The Visual Computer

Texture-based volume rendering of adaptive mesh refinement data

Ralf Kähler,
Hans-Christian Hege

Konrad-Zuse-Zentrum für Informationstechnik
Berlin, Takustraße 7, 14195 Berlin-Dahlem, Germany
E-mail: {kaehler,hege}@zib.de

Published online: November 6, 2002
© Springer-Verlag 2002

Many phenomena in nature and engineering happen simultaneously on rather diverse spatial and temporal scales. In other words, they exhibit a multi-scale character. A special numerical multilevel technique associated with a particular hierarchical data structure is 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 fast rendering performance is achieved even for high-resolution data sets. To avoid multiple rendering of regions that are covered by grids of different levels of resolution, we propose a space partitioning scheme to decompose the volume into axis-aligned regions of equal-sized cells. Furthermore the problems of interpolation artifacts, opacity corrections, and texture memory limitations are addressed.

Key words: Scalar field visualization – Multi-resolution volume rendering – AMR data – 3D texture mapping.

Correspondence to: R. Kähler

1 Introduction

In numerical analysis hierarchical techniques for local refinement have became increasingly popular in recent years because they lead to more reliable results and allow one to simulate more complex phenomena. One such scheme is adaptive mesh refinement (AMR), introduced by Berger in the 1980s [5]. In this approach, a hierarchy of nested grids is generated, representing relevant regions of the computational domain of different levels of resolution.
This numerical technique is applied in many fields, like hydrodynamics [4], meteorology [3] and, in particular, astrophysics [8]. An increasing number of scientists are in need of appropriate interactive visualization techniques to interpret and analyze 3D AMR simulation data. They require tools for both 2D analysis to quantitatively convey the information within single slices and 3D representation to quickly grasp the overall structure. A popular technique for interactive visualization of scalar data is texture-based volume rendering, which is able to make advantageous use of modern graphics hardware.
In this paper we present a 3D texture-based algorithm for the volume rendering of AMR data that can achieve interactive frame rates even for large hierarchies. It directly exploits the hierarchical structure of the AMR data, thereby accelerating rendering while also allowing a high level of detail in 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 of related work in the field of multi-resolution volume visualization. The AMR data structure is explained in Sect. 3. In Sect. 4, we discuss the problem of interpolation artifacts at grid boundaries. The space decomposition and the associated data structure utilized for volume rendering of AMR hierarchies are described in Sects. 5 and 6. In Sect. 7, the adaptive node selection strategies, as well as opacity corrections, are presented. We discuss the 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 [10]. In particular, they compared axis- and viewpoint-aligned approaches. Cabral et al. pointed out the correspondence between volume rendering integrals and Radon transformations [9]; 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 data sets to represent regions of the data volume of different levels of resolution. Additionally, they proposed different strategies for view-dependent node selection and introduced the use of spherical shells as proxy geometries. Weiler et al. presented a similar approach, paying special attention to avoid interpolation artifacts at the boundaries of adjacent cells of different levels of resolution [20]. 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 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 [7].

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 [14]. 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 [15]. Ma presented a parallel volume renderer for AMR data generated by the PARAMESH package [13, 16]. Weber et al. proposed a software and hardware-accelerated cell-projection algorithm for AMR data in [17]. They further proposed the use of different types of stitching cells to connect cells of different levels of resolution and applied this technique for crack-free isosurface extraction of AMR data [18] and a high-quality software cell-projection algorithm [19]. This approach avoids resampling of cell-centered data but has the drawback that it restricts the applicability of their algorithm to AMR schemes, which guarantee that refined levels are surrounded by at least one layer of cells from the next coarser level.

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

We present a 3D texture-based volume-rendering algorithm that 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 the rendering of subgrids can be omitted if their parent cells have subpixel size. We propose a new space-partitioning scheme that decomposes the volume into axis-aligned bricks of the same level of resolution. The procedure creates a small number of bricks and is utilized for view-consistent rendering. We further avoid interpolation artifacts at grid transitions by interpolating continuously even at boundaries between subgrids of arbitrarily different resolution.

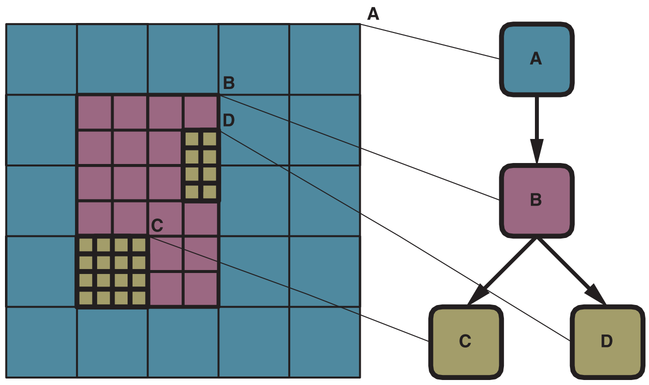
3 The AMR data structure

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 some refinement factor. Figure 1 shows a 2D example. This process repeats until all leaf grid cells satisfy certain error criteria, which depend on the particular numerical simulation.

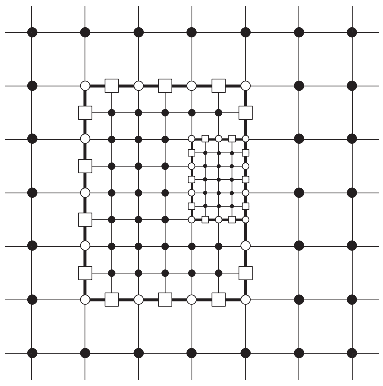
The data values are usually stored at the grids nodes (vertex-centered) or at the centers of the cells (cell-centered). For simplification purposes the AMR schemes usually fulfill the following restrictions:

- 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.

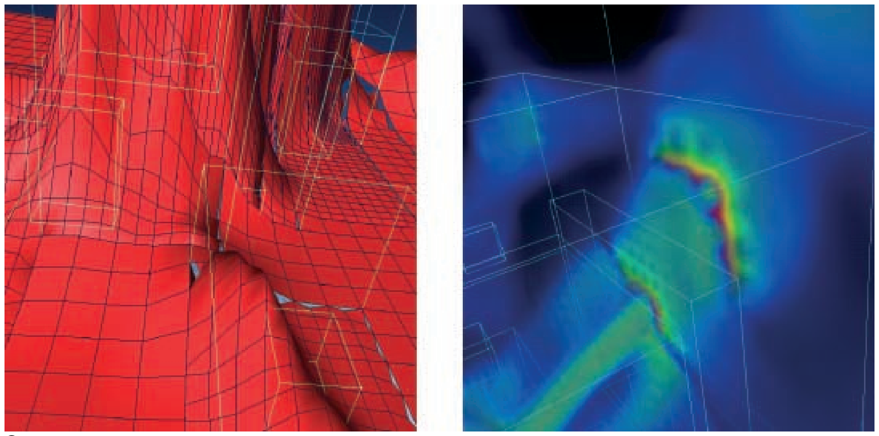
Notice that the resolution levels of adjacent cells may differ by more than 1. This has to be taken into account during interpolation to avoid artifacts due to discontinuities at grid boundaries.



1



3



2

Fig. 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*)

Fig. 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

Fig. 3. A 2D example of an AMR grid with two subgrids: Data values on common boundary nodes (*white circles*) are equal, hanging nodes (*white rectangles*), which are nodes with no counterpart on the next coarser level, are obtained by linear interpolation (bilinear in the 3D case) between adjacent common node

4 Interpolation

Artifact-free volume rendering of AMR data requires a globally continuous interpolation of the discrete data (c.f. Fig. 2). In this section, we address the problem for both vertex-centered and 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. 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 resolutions 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 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 (usually called *hanging* or *dangling* nodes) have to be obtained by bilinear interpolation between the adjacent nodes. This replaces some of the fine boundary nodes, but 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 APIs like OpenGL can be used to render the data directly and give scientists the possibility of examining the unmodified results of their simulations. Of course this scheme has the drawback that the resulting images usually show quite granular structures. So, one still needs an approach that achieves better image quality. Defining the dual grid of each AMR grid as a 3D texture and employing trilinear interpolation would result in gaps between the grids. Inserting different types of stitching cells to fill the gaps, as worked out in [18, 19], is not applicable to volume rendering via 3D textures, since the texels are supposed to be equidistant.

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 eight surrounding cell-centered data values, as proposed in [14] and [13]. For the boundary nodes, we distinguish the following cases:

- If 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, the more accurate values of the finer grids are also 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 as proposed in [13] and project the values to the coarse nodes. But we experienced that this can introduce artifacts on the corresponding layer of coarse grid cells, which become visible if subgrids are omitted during rendering to increase the performance, see Sect. 7. This holds, for example, if the values on the higher-resolved levels strongly increase, since in this case the values at the replaced boundary nodes on the coarse grid tend to be larger than the adjacent ones.

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

In this section, we assume that the reader is familiar with texture-based volume rendering, as for example proposed in [9, 10].

A possible approach for volume rendering an AMR hierarchy via 3D textures would be to process each slice separately in a view-consistent order, followed by a blending step in the frame buffer. First the polygons resulting from the intersection of the first slice 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. Then the polygons resulting from intersection of this slice with grids on the next coarser level would be rendered, with the stencil buffer being updated appropriately. This would continue until the polygons on the root-level grid are processed. Then the next slice would be processed, blending the resulting polygon into 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.

In order to avoid multiple rendering of regions that are covered with cells of different resolution, we subdivide the data volume into non-overlapping axis-aligned regions (called *bricks* in the following) that consist either completely of cells being refined by

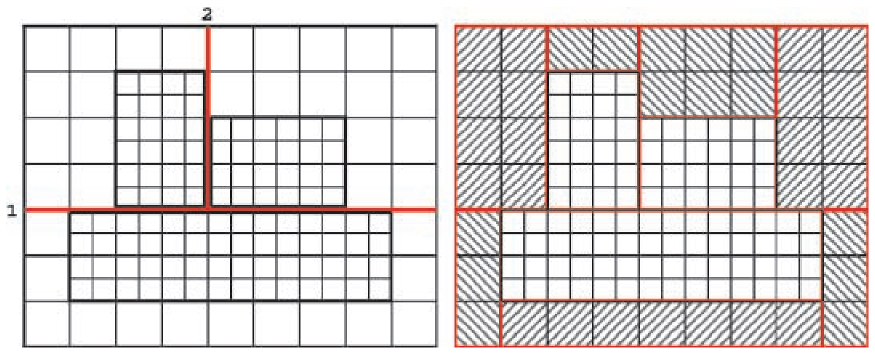


Fig. 4. *Left*: 2D example of one grid, that contains three subgrids on the next level of resolution. In order to subdivide the volume into axis-aligned, non-overlapping regions containing only cells of the same refinement level, two split axis are determined. *Right*: Subregions that contains just one subgrid, are finally partitioned in up to four subregions (up to six in the 3D case)

subgrids, or of cells that are not further refined. In the rendering step, each of these bricks is processed separately in a view-consistent order. So it has to be ensured that there are no visibility cycles between these bricks.

Notice that for an octree data structure this decomposition is explicitly given. In principle one could overlay the computational domain with an octree and subdivide it until each octant (brick) contains only cells of the same resolution level. This usually leads to a very large number of bricks, since many octants would have to be subdivided until they contain only one cell. But minimizing the number of bricks is important in texture-based approaches, in order to reduce the amount of computation for intersection and texture coordinates, as well as I/O overhead due to texture paging.

Weber et al. also proposed a scheme for decomposing the domain into regions of refined and unrefined cells and used it for hardware-accelerated cell projection [17]. This approach usually splits grids and their subgrids and hence increases the number of bricks.

We propose a decomposition heuristic that creates only few regions and avoids the splitting of subgrids in most cases. To see how this decomposition works, consider a simple 2D example of one coarse grid that contains three subgrids on the next level of refinement, as shown in Fig. 4. At first the region covered by the coarse grid is subdivided in such a way that one subgrid lies on one side and the other two subgrids lie on the other side of the split (split 1 in left side of Fig. 4). The subdomain with the two grids is split again at their common boundary. Now every subvolume contains exactly one subgrid. In a last step, the coarse cells are enclosed by up to four axis-aligned bounding boxes (six in the 3D case) as indicated in the right picture in Fig. 4.

It is straightforward to generalize this scheme to general grid configurations. In some cases, several splits might be performed to obtain the partition. In this case, the split with the most similar number of grids on both sides of the split is chosen. There might further exist configurations for which it is not possible to find such a partition position, such as if the subgrids form visibility cycles for certain viewpoints. 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 Berger et al. [6], which create subgrids by binary space partitioning. This ensures that always at least one partition position exists, that does not intersect any of the subgrids.

In some cases, the partition scheme mentioned above still produces unnecessarily many bricks – see for example the left side of Fig. 5. In order to reduce the number of bricks in such cases, we enclose the subgrids with a minimal bounding box, and first decompose the outer region, like indicated on the right side of Fig. 5. Notice that this never increases the number of created bricks. For our data sets, this step reduced the total number of leaf bricks by up to 10%. This space decomposition is stored in a kD-tree data structure. For each grid, a separate 3D texture is

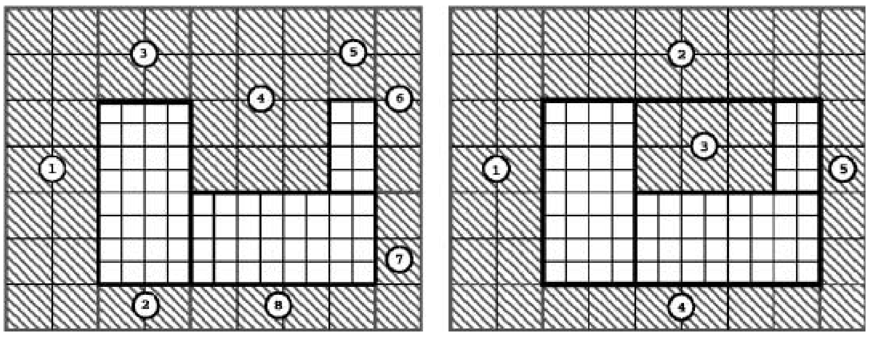


Fig. 5. *Left*: Subgrid configuration where the unmodified decomposition leads to an unnecessarily high number of bricks. *Right*: By first enclosing the subgrids with a minimal bounding box, and decomposing the outer region, the number of created bricks is reduced

allocated (or inserted into a bigger texture in the case of texture packing, as described in 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, information about the partition axis and references to the two subnodes that are associated with the first two subvolumes are also stored.
In a next step, the subnode information is determined. In the case of a subregion containing 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. To avoid artifacts originating from discontinuities between adjacent grids during the rendering, one has to assure that adjacent bricks share one row of data samples at their common boundary faces.
If the subregion covers both refined and unrefined grid cells, only information about the next partition axis and pointers to the next two subregions are stored. These nodes are used to traverse the hierarchy from back to front in the rendering step. In the case that the subregion contains only grid cells that are further refined by a subgrid, a reference to that AMR subgrid texture object is stored. 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 by, say, clamping the boundary texels and restricting the generated texture coordinates to the unextended area. With regard to 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. For more details refer to [11].

7 Rendering

As mentioned in Sect. 5, the brick structure is utilized for traversing the separate bricks from back to front, starting at the root node. If a node which stores texture brick information is processed, 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, and so represents a grid that is further refined. In this case, it has to be checked if it suffices to render this region with this level of detail, or if its subnodes need to be visited because higher visual accuracy is required.

In the second case, the following steps are performed in order to check whether a brick is selected for rendering. Since it is sufficient to render regions with a level of detail such that projected cells have subpixel size in screen space, we determine the extent in screen space of a ball centered at the grid's bounding box corner closest to the view point and with a diameter equal to the grid cells' 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 rendered, omitting its subnodes. Notice that rendering performance can be increased by weakening this criterion from subpixel sizes to larger screen extents.
Further a maximal level at which the hierarchy traversal is stopped, can be specified. This can be 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, 10]. 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 color-table 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. Since the interpolation function continuity 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 [20], we accomplish this by defining a separate color map for each level l of the hierarchy with opacity values $\alpha_i[l]$, $l = 0, 1, \ldots l_{\max}$, where the index i numbers the color map 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 $\alpha_i[0] = 1 - e^{-\tau_i d_0}$. We obtain the opacity entries for a higher-resolved level l as

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

where r is the relative refinement factor. Before a subregion of the hierarchy is rendered the appropriate color table is activated.

8 Results and discussion

The algorithm has been implemented in Amira, an object-oriented, extendible 3D data visualization system developed at ZIB [1, 2]. The runs were performed on a single 195 MHz MIPS R10k processor of a SGI Onyx2 system. The performance was tested on a single InfiniteReality2 pipeline with two RM7 raster managers and 64 MB of texture memory each.
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 image examples the size of the viewport was 764×793 pixels.
Data set I, resulting from an AMR galaxy cluster simulation, consists of 91 grids on 7 levels of refinement. The (resampled) root level contained 33^3 data samples and was rendered with 120 slices, the more refined grids with progressively more. If resampled to an uniform grid, the grid would contain more than 4000^3 data samples, corresponding to about 70 000 MB of texture memory.
Data set 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 data set 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].

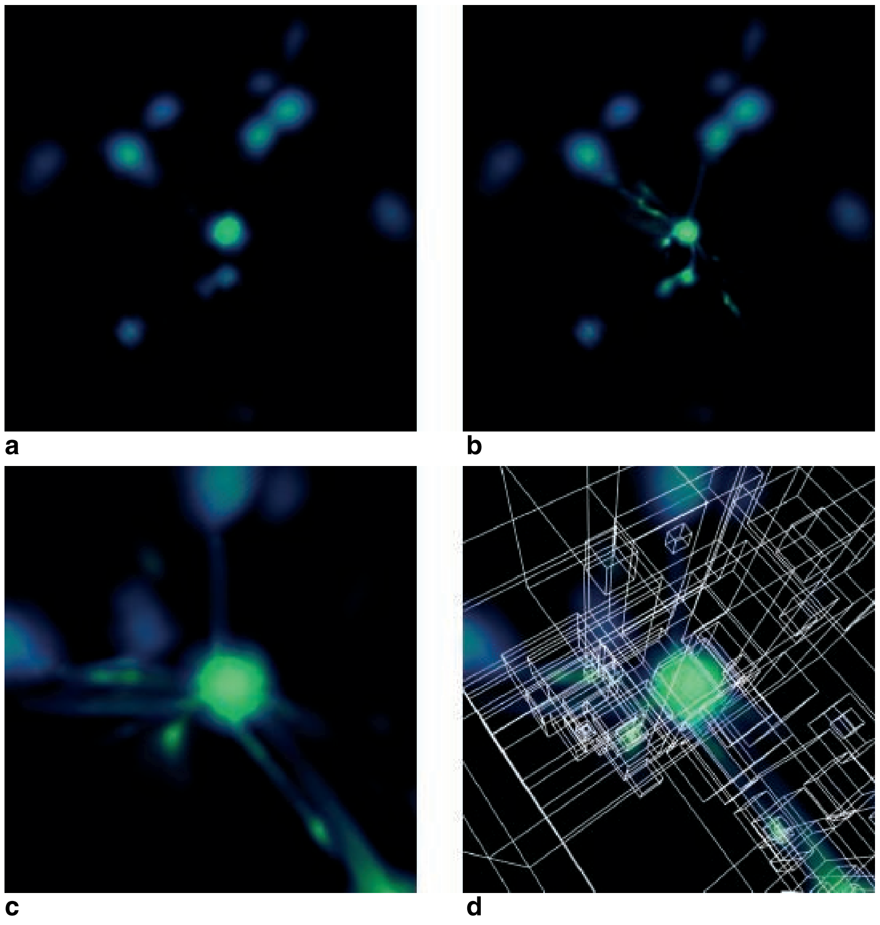


Fig. 6a–d. Data set I. The result of a galaxy cluster AMR simulation, consists of 91 subgrids on 7 levels of refinement: **a** root level; **b** full hierarchy; **c** close-up view of refined inner region; **d** associated bounding boxes

Data set III is an AMR hierarchy resulting from a cosmological AMR simulation that consists of 813 grids on 9 levels of refinement. The (resampled) root grid contained 129^3 data samples and was rendered with 250 slices. If resampled to an uniform grid, the grid would contain about $66\,000^3$ data samples, resulting in an amount of 2.7×10^8 MB of texture memory.

Table 1 lists the number of leaf bricks created, 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 number 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. 6–8. They show the root grid (a), the full hierarchy rendered with all bricks (b), a close-up view of the

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

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

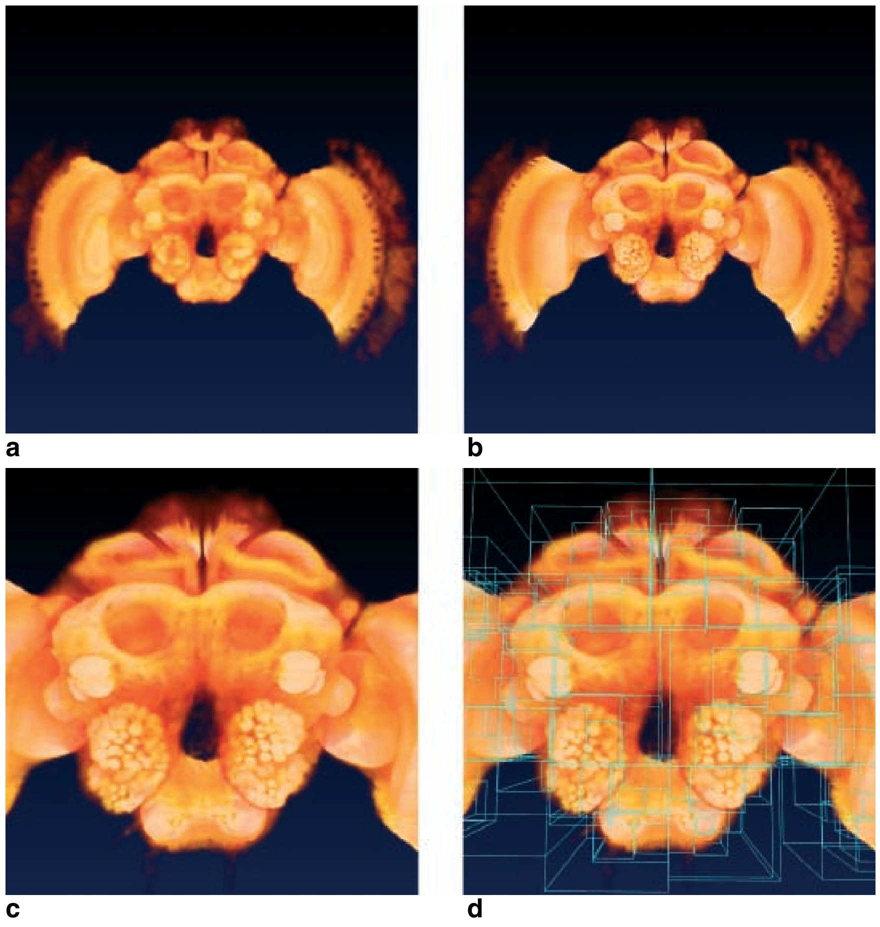


Fig. 7a–d. Data set II. An AMR tree generated from a confocal microscopy image stack of a bee's brain, consists of 359 subgrids on 4 levels of refinement: **a–d** as in Fig. 6

refined part of the hierarchy (c) and the associated bounding boxes of the subgrids (d).

Looking at the close-up views in Figs. 6–8 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. Weiler et al. addressed that problem in [20].

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. 6–8. The last entry represents the frame rate achieved when rendering the full hierarchy in the mode described in Sect. 7, where bricks are omitted if their parent grids have subpixel size. Rendering the data sets with the stencil buffer approach as described in Sect. 5 was about 3 times slower than the frame rates for the decomposition ap-

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

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

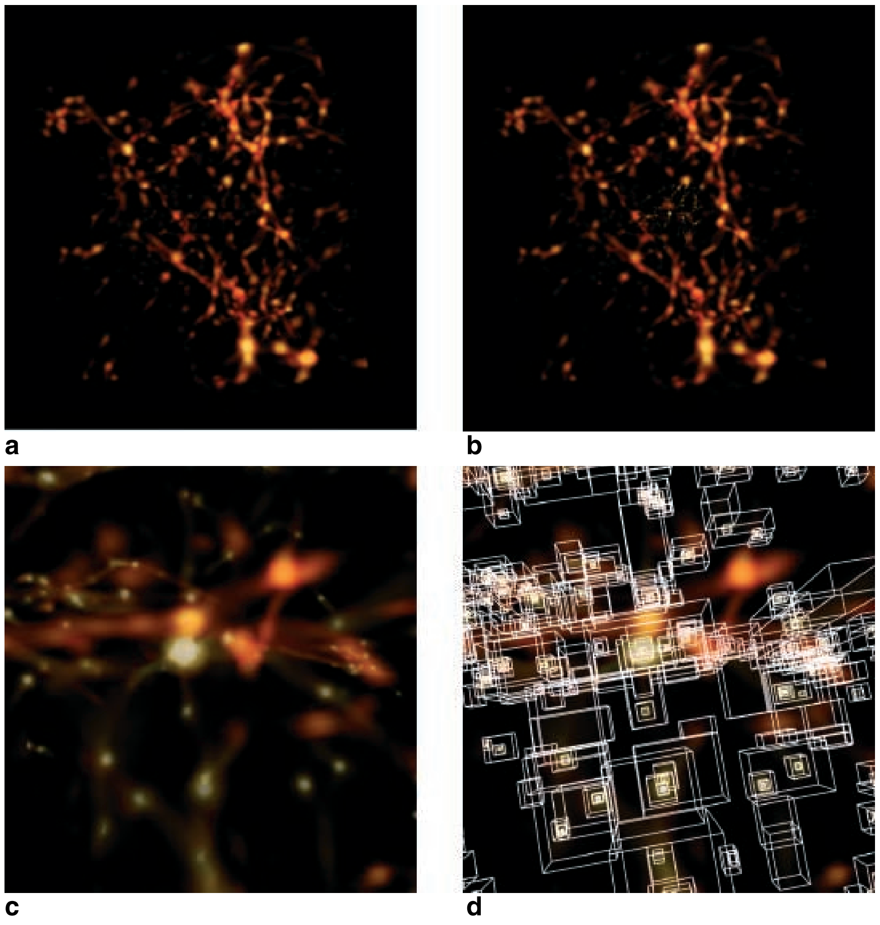


Fig. 8a–d. Data set III. The result of a cosmological AMR simulation, consists of 813 subgrids on 9 levels of refinement: **a–d** as in Fig. 6. For clarity only a subset of the bounding boxes was rendered in image **d**

proach in 2. Rendering the uniform data set being used to generate data set II with the standard approach for texture-based volume rendering resulted in frame rates below 2 fps. The amount of texture memory was 64 MB.

For all data sets, almost 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 data set 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 cells of 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 [6]. 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.
As future work, 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. [20] without significantly decreasing the rendering performance. Furthermore, the application of data homogeneity and importance-based criteria as proposed by Boada et al. in [7] could be utilized for pruning branches of AMR hierarchies. For camera positions inside the volume, such as in immersive environments, the use of spherical shells as proxy geometries, as introduced by LaMar et al. [12], could also be investigated. Furthermore, the use of improved packing schemes decreasing the amount of texture memory consumption would be interesting.

Acknowledgements. We thank Detlev Stalling for helpful discussions and proofreading. The cosmological simulation data sets were provided by Mike Norman and Brian W. O'Shea (National University of California, San Diego), and Stuart Levy (National Center for Supercomputing Applications (NCSA), Urbana, Illinois). The confocal bee brain data set was provided by Robert Brandt (research group Randolf Menzel, Freie Universität at Berlin).
This work was supported in part by the Max-Planck-Institut für Gravitationsphysik (Albert-Einstein-Institut) in Potsdam, Germany.

References

1. Amira (2001a) Amira User's Guide and Reference Manual. Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB) and Indeed Visual Concepts GmbH, Berlin. http://www.amiravis.com. Cited 17 July 2002
2. Amira (2001b) Amira Programmer's Guide. Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB) and Indeed Visual Concepts GmbH, Berlin. http://www.amiravis.com. Cited 17 July 2002
3. Almgren AS, Bell JB, Colella P, Howell LH, Welcome ML (1997) A high-resolution adaptive projection method for regional atmospheric modeling. In: Delic G et al. (ed) Next generation environment models and computational methods. Workshop held in Bay City, MI (USA), August 7–9, 1995. Philadelphia, PA. SIAM Proc Appl Math: 69–79
4. Berger MJ, Collela P (1989) Local adaptive mesh refinement for shock hydrodynamics. J Comput Phys 82(1):64–84
5. Berger MJ, Oliger J (1984) Adaptive mesh refinement for hyperbolic partial equations. J Comput Phys 53:484–512
6. Berger MJ, Rigoutsos I (1991) An algorithm for point clustering and grid generation. IEEE Trans Syst Man Cybern 21(5):1278–1286
7. Boada I, Navazo I, Scopigno R (2001) Multiresolution volume visualization with a texture-based octree. Vis Comput 17(5):185–197
8. Bryan GL (1999) Fluids in the universe: adaptive mesh refinement in cosmology. Comput Sci Eng 1(2):46–53
9. Cabral B, Cam N, Foran J (1994) Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In: Kaufman A, Krueger W (eds) 1994 Symposium on Volume Visualization. IEEE Computer Society Press, Los Alamitos, Calif., pp 91–98
10. Cullip T, Neumann U (1993) Accelerating volume reconstruction with 3D texture mapping hardware. Technical report TR93-027. Department of Computer Science, University of North Carolina, Chapel Hill
11. Kähler R, Simon M, Hege HC (2001) Fast volume rendering of sparse datasets using adaptive mesh refinement. ZIB-Report 01-25 July 2001 (to appear in IEEE Transactions on Visualization and Computer Graphics)
12. LaMar EC, Hamann B, Joy KI (1999) Multiresolution techniques for interactive texture-based volume visualization. In: Ebert D, Gross M, Hamann B (eds) Proceedings of IEEE Visualization '99, San Francisco, 24–29 October 1999. IEEE Computer Society Press, Los Alamitos, Calif., pp 355–362
13. Ma K-L (1999) Parallel rendering of 3D AMR data on the SGI/Cray T3E. In: Proceedings of the 7th Symposium on Frontiers of Massively Parallel Computation. IEEE Computer Society Press, Los Alamitos, Calif., pp 138–145
14. Max NL (1993) Sorting for polyhedron composition. In: Hagen H, Müller H, Nielson GM (eds) Focus on Scientific Visualization. Springer, Berlin Heidelberg New York, pp 259–268
15. Norman M, Shalf J, Levy S, Daues G (1999) Diving deep: data-management and visualization strategies for adaptive mesh renement simulations. Comput Sci Eng 1(4):22–32
16. PARAMESH (1998) NASA Goddard Space Flight Center. http://webserv.gsfc.nasa.gov/rib/repositories/inhouse_gsfc/paramesh.html. Cited 17 July 2002
17. Weber GH, Hagen H, Hamann B, Joy KJ, Ligocki TJ, Ma KL, Shalf JM (2001) Visualization of adaptive mesh refinement data. In: Erbacher RF, Chen PC, Roberts JC, Wittenbrink CM, Groehn M (eds) Visual Data Exploration and Analysis VIII, Proc. SPIE Vol. 4302, SPIE – The International Society for Optical Engineering, Bellingham, Washington, pp 121–132
18. Weber GH, Kreylos O, Ligocki TJ, Shalf JM, Hagen H, Hamann B, Joy KI (2001) Extraction of crack-free isosurfaces from adaptive mesh refinement data. In: Data Visualization 2001 (Proceedings of VisSym '01). Springer, Berlin Heidelberg New York, pp 25–34
19. Weber GH, Kreylos O, Ligocki TJ, Shalf JM, Hagen H, Hamann B, Joy KI (2001) High-quality volume rendering of adaptive mesh refinement data. In: Proceedings of Vision, Modeling, and Visualization 2001, Stuttgart. IOS Press, Amsterdam, pp 121–128
20. Weiler M, Westermann R, Hansen C, Zimmerman K, Ertl T (2000) Level-of-detail volume rendering via 3D textures. In: IEEE Volume Visualization and Graphics Symposium 2000. IEEE Computer Society Press, Los Alamitos, Calif., pp 7–13

RALF KÄHLER studied physics at the Free University of Berlin, where he received his MS degree in 1999. Afterwards he joined the Konrad-Zuse-Zentrum Berlin (ZIB) as a research scientist in the Scientific Visualization department. His research interests include computer graphics and data visualization. Currently, he is working in the field of hierarchical methods in volume rendering.

HANS-CHRISTIAN HEGE is head of the Scientific Visualization department at Zuse Institute Berlin (ZIB, www.zib.de). He studied physics at the Free University of Berlin. From 1984 to 1989, he was a research assistant in the physics department, working in quantum field theory and numerical physics. In 1986, he co-founded Mental Images and GDS. From 1986 to 1989, he worked as a researcher at Mental Images and as managing director at GDS. Since 1989, he has been with ZIB, a non-university research institute of the State of Berlin, operating in the field of information technology. At ZIB, he built up and now directs the Scientific Visualization department. In parallel, he acts as managing director of Indeed Visual Concepts, a company specializing in data visualization which he co-founded in 1999. His current research interests are in computer graphics, data visualization, image analysis, and biomedical computing.