Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

RALF KÄHLER, HANS-CHRISTIAN HEGE

Interactive Volume Rendering of Adaptive Mesh Refinement Data

ZIB-Report 01-30 (October 2001)

Interactive Volume Rendering of Adaptive Mesh Refinement Data

Ralf Kähler Hans-Christian Hege*

Abstract

Many phenomena in nature and engineering happen simultaneously on rather diverse spatial and temporal scales, i.e. exhibit a multi-scale character. Therefore various hierarchical data structures and numerical schemes have been devised to represent quantitatively such phenomena. A special numerical multilevel technique, associated with a particular hierarchical data structure, is so-called Adaptive Mesh Refinement (AMR). This scheme achieves locally very high spatial and temporal resolutions. Due to its popularity, many scientists are in need of interactive visualization tools for AMR data.

In this article we present a 3D texture-based volume rendering algorithm for AMR data, that directly utilizes the hierarchical structure. Thereby interactive rendering even for large data sets is achieved. In particular the problems of interpolation artifacts, opacity corrections, and texture memory limitations are addressed.

The algorithm's value in practice is demonstrated with simulation and image data.

Keywords: Scalar field visualization, multiresolution volume rendering, AMR hierarchies, 3D texture mapping.

1 Introduction

In numerical analysis hierarchical techniques for local refinement became more and more popular in the last years, because they lead to more reliable results and allow to simulate more complex phenomena. A special scheme is Adaptive Mesh Refinement (AMR), introduced by M. Berger in the 1980's [4]. In this approach a hierarchy of nested grids is generated, representing relevant regions of the computational domain on different levels of resolution. The numerical technique is applied in many domains like hydrodynamics [3], astrophysics [7] and meteorology [2].

Hence an increasing number of scientists is in need of appropriate interactive visualization techniques to interpret and analyze 3D AMR simulation data. They need tools for both, 2D analysis to quantitatively convey the information within single slices and a 3D representation to quickly grasp the overall structure. A popular interactive technique for interactive visualization of scalar data is texture based volume rendering, which leverages modern graphics hardware.

In this paper we present a 3D texture based algorithm for volume rendering of AMR data that achieves interactive frame rates also for large hierarchies. It directly exploits the hierarchical structure of the AMR data thereby accelerates rendering and also allows a level of detail representation. Interpolation artifacts are avoided by employing globally continuous interpolation. Economical use of limited texture memory is achieved by utilizing packing algorithms.

In Sect. 2 we give an overview about related work in the field of multiresolution volume visualization. Subsequently the AMR data structure is explained. In Sect. 4 we discuss the problem of interpolation artifacts at grid

*Email: {kaehler,hege}@zib.de

boundaries. The data structure utilized for volume rendering of AMR hierarchies and a texture packing scheme is described in Sections 5 and 6. In Sect. 7 the adaptive node selection strategies as well as opacity corrections. We discuss results of our algorithm applied to AMR data in Sect. 8, and finally conclude with areas of future research in Sect. 9.

2 Related Work

Volume rendering via 3D texture mapping hardware was introduced by Cullip and Neumann in 1989 [9]. In particular they compared axis- and viewpoint-aligned approaches. Cabral et al. [8] pointed out the correspondence between volume rendering integrals and Radon transformations; furthermore they demonstrated that 3D texture-based approaches allow interactive volume rendering and volume reconstruction.

A multi-resolution approach for volume rendering with 3D textures was described by LaMar et al. [12]. They employed an octree based subsampling scheme for uniform scalar datasets to represent regions of the data volume on different levels of resolution. Additionally they proposed different strategies for view-dependent node selection and introduced the use of spherical shells as proxy geometries, with the advantage of larger field of view, but the drawback of higher computational effort. Weiler et al. presented a similar approach, paying special attention to avoidance of interpolation artifacts at the boundaries of adjacent cells on different levels of resolution [21]. Their approach requires that adjacent regions differ by at most one level of resolution. They further improved the technique of opacity corrections to reduce visual artifacts that are caused by varying sampling distances of the texture slices. Boada et al. presented strategies for adaptive selection of octree nodes from the full pyramidal structure, utilizing data homogeneity and importance criteria [6].

Still only a small number of papers deal with rendering methods for AMR data. A back-to-front cell-sorting algorithm for AMR hierarchies was presented by Max and utilized for volume rendering as well as contour surface extraction [15]. In a preprocessing step he resamples cell-centered data to vertex-centered data to ensure smooth volume rendering and crack-free surfaces. Norman et al. describe their approach of resampling AMR data to uniform as well as unstructured grids, which allows them to apply standard rendering algorithms [16]. Ma presented a parallel volume renderer for AMR data generated by the PARAMESH [17] package, resampling cell-centered data to vertex-centered data in a preprocessing step [14].

Weber et al. proposed an approach for crack-free isosurface extraction of AMR data [19] and a software and hardware accelerated cell-projection algorithm for AMR data [18],[20]. The authors utilize different types of stitching cells to connect cells on different levels of resolution. This approach avoids resampling of cell-centered data, but restricts the applicability of their algorithm to AMR schemes that which guarantee that refined levels are surrounded by a layer of cells from the next coarser level.

Kähler et al. accelerated the volume rendering of large, sparse datasets by utilizing AMR data structures to extract non-transparent regions of the data volume [11].

We present a 3D texture based volume rendering algorithm, which exploits

the hierarchical nature of AMR data in the sense that the distance of texture slices is chosen with respect to the locally varying cell-sizes and rendering of subgrids can be omitted, if their parent cells have subpixel size. For this we propose a new space partitioning scheme that decomposes the volume into axis-aligned bricks on the same level of resolution. This is also utilized for view-consistent rendering. The procedure creates a small number of bricks, which accelerates texture based volume rendering, since intersection computation is done in software. We further avoid interpolation artifacts at grid transitions, by interpolating continuously even at boundaries between subgrids of arbitrarily different resolution.

3 The AMR Data Structure

In the AMR approach the whole computational domain is covered by a coarse grid, representing the root node of the hierarchical data structure. In regions where higher resolution is required, finer subgrids are created as child nodes of the root grid. Together they define a new level of the hierarchy, increasing the resolution of their parent grid by the so-called refinement factor. Fig. 1 shows a 2D example. This process repeats until all leaf grid cells satisfy certain error criteria, which depend on the particular numerical simulation.

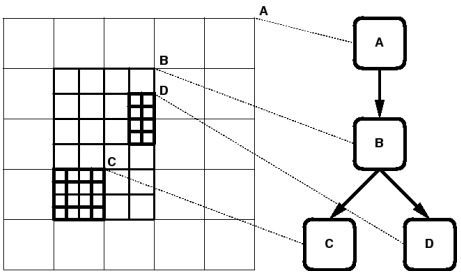


Figure 1: 2D example of an AMR grid hierarchy with a refinement factor of 2. Root grid A has one subgrid B, which again has two children (C, D).

The data values are normally stored at the grids nodes (vertex-centered) or at the centers of the cells (cell-centered). For simplification purposes the AMR schemes usually fulfill the following restrictions:

- The subgrids are axis-aligned, structured rectilinear meshes, consisting of hexahedral cells with constant edge lengths.
- Subgrids are completely contained within their parent grids.
- Subgrids begin and end on parent cell boundaries, which implies that parent grid cells are either completely refined or completely unrefined.

Notice that the resolution levels of adjacent cells may differ by more than 1. This has to be taken into account during interpolation to avoid artifacts due to by discontinuities at grid boundaries.

4 Interpolation

Artifact-free volume rendering of AMR data requires globally continuous interpolation of the discrete data, c.f. Fig. 2.

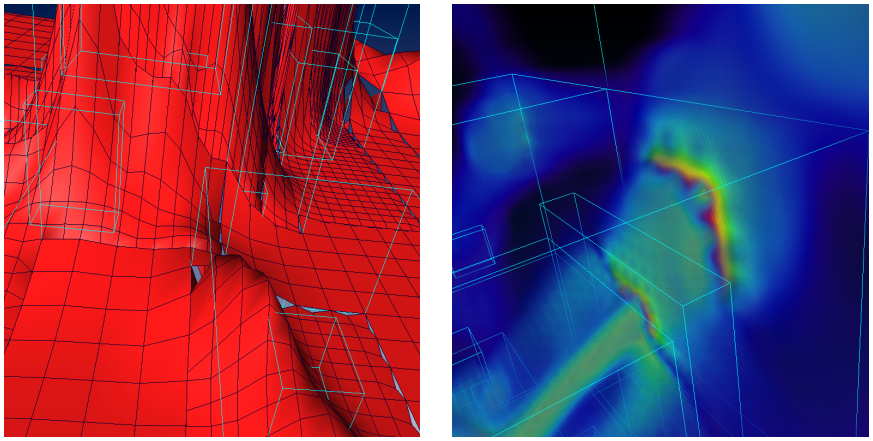


Figure 2: Left: Cracks in a heightfield surface caused by discontinuous transitions of the interpolation function at grid boundaries. Right: In the volume rendered image interpolation artifacts are visible as discontinuous change of color.

In this section we address the problem for vertex- as well as cell-centered data. Let us first consider the case of vertex-centered AMR data as indicated in Fig. 3. Within hexahedral cells typically trilinear interpolation is applied.

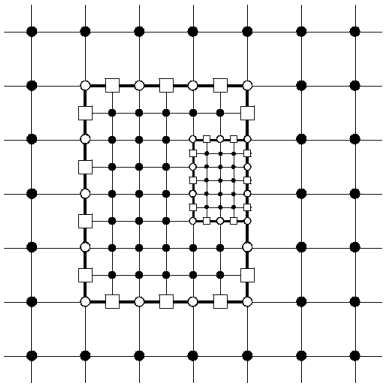


Figure 3: A 2D example of an AMR grid with one subgrid: Data values on common boundary nodes (white circles) are equal, hanging nodes (white rectangles) are obtained by linear interpolation (bilinear in the 3D case) between adjacent common nodes.

Hence it has to be ensured that the interpolated scalar function is continuous also at the boundaries of adjacent cells. For adjacent cells of the same resolution

level this is automatically fulfilled, since both trilinear interpolants degenerate to the same bilinear interpolant on the common face.

Continuity at boundaries of grid cells with different resolution requires the following: First, the data values on the boundary nodes common to a subgrid and its parent grid have to be equal. This is ensured in numerical AMR schemes by the so-called projection step, which updates the data values at coarser grid nodes by the more accurate values of the subgrids. Second, the values at grid boundary nodes without corresponding coarse nodes on their parent grid (hanging or dangling nodes) have to be obtained by bilinear interpolation between the adjacent nodes. This ensures, that for both sides the same values are computed on the whole interface.

We now consider the case of cell-centered AMR data. Nearest neighbour (i.e. constant) interpolation schemes available in graphics API's like OpenGL can be used to render the data directly and give scientists the possibility to examine the unmodified results of their simulations. But of course this scheme has the drawback that the resulting images usually show quite granular structures, see Fig. 4.

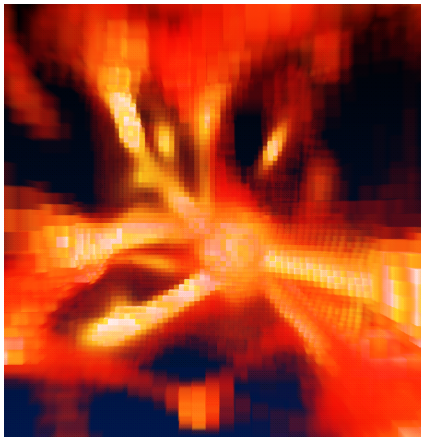


Figure 4: Nearest neighbour, i.e. constant interpolation tends to create granular and blurry images.

So one further needs an approach that obtains better image quality. Defining the dual grid of each AMR grid as a 3D texture and using trilinear interpolation would result in gaps between the grids. Inserting different types of stitching cells to fill the gaps, as worked out in [18], is not applicable for volume rendering via 3D textures, since the texels are supposed to be equidistant. Furthermore this approach would restrict our algorithm to AMR schemes which place a layer of width of one or more cells between a parent grid and the boundary of the refined level [20], [7].

So we decided to resample the cell-centered AMR data to achieve the vertex-centered case described above. Inner nodes of the vertex-centered grid are obtained by trilinear interpolation of the 8 surrounding cell-centered data values, like proposed in [15] and [14], see Fig. 5.

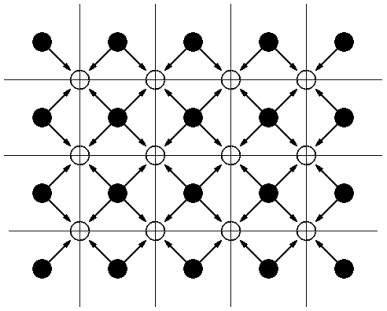


Figure 5: Resampling of cell-centered data. The values of inner vertex-centered nodes are obtained by trilinear interpolation of the 4 (8 in the 3D case) adjacent cell-centered data values.

For the boundary nodes we distinguish the following cases:

- In case the boundary node is surrounded by cells of AMR grids with the same level of resolution, the value is obtained by trilinear interpolation, as in the case of an inner node.
- If the node on the vertex-centered grid has a corresponding coarse cell node and is adjacent to a coarse cell, the value is obtained by projection from the parent grid.
- If the node is a hanging node, the value is obtained by bilinear interpolation between the surrounding boundary nodes, just like in the vertex-centered case.

Notice that by projecting the values from grid nodes to subgrid nodes, also the more accurate values of the finer grids are taken into account, since the coarse cell values that are further refined are averaged from the subgrid cells in the numerical scheme, as mentioned above. Alternatively one could average between adjacent coarse and fine cells like proposed in [14] and project the values to the coarse nodes. But this could introduce artifacts on the coarse grid, which become visible if subgrids are omitted during rendering to increase the performance, see Sect. 7.

The AMR hierarchy is resampled in a top-to-bottom traversal, starting at the root grid. The vertex-centered boundary values of the root grid have to be obtained by extrapolation.

We end up with a vertex-centered hierarchy, that ensures continuous trilinear function interpolation, even if adjacent cells differ by more than one level of resolution.

5 Texture-Brick Hierarchy

A possible approach for volume rendering an AMR hierarchy via 3D textures is to process each slice separately, followed by a blending step in the frame buffer (slice-by-slice approach). The intersection points of each grid with the slices would have to be computed and the associated polygons would have to

be transferred to the texture hardware. First the polygons that result from intersection with the finest grids would have to be interpolated and rendered. In order to prevent these regions from being painted over by the following slice parts one could utilize the stencil-buffer available in the graphics API. In the next steps areas defined by the polygons of the next coarser levels would be rendered, with the stencil buffer being updated appropriately. Then the next slice would be processed and the slices would be blended back-to-front in the frame buffer.

However, this approach has some drawbacks: Since volume rendering is fill-rate limited, it is disadvantageous to render the same area in screen space several times, even if the stencil buffer prevents the frame buffer from being painted over. If the total size of textures needed to represent the AMR hierarchy exceeds the amount of available texture memory, texture swapping will take place. In the approach described above swapping occurs several times for each region corresponding to a texture, or part of a texture.

Therefore we decided to render the hierarchy brick-by-brick. Multiple rendering of regions that are covered with cells of different resolution is avoided by decomposing the grids into axis-aligned regions that consist either of cells that are refined by subgrids, or of cells which are not further refined. In the rendering step these regions are visited back-to-front, and each of them is processed separately.

Weber et al. also proposed a scheme for decomposing the domain into regions of refined and unrefined cells used for a hardware accelerated cell-projection algorithm [18]. This approach splits subgrids and creates a large number of regions, which is appropriate for cell-projection. However, for texture based volume rendering this would cause a lot of computation for intersection and texture coordinates, and hence would slow down the performance.

We propose a decomposition heuristic that creates few regions and avoids the splitting of subgrids in most cases. To see how this decomposition works, consider a 2D example of one root grid that has 3 subgrids on the first level of refinement, as shown in Fig. 6.

The first step is to split the region covered by the root grid so that one subgrid falls into one subvolume and the other two subgrids fall into in the other subvolume (split 1 in Fig. 6). In general, several splits could be possible. In that case the one which subdivides the grids most balanced is taken. There might also exist configurations for which it not possible to find such a partition position, see for example Fig. 7.

In these cases one or more grids and possibly also their subgrids have to be split up. Notice that AMR schemes usually utilize clustering algorithms like the one presented by M. Berger [5], which create subgrids by binary space partitioning approaches. This ensures that always at least one partition position exists, that does not intersect any of the subgrids. In order to simplify the discussion, in the following we assume that such a clustering algorithm was used in the AMR scheme, though the presented algorithm does apply to the general case, too.

In the 2D example the partitioning is continued on the subvolume that contains the two subgrids (split 2 in Fig. 6). Now every subvolume contains exactly one subgrid. Each subvolume is further subdivided as indicated in Fig. 8, resulting in up to 6 new subvolumes in the 3D case.

There exist configurations of subgrids, where the partition scheme mentioned

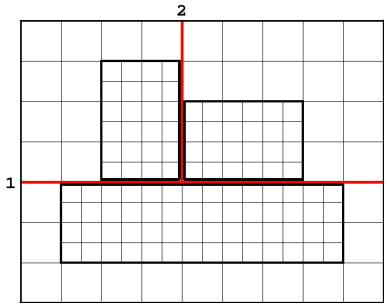


Figure 6: In order to subdivide the volume into axis-aligned, non-overlapping regions containing only cells of the same refinement level, two split axis are determined.

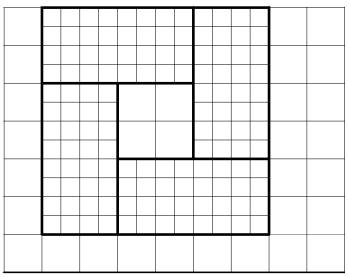


Figure 7: Example of a subgrid configuration, that can not be partitioned without splitting grids.

above, produces unnecessarily many bricks, see Fig. 9. In order to reduce the number of bricks in these configurations, we enclose the subgrids with a minimal bounding box, and first decompose the outer region, c.f. Fig. 10. Notice that this never increases the number of created bricks. For our datasets, this step reduced the total number of leaf bricks by up to 10 %.

Finally we end up with a partition of the root grid into subregions consisting of cells that are not covered by subgrid cells, and subregions that are totally refined by subgrid cells. If the subgrid is a leaf grid of the AMR hierarchy, the partitioning process is stopped, otherwise it is repeated on the region defined by the subgrid and its children.

This space decomposition is stored in a kD-tree data structure. For each grid a separate 3D texture is allocated (or inserted into a bigger texture in the case of texture packing, see Sect. 6) and a data structure node that can store texture brick and space partition information is allocated. The node associated with a grid stores a reference to its texture object, positional information, and an offset into the texture coordinates. If the grid is not a leaf grid and thus a space partition as described above is carried out, also information about the partition axis and references to the two subnodes that are associated with the first two subvolumes are stored.

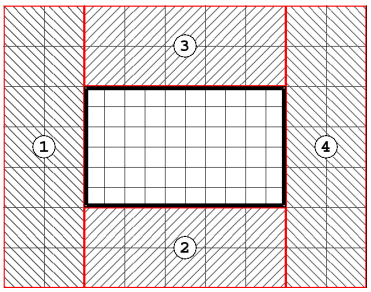


Figure 8: If a subregion of the parent grid contains just one subgrid, it is finally partitioned in up to 4 subregions (up to 6 in the 3D case).

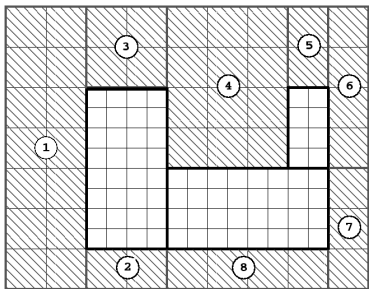


Figure 9: Subgrid configuration where the unmodified decomposition leads to an unnecessarily high number of bricks.

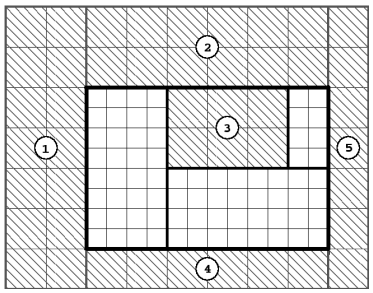


Figure 10: By first enclosing the subgrids with a minimal bounding box, and decomposing the outer region, the number of created bricks is reduced.

In a next step the subnode information is determined. In case a subregion contains only unrefined grid cells just the texture brick information is stored. It contains a reference to the grid's 3D texture object, the bricks' bounding box position and texture offset information of the subregion. This node becomes a leaf node. If the subregion covers both, refined and unrefined grid cells, then only information about the next partition axis and pointers to the next two subregions are stored. These nodes are used to traverse the hierarchy back-to-front in the rendering step. In the last case the subregion contains only grid cells that are further refined by a subgrid. A reference to that AMR subgrid texture object is stored and if the grid itself is further refined the process continues, otherwise this node becomes a leaf node.

6 Texture Packing

The graphics hardware assumes the dimensions of 3D-textures to be equal to a power of two. This could be achieved by extending the data subvolume of each leaf grid of the AMR hierarchy to the next bigger power of two, for example by clamping the boundary texels and restricting the generated texture coordinates to the unextended area. Regarding the potentially large number of textures to deal with, this often results in an enormous amount of unused texture memory. This holds especially in the case where cell-centered data is resampled onto vertex-centered data. Often the grids have dimensions equal to a power of two, and thus the corresponding vertex-centered grids contain a disadvantageous number of $(2^n + 1)$ data samples.

Therefore we reduce the texture memory requirements by utilizing a packing algorithm that merges separate textures into one bigger texture. We only sketch the algorithm in the following. For more details refer to [11].

We adopt a level-by-level layer-by-layer packing scheme. It is a three-dimensional version of the next-fit-decreasing-height (NFDH) algorithm [10], [13]. In the first step the boxes are inserted into a list in the order of decreasing height. The packing algorithm can be imagined as starting at the lower left-hand corner of the container and inserting the boxes from left to right until the right border is reached. Then a new row is opened, with a depth coordinate given by the largest depth of the already inserted boxes. This procedure is repeated until the lowest layer of the container is filled. Then a new layer is opened and this process continues until all boxes are inserted.

If the resulting packed texture is still to big to fit into texture memory, one could try to pack them into several smaller textures, which have to be swapped during the rendering. Since swapping is usually faster with smaller bricks, we do not apply the packing approach in this case, but define a separate texture for each subgrid.

7 Rendering

As mentioned in Sect. 5, the brick structure is utilized for traversing the separate bricks back-to-front, starting at the root node. If a node is processed that stores texture brick information, two cases have to be distinguished:

- The node is a leaf node, indicating that the covered cells are not further refined and thus have to be rendered.
- The node is not a leaf node, i.e. it represents an grid that is further refined. In this case it has to be checked if it is sufficient to render this region on this level of detail, or if the subnodes have to be visited, since higher visual accuracy is required.

In the second case the following strategies are applied, in order to check whether a brick is selected for rendering.

It is sufficient to render a region with the level of detail of the associated grid, if the projection of the grid cells in screen space have subpixel size. To check this, we determine the extent in screen space of a ball that is centered at the grid's bounding box corner nearest to the view point and whose diameter equals the grid cells biggest diagonal. If the projected size is smaller than the size of a pixel, all cells of this grid have subpixel size and the brick is chosen to be rendered, omitting its subnodes.

Further a maximal level at which the hierarchy traversal is stopped, can be specified. This is used to guarantee a desired lower bound of the frame rate.

7.1 Opacity Corrections and Adaptive Brick Selection

If a brick is selected it is rendered utilizing the standard approach for volume rendering with 3D textures as for example proposed in [9], [8]. The 3D texture is sampled on slices perpendicular to the viewing direction and blended in the frame buffer. We use one channel textures and the OpenGL colortable extension.

In order to take advantage of the multi-resolution structure of the AMR data for fast rendering, the sample distance of the slices is set with respect to the resolution level of the texture brick: For bricks on the root level a distance d_0 is chosen, bricks on level l are rendered with slice distances $d_l = \frac{d_0}{r^l}$, where r is the relative refinement factor between two consecutive levels of the AMR hierarchy. Slices are aligned like indicated in Fig. 11. Since the function interpolation con-

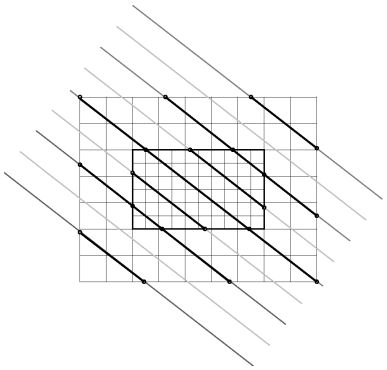


Figure 11: The sample distance of slices varies locally and depends on the bricks' cell size. Correct alignment is ensured by choosing the slice distances as integer multiples of a minimal distance.

tinuity is ensured (see Sect. 4) adjacent parts of the slices show no interpolation

artifacts. Nevertheless, bricks from different levels are rendered with varying sample distances, so one has to perform opacity corrections. Similar to the approach discussed in [21], we accomplish this by defining a separate colormap for each level l of the hierarchy with opacity values $\alpha_i[l]$, $l = 0, 1, ...l_{max}$, where the index i numbers the colormap entries for each level. Assuming the following relationship between the opacity entry $\alpha_i[0]$ and the associated extinction values τ_i for the root level of the hierarchy

$$\alpha_i[0] \quad = \quad 1 - e^{-\tau_i d_0}.$$

We obtain the opacity entries for a higher resolved level l as

$$\begin{aligned} \alpha_i[l] &= 1 - e^{-\tau_i d_l} \\ &= 1 - (e^{-\tau_i d_0})^{\frac{d_l}{d_0}} \\ &= 1 - (1 - \alpha_i[0])^{\frac{d_l}{d_0}} \\ &= 1 - (1 - \alpha_i[0])^{\frac{1}{r^l}}, \end{aligned}$$

where r is the relative refinement factor. Before a subregion of the hierarchy is rendered the appropriate colortable is activated. Computing the intersection points of the slices and the bricks' bounding boxes is done in software. We speed up this procedure by first determining for each brick the interval of slices intersecting it. This is accomplished by projecting the bounding box corners of the brick on the planes normal direction as shown in Fig. 12. Only for this subset the intersections need to be computed.

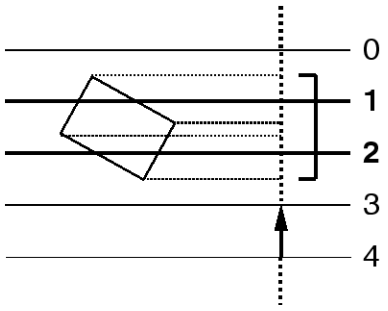


Figure 12: The interval of slices that intersects a brick is precomputed by projecting the grids' bounding boxes onto the slice normal direction.

8 Results and Discussion

The algorithm has been implemented in Amira [1], an object-oriented, extendable 3D data visualization system developed at ZIB. We applied the algorithm to several AMR datasets. The performance was tested on a SGI Onyx2 InfiniteReality2 with two RM7 raster managers with 64 MB texture memory. The runs were performed on a single 195 MHz MIPS R10k processor.

Since texture based volume rendering is fill rate limited, the frame rates depend on the size of the viewer window, the number of slices, and the area in screen space covered by the data volume (and thus on the actual position of the viewpoint). For all images examples the size of the viewport was 764×793 pixels.

Dataset I, resulting from an AMR galaxy cluster simulation, consists of 91 grids on 8 levels of refinement. The (resampled) root level contained 33^3 data samples and was rendered with 120 slices, the more refined grids with respectively more. Dataset II represents a hierarchy consisting of 359 grids on 4 levels of refinement. The root level contained $95 \times 63 \times 14$ data samples and 200 slices were chosen for the root level. This AMR hierarchy was generated from an uniform $749 \times 495 \times 100$ sized confocal microscopy dataset utilizing an opacity based importance criterion. Regions with associated opacity values below a certain threshold are represented with coarser resolution. For more details about this algorithm refer to [11]. Dataset III is an AMR hierarchy resulting from a cosmological AMR simulation that consists of 813 grids on 10 levels of refinement. The (resampled) root grid contained 129^3 data samples and was rendered with 250 slices. Table 1 lists the number of leaf bricks created,

	bricks	**prepr.**	**ratio**	**texmem**
Dataset I	345	0.2 s	45%	1 MB
Dataset II	970	1.2 s	43%	16 MB
Dataset III	3370	5.8 s	46%	16 MB

Table 1: This table lists the number of created leaf bricks, the preprocessing time for allocating and packing the textures as well as resampling, the achieved texture memory reduction and the resulting size of the packed texture.

the preprocessing time for allocating the brick's hierarchy and texture packing as well as resampling in case of cell-centered data, the percentage of texture memory reduction achieved by texture packing and the size of the resulting texture.

An average of 3 to 4 leaf bricks per subgrid was created, independent of the depth and total number of grids of the hierarchy. The average texture memory reduction achieved by packing was about 45 %.

Resulting renderings are presented in Figs. 13 to 15. They show the root grid (a), the full hierarchy rendered with all bricks (b), a close-up view of the refined part of the hierarchy (c) and the associated bounding boxes of the subgrids (d).

Looking at the close-up views in Figs. 13 to 15 one notices regions with rendering artifacts caused by adaptive slice distances. They arise at boundary regions of adjacent grids with different resolution, since edges of slice parts that are not present on the coarser grids become visible. The effect gets less visible if the hierarchy is rendered with higher density of slices, which decreases the frame rates. So one has to balance frame rates against image quality. Table 2 displays the associated frame rates for the root level data, the full hierarchy and the close-up view on the refined part for the viewer positions chosen in Figs. 13 to 15. The last entry represents the frame rate achieved when rendering the full hierarchy in the mode described in Sect. 7, i.e. where bricks are omitted if their parent grids have subpixel size.

	root	full	close-up	full adap.
Dataset I	10.4	6.7	2.0	7.2
Dataset II	10.1	3.2	2.0	3.2
Dataset III	6.5	1.4	1.1	2.4

Table 2: The table shows the frame rates (fps) for rendering the root level data, the full hierarchy, the close-up view, and the full hierarchy in the mode where grids are omitted whose parents cells have subpixel size.

For all datasets interactive frame rates were achieved. The frame rates were minimal for the close-up views, since the covered screen space is maximal for these view points. As the performance figures for dataset III show, omitting subgrids with subpixel-sized parents can result in significant performance gains. In general the effect is more pronounced for deep hierarchies with a large number of subgrids on the more refined levels.

9 Conclusions and Future Work

We presented a hardware accelerated volume rendering approach for adaptive mesh refinement data utilizing 3D textures. Since current texture hardware requires axis-aligned texture bricks which contain cell on the same resolution level, some preprocessing is necessary. For this we proposed a new partitioning heuristics which creates a small number of such bricks – a prerequisite for interactive rendering. The heuristics avoids splitting of subgrids, if the grid hierarchy was created by a BSP algorithm like the widespread clustering algorithm of Berger [5]. During rendering, branches of the AMR tree are pruned, based on a projection criterion ensuring that the rendering results are not affected. Artifacts at grid boundaries due to discontinuities are avoided by globally continuous interpolation which does not impose restrictions on the level differences at grid boundaries. The amount of texture memory is reduced by employing a packing scheme.

Applying the rendering algorithm to several AMR datasets, we observed interactive frame rates even for large hierarchies.

As future work we will examine the use of larger colortables for more precise opacity correction, specifically for very deep hierarchies.

It would be interesting to investigate whether it is possible in spite of the potentially large number of bricks for deep hierarchies, to apply the additional opacity corrections at grid boundaries proposed by Weiler et al. [21] without significantly decreasing the rendering performance.

Furthermore the application of data homogeneity and importance based criteria as proposed by Boada et al. in [6] could be utilized for pruning branches of AMR hierarchies.

For camera positions inside the volume, e.g. in immersive environments, also the use of spherical shells as proxy geometries, as introduced by LaMar et al. [12], could be investigated.

10 Acknowledgments

We thank Detlev Stalling for helpful discussions and proof reading. The cosmological simulation datasets were provided by Stuart Levy, Mike Norman and Brian W. O'Shea, (National Center for Supercomputing Applications (NCSA), Urbana, Illinois). The confocal bee brain dataset was provided by Robert Brandt (research group Randolf Menzel, Freie Universität Berlin).

This work was supported in part by the Max-Planck-Institut für Gravitationsphysik (Albert-Einstein-Institut) in Potsdam/Germany.

References

[1] *Amira User's Guide and Reference Manual* as well as *Amira Programmer's Guide*. Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB) and Indeed - Visual Concepts GmbH, Berlin, http://www.amiravis.com, 2001.

[2] A.S. Almgren, J.B. Bell, P. Colella, L.H. Howell, and M. Welcome. A high-resolution adaptive projection method for regional atmospheric modeling. In *Proceedings of the NGEMCOM Conference sponsored by the U.S. EPA, Bay City, MI.*, 1995.

[3] M. J. Berger and P. Collela. Local adaptive mesh refinement for shock hydrodynamics. *J. Computational Physics*, 82(1):64–84, 1989.

[4] M. J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial equations. *Journal of Computational Physics*, 53:484–512, 1984.

[5] M. J. Berger and I. Rigoutsos. An algorithm for point clustering and grid generation. *IEEE Transactions on Systems, Man and Cybernetics*, 21(5), 1991.

[6] I. Boada, I. Navazo, and R. Scopigno. Multiresolution volume visualization with a texture-based octree. *The Visual Computer*, 17(5):185–197, 2001.

[7] G. L. Bryan. Fluids in the universe: Adaptive mesh refinement in cosmology. *Computing in Science and Engineering*, 1(2):46–53, 1999.

[8] B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In Arie Kaufman and Wolfgang Krueger, editors, *1994 Symposium on Volume Visualization*, pages 91–98, 1994.

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

[10] D. S. Johnson E.G. Coffman, JR., M.R.Garey and R. E. Tarjan. Performance bounds for level-oriented two dimensional packing algorithms. *SIAM Journal on Computing*, 9:808–826, 1980.

[11] R. Kähler, M. Simon, and H. C. Hege. Fast volume rendering of sparse datasets using adaptive mesh refinement. *ZIB-Report 01-25, July 2001*, submitted to IEEE Trans. on Visualization and Computer Graphics.

[12] E. C. LaMar, B. Hamann, and K. I. Joy. Multiresolution techniques for interactive texture-based volume visualization. In D. Ebert, M. Gross, and B. Hamann, editors, *IEEE Visualization '99*, pages 355–362, San Francisco, 1999. IEEE.

[13] K. Li and K.-H. Cheng. On three-dimensional packing. *SIAM Journal on Computing*, 19(5):847–867, 1990.

[14] K.-L. Ma. Parallel rendering of 3D AMR data on the sgi/cray T3E. *Proc. 7th Symposium on Frontiers of Massively Parallel Computation*, pages 138–145, 1999.

[15] N. L. Max. Sorting for polyhedron composition. *In H. Hagen, H.Mueller, G. M. Nielson: Focus on Scientific Visualization, Springer Verlag*, pages 259–268, 1993.

[16] M. Norman, J. Shalf, S. Levy, and G. Daues. Diving deep: Data-management and visualization strategies for adaptive mesh renement simulations. *Computing in Science and Engineering*, 1(4):22–32, 1999.

[17] PARAMESH. URL: http://webserv.gsfc.nasa.gov/rib/repositories/inhouse_gsfc/paramesh.html. *NASA Goddard Space Flight Center*, 1998.

[18] G. H. Weber, H. Hagen, B. Hamann, K. I. Joy, T. J. Ligocki, K.-L. Ma, and J. Shalf. Visualization of adaptive mesh refinement data. In *Proceedings IS&T/SPIE Electronic Imaging 2001*, 2001.

[19] G.H. Weber, O. Kreylos, T.J. Ligocki, J.M. Shalf, H. Hagen, B. Hamann, and K.I. Joy. Extraction of crack-free isosurfaces from adaptive mesh refinement data. In *Data Visualization 2001 (Proceedings of VisSym '01)*, pages 25–34, 2001.

[20] G.H. Weber, O. Kreylos, T.J. Ligocki, J.M. Shalf, H. Hagen, B. Hamann, and K.I. Joy. High-quality volume rendering of adaptive mesh refinement data. In *to appear in: Proceedings of Vision, Modeling, and Visualization 2001*, 2001.

[21] M. Weiler, R. Westermann, C. Hansen, K. Zimmerman, and T. Ertl. Level-of-detail volume rendering via 3D textures. In *IEEE Volume Visualization and Graphics Symposium 2000*, pages 7–13, 1994.

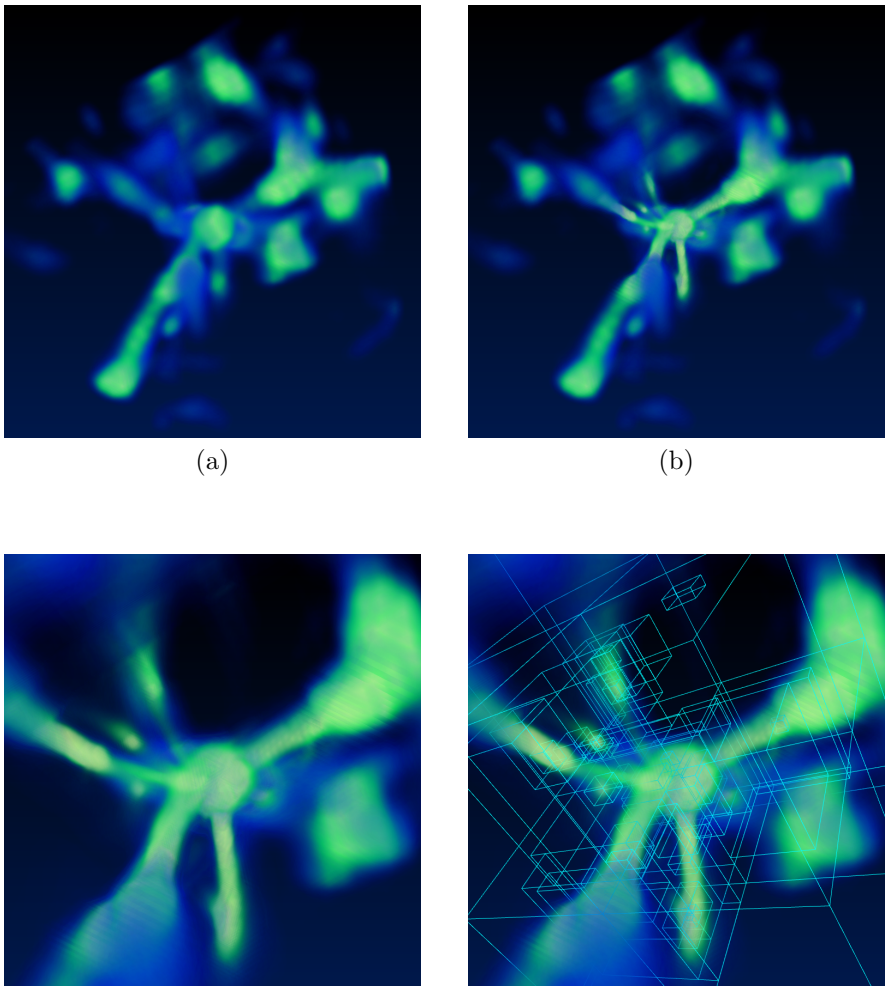
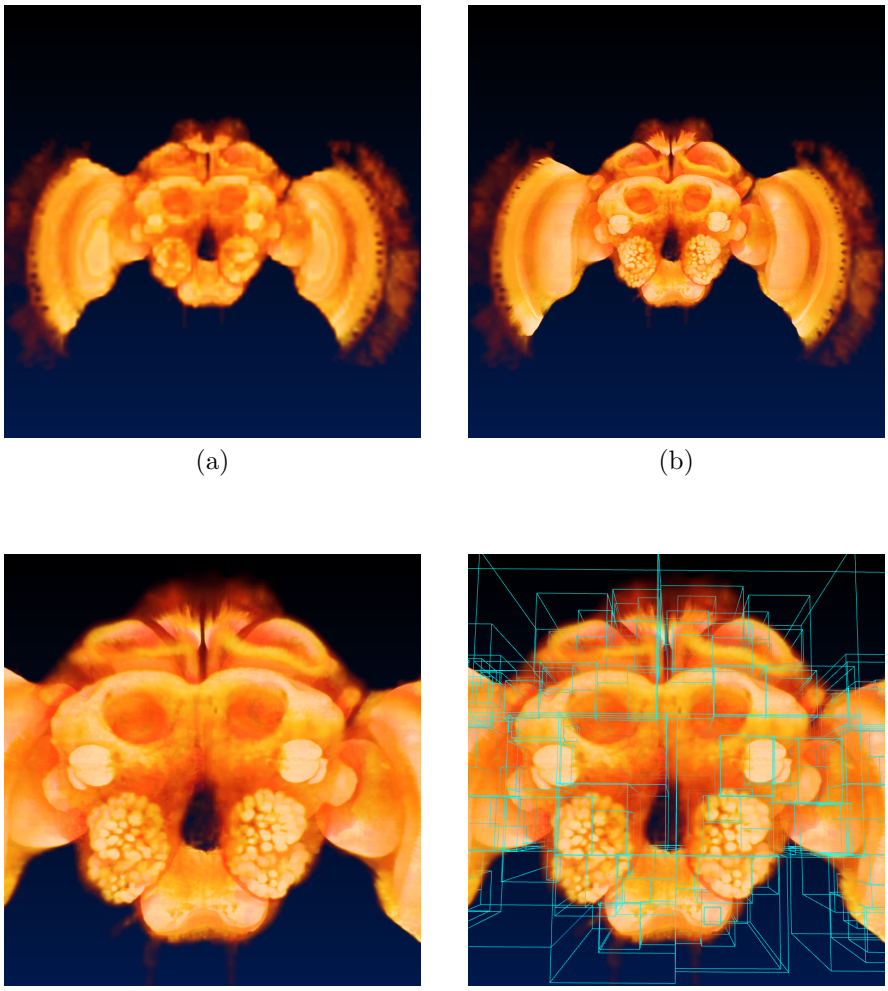


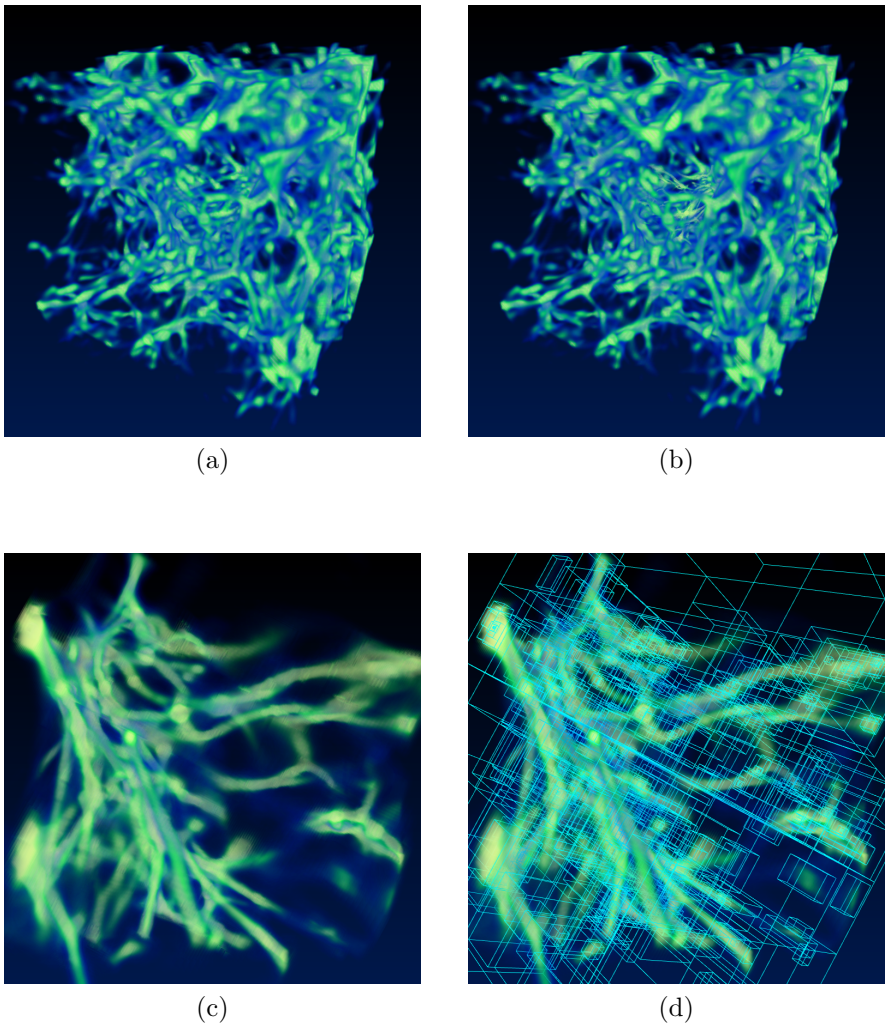
Figure 13: Dataset I, the result of a galaxy cluster AMR simulation, consists of 91 subgrids on 8 levels of refinement. (a) root level, (b) full hierarchy, (c) close-up view of refined inner region, (d) associated bounding boxes.



(a) (b)

(c) (d)

Figure 14: Dataset II, a AMR tree generated from a confocal microscopy image stack of a bee's brain, consists of 359 subgrids on 4 levels of refinement. (a) – (d) as in Fig. 13.



(a) (b)

(c) (d)

Figure 15: Dataset III, the result of a cosmological AMR simulation, consists of 813 subgrids on 10 levels of refinement. (a) – (d) as in Fig. 13. In image (c) root level bricks are not rendered in order to depict more clearly the refined regions of the data volume.