resolution. Each brick builds its own local hierarchy by constructing copies of the original texture at ever coarser resolution. These different levels-of-detail are stored in additional texture maps as shown in Figure 2.

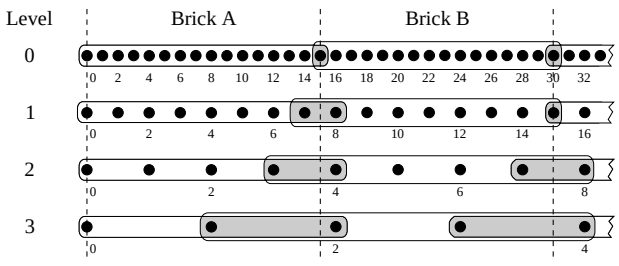

Figure 2: *One-dimensional example of a data set with four levels of detail. On every level the size of texture elements increases by a factor of two. Note that at brick boundaries on the same level texture elements have to be included in multiple bricks in order to guarantee continuous texture interpolation (filled areas). Since textures always have to be specified in powers of two, copies for each brick need to be extended and padded with zero voxels.*

The original data set is represented by textures of level 0. Ascending level indices indicate coarser texture resolution. The width of texture elements on level k is twice the width of elements on level $k-1$, whereas the texture size is reduced by a factor of two. Texture elements on different resolution levels are aligned in such a way, that their centers on a certain level correspond to the center of an even element on the next finer level. Since the geometry or shape of bricks will be retained throughout the hierarchy, the domain of the underlying texture function, necessary to compute appropriate texture coordinates, has to be adapted accordingly.

On the first level, the domain of texture coordinates ranges from the center of the first element to the center of the last one. Due to the alignment of voxels on different levels, borders of the texture function domain on coarser levels often fall between two adjacent voxels. In this case additional voxels are needed in order to guarantee correct interpolation. On the other hand, since the width of texture elements on each level is known and because the shape of bricks is not going to be modified, offsets for correct calculation of texture coordinates can always be determined.

Copies of the original data at coarser resolutions are constructed by iteratively filtering the data. In general, this filter can be chosen arbitrarily, although merely sub-sampling the original data to obtain coarser approximations leads to unsatisfactory results after only a few coarsening steps. In our implementation a quadratic spline kernel was used to consecutively build the low-pass filtered copies.

We should note that it is always necessary to expand textures in order to cope with the power-of-two restriction imposed by the OpenGL implementation. However, a hierarchical technique will be outlined below that allows us to effectively avoid unnecessary texture elements.

3.2 Continuous level transitions

If local texture hierarchies are constructed as described, interpolation artifacts at the boundaries between adjacent bricks at the same level do not appear. This is because boundary voxels are shared by adjacent bricks. However, this does not apply to adjacent bricks rendered on different levels.

Therefore, it is necessary to slightly modify the treatment of level transitions to meet the continuity requirements: in the level-of-detail representation the continuity at level transitions can be established by letting the finer cells to the left and to the right of the brick boundary interpolate the scalar field from the coarser level (Figure 3). For cells with even index in each dimension this is equivalent to a copy operation as they correspond exactly to a cell on the next coarser level. This procedure *only* adapts the brick textures on the finer level. Thus, with the combination of the proposed multiresolution representation together with appropriately re-sampled values at level transitions we guarantee the continuity of the 3D scalar field.

Note in particular, that we restrict transitions to differ by at most *one* level in order to maintain the continuity between levels. However, this does not impose a real restriction as higher transitions can be achieved by consecutive transitions of one level.

The number of voxels that has to be adapted depends on the position of the brick boundary relative to the voxel coordinates. Either the boundary is located at the center of one voxel, as is the case on level 0, or between two voxels, as is the case on other levels. In the first case only one voxel needs to be adapted while both voxels have to be modified in the latter case. Due to the overlap between bricks, adaption has to be performed whenever a brick is adjacent to a coarser one. Consequently up to 26 bricks have to be considered in 3D.

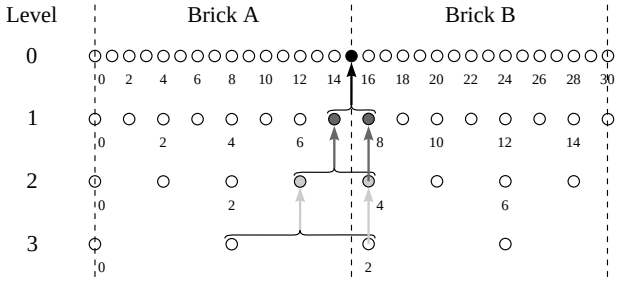

Figure 3: *Adjacent bricks with different level-of-detail need adaption to ensure consistent texture interpolation. Only the texture on the finer level has to be adapted by either interpolating or copying the voxels to the left and to the right of the brick boundary from the coarser level. On a brick at level 0 the black voxel has to be interpolated. Those voxels to be adapted on level 1 to level 2 are colored in dark and light grey respectively .*

3.3 Level-wise texture merging

The enlargement of textures in order to guarantee continuous level transitions leads to significant overhead in the texture memory that is used. Refer to the example shown in Figure 2. If we want to render the information represented by the first 31 voxels on level 0, we need two textures of size 16 for the finest level and two textures of size 16 on level 1 whereas only nine texture elements are necessary to store the information. Effectively there is no saving of texture memory when switching from level 0 to level 1. Considering higher levels leads to similar results, as only half of the voxels of every texture – plus one or two for overlap – contain non-redundant information.

This overhead can be minimized by merging the texture data of adjacent bricks into a single texture as shown in Figure 4. This figure demonstrates the one-dimensional analogue. For each level-of-detail the same texture size is used. In level 0 each brick has a texture of its own. As in any direction only half the voxels of a particular level are needed on the next coarser level, eight adjacent bricks can be represented by the same texture map, with appropriate texture coordinates. On higher levels, the number of bricks sharing the same texture map is multiplied by a factor of 8, which finally results in an octree-like hierarchy of texture maps.

Merging textures works best when the starting number of bricks in every direction matches a power of two. If more than 2^{l-1} bricks are created by the subdivision of the initial data set, where l is the number of levels used for rendering, additional bricks will be generated that build their own set of textures as described in subsection 3.1. If fewer bricks are generated, then the depth of the texture hierarchy has to be reduced by storing textures on the coarsest level individually. We utilize a texture manager object for administration of the texture hierarchy. This object creates the desired texture maps and assigns appropriate sub-textures to the bricks requesting texture memory.

3.4 Adaptive level-of-detail

The initial size of each brick has to be specified in advance before the multiscale volume representation is constructed. In our implementation the resolution of each brick is set up via a focus point oracle. According to the distance of the center of a brick to a user defined focus point, each brick determines its appropriate level-of-detail. In order to account for continuous level transitions, brick boundaries need to be adapted again whenever the hierarchy is changed by moving the focus point.

Furthermore, as long as no level transitions greater than one are

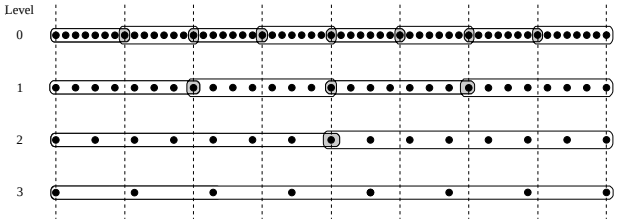

Figure 4: *A hierarchy is used to store the textures of the bricks more efficiently. The same size of textures is used on every level. On level 0 every brick contains a texture of its own. On level 1 one texture is shared by 8 adjacent bricks. On every consecutive level the number of bricks sharing a single texture is multiplied by a factor of 8. In order to take advantage of the hierarchy as effective as possible, the number of bricks created in each direction has to be a power of two.*

specified, our approach allows us to use arbitrary texture resolution for every brick. In particular, this can not be achieved by the technique proposed in [7], due to the strict correspondence between texture tiles and brick geometry.

One major drawback of the focus-point based construction of the texture hierarchy is that we do not consider the approximation error that is introduced if the data is sampled from a certain resolution level. As a matter of fact, if high frequencies are present in the data we obtain an approximation that differs significantly from the original signal. Consequently this scheme will impose artifacts between brick boundaries even if continuity is guaranteed.

To overcome this problem other criteria will have to be considered to determine the appropriate level-of-detail of each brick. An evaluation of different kinds of pyramidal representations for error controlled volume ray-casting has been presented in [4]. Apart from that, wavelet analysis as introduced for improved volume rendering in [9, 13] provides an effective tool for the calculation of the deviation between the original data and approximations at ever coarser resolution. This measure can then be used to select the appropriate level-of-detail automatically.

3.5 Internal texture format

According to the OpenGL specification textures can be stored in different internal formats in the graphics subsystem. In general, the texture maps used for volume rendering can either consist of color values or color indices, which are then transformed to color values using texture lookup tables.

The hierarchical decomposition as proposed does not imply any restrictions on the internal format to be used. Down-sampling and interpolation at brick boundaries can be performed in any case.

Although we are aware of the fact that in practical applications pre-shaded color values are often favored, interactive manipulation of visual quantities can only be achieved in color index mode. Thus, in the current implementation we decided to keep the obvious benefits of texture lookup tables resulting in the hierarchical decomposition of color indices, which are then transformed to color values during rendering.

4 Opacity corrected texture slicing

Once the original volume data set has been decomposed into the multiscale representation as described in section 3, each brick is rendered separately in back-to-front order at the appropriate level-of-detail. Therefore, we utilize the standard technique usually employed in volume rendering via 3D textures as outlined in [2, 3]. The basic idea is to re-sample a discrete 3D texture map on cutting planes parallel to the viewing plane by trilinear interpolation, and by compositing the resulting fragments in the frame buffer. Before we continue with a detailed description of our extensions for multiresolution 3D texture based volume rendering, let us point out again that the following explanations assume the color index mode.

Initially, the sample slice distance between consecutive cutting planes is chosen with respect to the width of texture elements. Let Δ_0 be this distance and assume that for a certain color index C_i the opacity entry α_0 in the color lookup table is computed as $1 - e^{-map[C_i]\Delta_0}$, where $map[C_i]$ defines the mapping from color indices to extinction coefficients. When the sample slice distance changes to Δ_k, opacity values stored in the color table have to be corrected with the new values computed as follows:

$$\begin{aligned} \alpha_k &= 1 - e^{-map[C_i]\Delta_k} \\ &= 1 - \left[e^{-map[C_i]\Delta_0}\right]^{\frac{\Delta_k}{\Delta_0}} \\ &= 1 - \left[1-\alpha_0\right]^{\frac{\Delta_k}{\Delta_0}} \end{aligned} \tag{1}$$

Since our level-of-detail representation already guarantees continuous transitions at brick boundaries, we can render the volume with an arbitrary sample slice distance while still achieving correct results even if bricks from different resolution levels are adjacent to each other.

However as previously noted, the goal of the proposed method is twofold: to minimize the amount of texture memory needed to render the data at the desired level-of-detail *and* to reduce the number of rasterization operations or texture lookups that have to be performed. Therefore, we adapt the sample slice distance within each brick with respect to the chosen resolution level.

In order to account for increasing width of texture elements on coarser resolution levels, a lookup table equipped with the corrected opacity values is stored for each possible resolution. Thus, the opacity values in each table account for the varying distance of slices to be rendered. Whenever a brick is rendered at a certain resolution the sample slice distance is set and the appropriate color table is used. Slices on level k are positioned in such a way that they are always in a distance $n \cdot 2^k \Delta_0$, $n \in Z$, from the viewing plane in order to guarantee correct alignment between slices in different bricks. In the following, whenever two bricks at level k and $k+1$ are adjacent to each other, we will call slices at positions $n \cdot 2^{k+1}\Delta_0$ the *even*-slices and slices at positions $(n \cdot 2^{k+1}+1)\Delta_0$ the *odd*-slices.

4.1 Opacity correction by polygon clipping

Even with the opacity correction integrated in our approach as described above, the projection of adjoining voxels might still result in erroneously drawn pixels at boundaries between different resolution levels. The region of error can be determined by projecting cutting planes along the slice boundaries. The errors are due to the following cases, demonstrated in Figure 5:

- **Case 1:** One slice of thickness Δ_{k+1} and one slice of thickness Δ_k are rendered but should only cover a width of Δ_{k+1}. On the left of Figure 5, going from back-to-front we render *even*-slice i and *odd*-slice $i+1$ on the finer level in region A. In region B we only render *even*-slice i on the coarser level but with correctly adapted opacity. In region C, however, *even*-slice i is rendered on the coarser level and *odd*-slice $i+1$ is rendered on the finer level before *even*-slice $i+2$ will be rendered.

- **Case 2:** Only one slice of thickness Δ_k is rendered but should cover a width of Δ_{k+1}. On the right of Figure 5, going from back-to-front we render *even*-slice i and *odd*-slice $i+1$ on the finer resolution level in region B. In region A we only render *even*-slice i on the coarser level but with correctly adapted opacity. In region C, however, only *even*-slice i is rendered on the finer level before *even*-slice $i+2$ will be rendered.

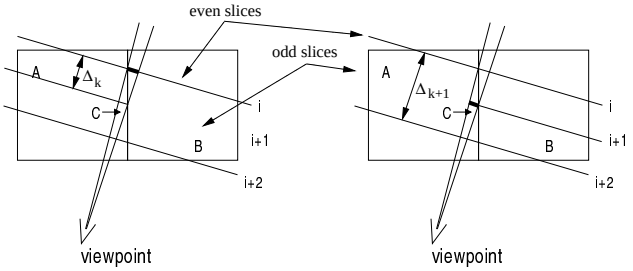

Figure 5: *Opacity correction at level transitions for orthographic and perspective views falls into two basic classes, which depend on whether we look through a brick at the coarser level to a brick at the finer level or vice versa.*

In general, however, incorrect opacities can be avoided by determining the erroneously rendered regions within cutting planes - the thick line segments in Figure 5 - and by rendering these parts with corrected opacity.

We proceed by recognizing that only bricks adjacent to at least one neighbor on a finer resolution level need to be adapted. In case 1, the *even*-slice i in the coarser brick has to be clipped with the perspective projection of the *odd*-slice $i+1$ that has already been reconstructed in the finer brick onto slice i. Finally, the clipped polygon in region C has to be rendered with the same opacity as issued on the finer level. In case 2, *odd*-slice $i+1$ has to be reconstructed in the coarser brick and clipped with the perspective projection of *even*-slice i in the finer brick onto the slice $i+1$. Finally, the clipped polygon included in region C has to be rendered with corrected opacity α_{k+1}. Note that this also works for orthographic projection.

The entire procedure can easily be generalized to the 3D case (see Figure 6). Here, for each pair of *even/odd*-slices in the coarser brick, we consecutively clip the *even*-slice i with all *odd*-slices $i+1$ in the same brick and neighboring bricks on the same or a finer level, and we clip the *odd*-slice $i+1$ only with *even*-slices i in neighboring bricks on a finer level. The opacity of clipping polygons is adapted according to the level of the neighboring brick. In this way, we completely avoid discontinuous transitions at brick boundaries as long as the difference between adjacent levels does not exceed one.

The described procedure still works if the previous *even*-slice or the next *odd*-slice are outside the current brick, and other special cases can only occur at the volume boundaries. At the boundaries it is possible that there are no other polygons with which the actual polygon can be clipped. In this case, we insert additional slices with the smallest sample distance in order to guarantee the correct opacity contribution.

As many clipping operations have to be performed, the implementation of the clipping algorithm is of particular relevance in order to keep the introduced overhead low. We modified a three-dimensional Sutherland-Hodgman [11] polygon clipping algorithm in such a way, that the clipped polygon as well as the area excluded from the original polygon is computed with little overhead. Thus the number of clip operations can be reduced significantly as both areas are needed in order to obtain correct opacities.

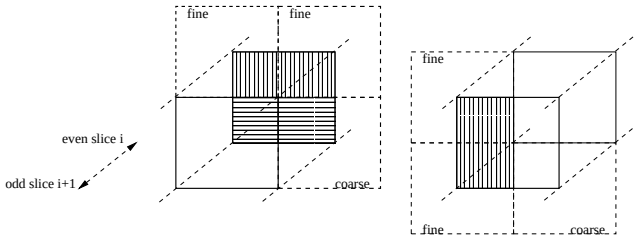

Figure 6: *The extension of opacity corrected clipping to 3D. Note that the brick under consideration is always located on the coarser resolution level. In the image, solid lines indicate clip polygon reconstructed in the current brick. Dashed lines indicate clip polygon reconstructed in neighboring bricks.*

5 Results and Analysis

In this section we provide results and we analyze the main modules of the proposed technique. All tests were run on a SGI Octane equipped with one R10000, 250 MHz processor, 4 MB texture memory and 256 MB main memory.

In Figure 7, results of the proposed level-of-detail rendering technique applied to a MRI-scan of size 256^3 are shown. All images were rendered using 8^3 bricks of size 32^3. An additional clipping plane was set up in order to demonstrate the multiresolution hierarchy more clearly. In the leftmost image all bricks were rendered with full resolution. In the middle image only the four upper slabs of bricks were rendered at full resolution. In the right image, only the two upper slabs of bricks were rendered with full resolution. The remaining slabs were consecutively rendered with decreasing resolution.

Although opacity was corrected as proposed, these images clearly show the potential drawback of a non topology preserving coarsification as introduced by our focus point based approach: the selection of the level-of-detail to be used in each brick exclusively relies on the focus point oracle but does not consider the approximation error that is introduced due to the coarsification. Thus, the topology of structures within the data set is not preserved *and* visual artifacts at brick boundaries may occur if the resampled approximation significantly differs from the original data. The same effect can be noticed in Figure 9, where the data set is rendered using four different levels of detail from front to back. Many of the relevant structures disappear due to the low pass filtering.

The hierarchical representation of the head data set, on the other hand, leads to a significant saving of texture memory. In the middle image only 64% of the original texture memory was used, while on the right, only 37% was used. The number of texture lookups was reduced, from 69% to 40%. This leads to improved rendering performance. Whereas every frame took about 2 seconds with the full resolution data set, the level-of-detail representation used for the rightmost image allows us to create a new image every 0.5 seconds.

Special attention has been paid to sorting the clipped polygons with regard to a minimal number of color table reloads. Otherwise the performance is decreased significantly: this is due to the fact that for opacity correction about 2500 clipping operations have to be performed on average, and every clipped region potentially requires a color table switch. In the worst case, when looking along a diagonal of the data set, up to 5500 polygons have to be clipped. The efficiency of our clipping algorithm can be seen from the fact that the overhead imposed by the clipping is less than 0.1 seconds.

We should point out the fact that the adaption of the finer textures, in order to guarantee continuity of the 3D scalar field (see section 3.2), only took about 0.2 seconds for the level-of-detail used in the rightmost image. Thus we can provide interactive change the

level-of-detail of the bricks.

In Figure 8 we demonstrate the artifacts that typically occur if opacity is not corrected with respect to different sample slice distances at level transitions. These artifacts manifest as light and dark bands along the boundary, which are exactly the erroneously rendered regions specified in section 4. In the the rightmost image, these artifacts are completely removed by the proposed technique. As can be seen, artifacts still show up at external faces, where we can clearly recognize the geometry of cutting planes used to resample the data. These are exactly the kind of artifacts that usually occur in volume rendering via 3D textures. However, in the presented examples, the sample slice distance is large at the boundaries. Thus, the artifacts are visually much more apparent than in the standard technique.

6 Conclusion

In this work we have emphasized a multiresolution approach for the rendering of large-scale volume data via 3D textures. The major contribution here is that we entirely avoid artifacts that occur due to incorrect texture interpolation and opacity correction at brick boundaries. In this respect, we have developed two beneficial extensions that guarantee continuous transitions between different levels of detail, and yield correct pixel opacities when using view dependent cutting planes.

We have demonstrated that the loss in the performance due to the polygon clipping is negligible when we factor in the gains from the considerable reduction of texture lookups and minimal texture memory use.

Furthermore, we developed a hierarchical texture representation that allows different bricks to be rendered at arbitrary resolution, with the only restriction being that the resolution of adjacent bricks can not differ by more than one level.

In the future, the integration and evaluation of different error measures that enable automatic selection of the appropriate level-of-detail are desirable. This would lead to a topology preserving coarsification.

Additional time should be spent in a detailed investigation of the pixel-wise error that is introduced due to the restrictions imposed by the available graphics hardware. Up to now, we did not consider the visual artifacts that may occur due to limited texture and frame buffer resolution. This is an open field of research.

7 Acknowledgements

This work was supported in part by the C-SAFE DOE ASCI Alliance Center and the DOE Advanced Visualization Technology Center (AVTC).

References

[1] Kurt Akeley. RealityEngine graphics. 27:109–116, August 1993.

[2] Brian Cabral, Nancy Cam, and Jim Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. *1994 Symposium on Volume Visualization*, pages 91–98, October 1994. ISBN 0-89791-741-3.

[3] 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, 1989.

[4] John Danskin and Pat Hanrahan. Fast algorithms for volume ray tracing. *1992 Workshop on Volume Visualization*, pages 91–98, 1992.

[5] George Eckel. *OpenGL Volumizer Programmer's Guide*. Silicon Graphics Computer Systems, Mountain View, CA, USA, 1998.

[6] Intense3D. *http://www.intense3D.com.*

[7] Eric LaMar, Bernd Hamann, and Kenneth Joy. Multiresolution techniques for interactive texture-based volume visualization. In *IEEE Visualization'99*. IEEE CS Press, October 1999.

[8] David Laur and Pat Hanrahan. Hierarchical splatting: A progressive refinement algorithm for volume rendering. *Computer Graphics*, 25(4):285–288, July 1991. ACM Siggraph '91 Conference Proceedings.

[9] Shigeru Muraki. Approximation and rendering of volume data using wavelet transforms. *Computer Graphics and Applications*, 13(4):50–56, 1993.

[10] Hanspeter Pfister, Jan Hardenbergh, Jim Knittel, Hugh Lauer, and Larry Seiler. The volumepro real-time ray-casting system. *Proceedings of SIGGRAPH 99*, pages 251–260, August 1999. ISBN 0-20148-560-5. Held in Los Angeles, California.

[11] B. Sutherland and G.W. Hodgman. Reentrant Polygon Clipping. *CACM 17(1)*, pages 32–42, January 1974.

[12] R. Westermann and T. Ertl. Efficiently using graphics hardware in volume rendering applications. In *Computer Graphics (SIGGRAPH 98 Proceedings)*, pages 291–294, 1998.

[13] Ruediger Westermann. A multiresolution framework for volume rendering. In *ACM Symposium on Volume Visualization'94*, October 1994.

[14] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL Programming Guide*. Addison-Wesley, Reading, MA, 1999.

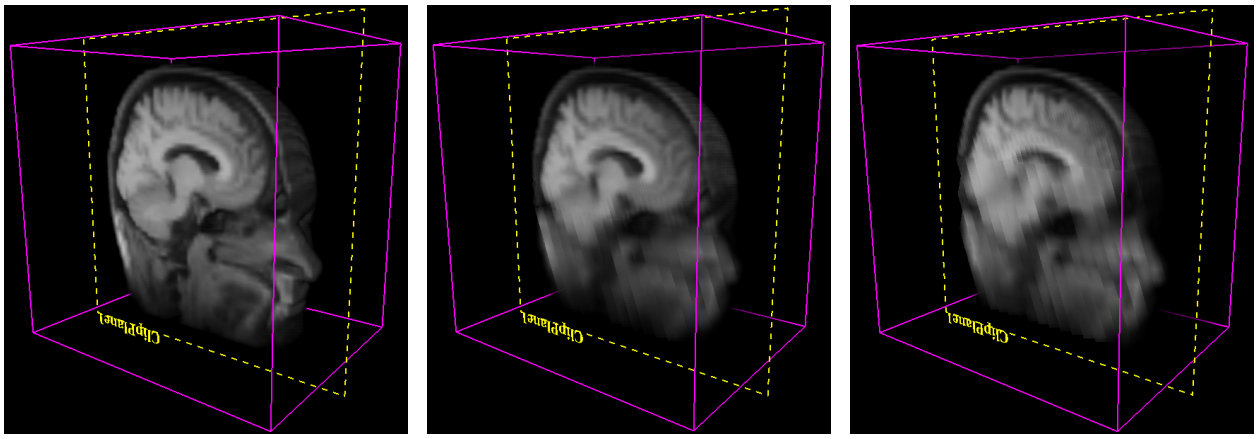

Figure 7: *The leftmost image shows the data set rendered with the original resolution. Then, the data was split into* 4^3 *bricks, each of size* 32^3*. In the middle/right image the upper two/one 4x4-slabs of bricks are rendered on the finest level-of-detail, continuously decreasing the resolution to the bottom.*

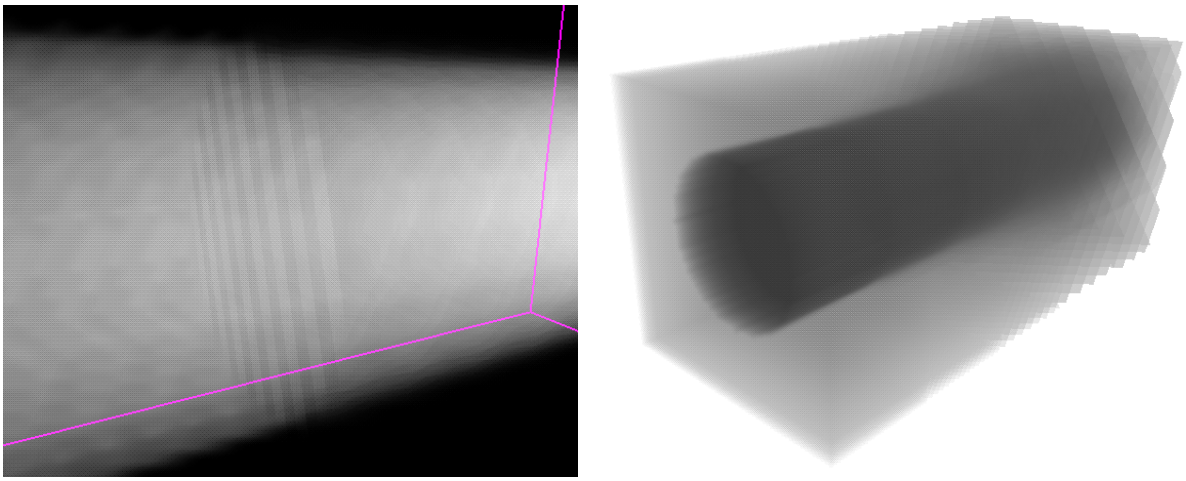

Figure 8: *In both images we show the same artificial data set rendered on four different resolution levels. On the left, severe artifacts at the brick boundaries occur because opacity is only corrected with respect to the sample slice distance. On the right, artifacts can be seen resulting from increasing sample slice distance at the volume boundaries.*

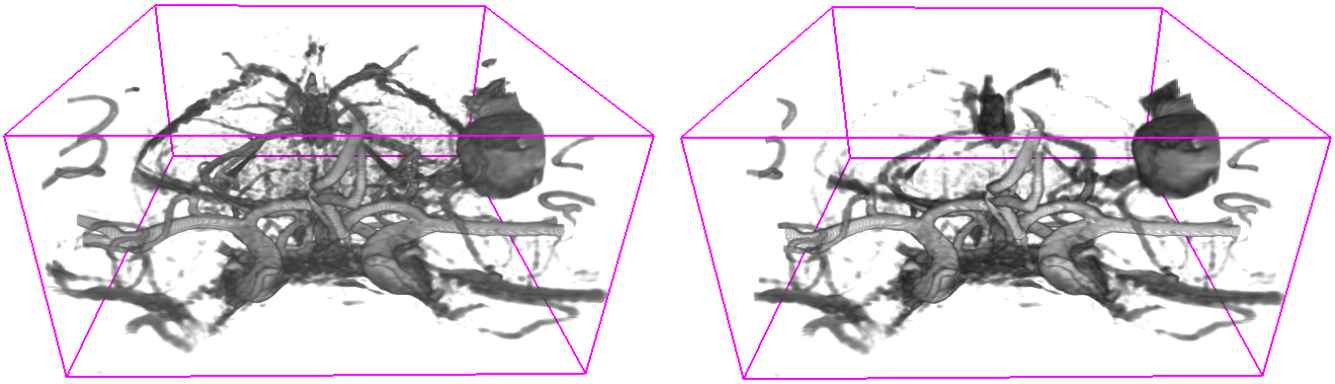

Figure 9: *A volume data set is rendered with full resolution via 3D textures. On the right, the adaptive representation using four levels of detail from front-to-back requires only 38% of the original texture memory, and it was rendered with only 41% of the originally needed number of texture lookups.*