Visualization of Adaptive Mesh Refinement Data

Gunther H. Weber[1,2], Hans Hagen[1], Bernd Hamann[1,3], Kenneth I. Joy[1],
Terry J. Ligocki[3], Kwan-Liu Ma[1], and John M. Shalf[3,4]

[1]Center for Image Processing and Integrated Computing, Department of Computer Science, University of California, Davis
[2]Department of Computer Science, University of Kaiserslautern, Germany
[3]Lawrence Berkeley National Laboratory, National Energy Research Scientific Computing Center
[4]National Center for Supercomputing Applications, University of Illinois, Urbana-Champaign

ABSTRACT

The complexity of physical phenomena often varies substantially over space and time. There can be regions where a physical phenomenon/quantity varies very little over a large extent. At the same time, there can be small regions where the same quantity exhibits highly complex variations. Adaptive mesh refinement (AMR) is a technique used in computational fluid dynamics (CFD) to simulate phenomena with drastically varying scales concerning the complexity of the simulated variables. Using multiple nested grids of different resolutions, AMR combines the topological simplicity of structured-rectilinear grids, permitting efficient computation and storage, with the possibility to adapt grid resolutions in regions of complex behavior. We present methods for direct volume rendering of AMR data. Our methods utilize AMR grids directly for efficiency of the visualization process. We apply a hardware-accelerated rendering method to AMR data supporting interactive manipulation of color-transfer functions and viewing parameters. We also present a cell-projection-based rendering technique for AMR data.

Keywords: volume visualization, direct volume rendering, adaptive mesh refinement, computational fluid dynamics, multiresolution method, multigrid technique, hardware-accelerated rendering

1. INTRODUCTION

AMR is a method in computational physics introduced by Berger and Oliger.[3] A modified version of their algorithm was published by Berger and Colella.[2] AMR has become increasingly popular in the field of computational physics and is used in a variety of applications in CFD. For example, Bryan[4] simulates astrophysical phenomena using a hybrid approach combining AMR and particle simulation.

An AMR simulation starts by subdividing physical space using one or multiple coarse grids. The governing equations are discretized on this grid and solved using a finite-difference method. Subsequently, simulation is performed, and errors for all cells are estimated. Higher-resolution grids are generated that contain all grid cells whose associated error values exceed a given threshold. There is no general rule governing the placement, orientation, or size of the refining grids, besides the requirement that a refining grid must be completely contained in one or more grids of the subsequent coarser level. New grids are initialized with interpolated values from the parent level. Simulation is performed on each grid separately, and values are propagated back to the parent level to ensure consistency between levels. Simulation and refinement are performed recursively until all regions are captured with the desired accuracy.

In this paper, we visualize astrophysical data sets generated by the Berger–Colella AMR method.[4] These data sets comprise axis-aligned structured-rectilinear grids consisting of hexahedral cells with a volume defined by an "edge-length vector" $\delta_g = (\delta_{g,0}, \delta_{g,1}, \delta_{g,2})$. Each grid can be positioned by specifying its local origin. Due to the finite-difference method used by the simulation, a *cell-centered* data format is used. Dependent function values are associated with the centers of the cells. We denote the region covered by the grid by Γ_g.

An AMR hierarchy consists of several levels Λ_l consisting of one or multiple grids. All grids in a level have the same resolution, *i.e.*, they share the same cell size δ_{Γ_l}. The region covered by a level Γ_{Λ_l} is the union of regions covered by the grids of that level.

The *root level* Λ_0 is the level initially used for the simulation and thus the coarsest level. Each level Λ_l may be refined by a higher resolution level Λ_{l+1}. A grid of the refined level is commonly referred to as a *coarse grid* and a

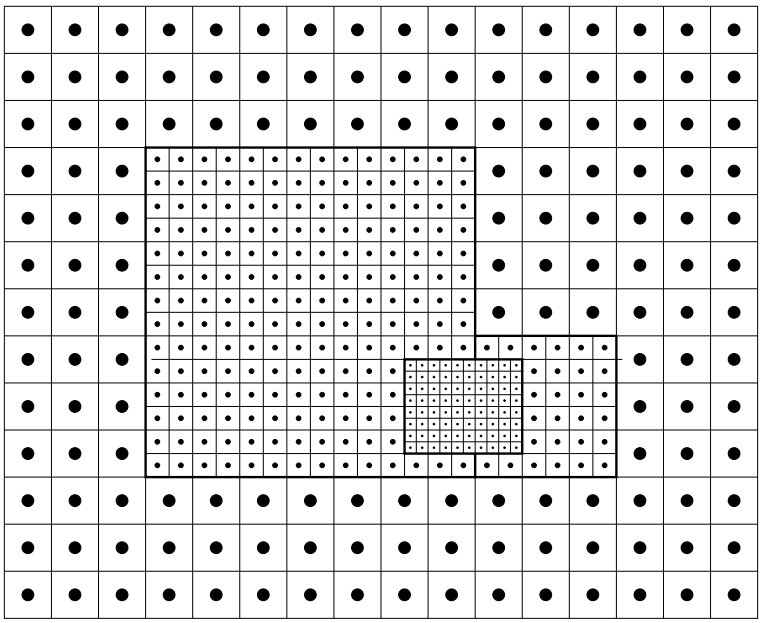


Figure 1. AMR hierarchy consisting of four grids in three levels. Grid boundaries are drawn as bold lines. Locations at which dependent data values are given are indicated by solid circles.

grid of the refining level as a *fine grid*. The *refinement ratio* r specifies how many fine-grid cells fit into a coarse-grid cell, considering all axial directions. (The value of r is always an integer.)

A refining grid refines a whole level Λ_l, *i.e.*, it is completely contained in Γ_{Λ_l} but not necessarily in the region covered by a single grid of that level. Each refining grid can only refine complete grid cells of the parent level, *i.e.*, it must start and end at the boundaries of grid cells of the parent level. Furthermore, there is always a layer of width of at least one grid cell between a refining grid and the boundary of the refined level.

Figure 1 shows a simple two-dimensional (2D) AMR grid hierarchy produced by the Berger–Colella method. It consists of four grids in three levels. The root level consists of one grid. This grid is refined by a second level consisting of two grids. A fourth grid refines the second level; it overlaps both grids of the second level.

The visualization community has just begun to explore the potential benefits of using the hierarchical structure of AMR data directly for improving the efficiency of the visualization process. With the increasing use of AMR in large-scale numerical simulations, and its advantage of concentrating computations where they are really necessary, appropriate visualization techniques become indispensable. In this paper, we describe visualization strategies for volume rendering of 3D AMR data operating directly on the AMR representation. We use a direct volume rendering approach for the visualization of AMR data and present a hardware-accelerated preview rendering scheme as well as a high-quality cell-projection-based approach.

2. RELATED WORK

Little research has been done to date regarding the visualization of AMR data. Published work has mainly focused on converting AMR data into suitable conventional representations and visualizing these, see Norman *et al.*,[14] while trying to preserve the original hierarchical nature. Ma[9] describes a parallel rendering approach for AMR data. Even though he re-samples the data to vertex-centered data, he still uses the hierarchical nature of AMR grids and contrasts that approach to re-sampling the data to the finest available resolution. Max[11] describes a sorting scheme for cells for volume rendering and uses AMR data as one application of his method.

Direct volume rendering is, besides the extraction of isosurfaces and slice-plane-based methods, the most common visualization method for scalar volume data. Using so-called *transfer functions*, scalar values are "mapped" to illumination properties. Regions in the volume are associated with the illumination properties that a transfer function

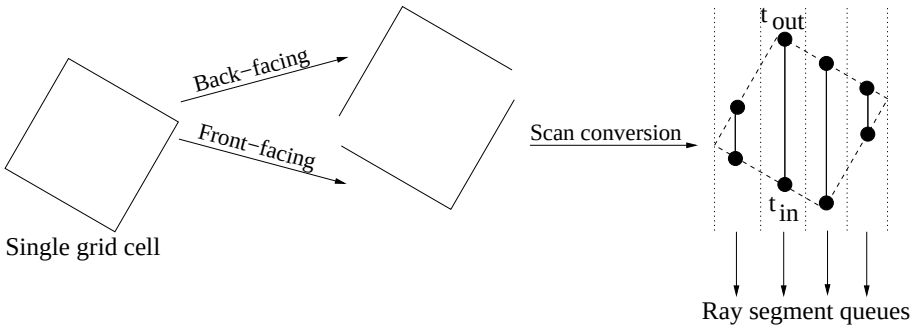


Figure 2. The cell-projection process.

assigns to the scalar values in that region. Direct volume rendering methods can be differentiated by their underlying illumination models (*i.e.*, by the optical "properties" of transfer functions) and whether they operate in *image space* or *object space*. Image-space-based algorithms, *e.g.*, ray casting,[16,7] operate on pixels in screen space as "computational units," *i.e.*, they perform computations on a per-pixel basis. Object-space-based methods, like cell-projection,[10] splatting,[20] or shear-warp factorization[5] operate on cells/voxels as computational units, *i.e.*, they perform computations on a per-cell basis. Sabella uses a light model based on "varying density emitters," resulting in images "perceived as a varying density "cloud" where color highlights the computed attributes".[16] Each scalar value is associated with an emission and a translucency. Light is emitted from each position in the volume and partially absorbed along the path to the image plane. Sabella uses an image-space-based ray casting approach. Rays are cast through each pixel and traced through a volume. Light contributions are integrated along the rays, while considering absorption on the way to the viewer. Levoy[7] uses a light model that simulates reflections of "surfaces" in volume data. Levoy also maps scalar values to colors and opacities. In addition to scalar values, Levoy considers gradients and uses a generalized Phong model to create the impression of shaded surfaces within a volume. Max[12] surveys optical models used for volume rendering.

Direct volume rendering is a computationally expensive method. Numerous attempts have been made to implement systems enabling interactive rendering of volumetric data. Meyer[13] provides an overview and introduces a system that adapts display quality of a rendered data set to maintain interactive rendering rates. Advances in hardware technology, aimed at increasing the performance for surface-based rendering, have also been used to increase the interactivity of volume rendering techniques. Van Gelder and Kim[17] use 3D textures and alpha blending to display volume-rendered images. Today, 3D graphics acceleration is widely available. Through the increasing proliferation of 3D computer games, 3D graphics acceleration hardware is now available on standard PCs at low cost. For example, Rezk-Salama *et al.*[15] present hardware acceleration schemes for volume rendering specifically tailored to the widely available NVIDIA GeForce graphics chip.

Multiresolution volume visualization methods use hierarchical representations of scalar volume data sets to render them at different resolutions, depending on time and/or quality constraints. LaMar *et al.*[6] use an octree-based representation of volume data. Each portion of a volume can be rendered at different resolutions. During rendering, priorities based on an error metric are used to decide which resolution to use for rendering a given region. For actual rendering, a hardware-accelerated slicing process is used, allowing interactive visualization of large-scale volume data.

3. PROGRESSIVE HIGH-QUALITY RENDERING USING CELL PROJECTION

3.1. General Cell-Projection Approach

Cell-projection[10] is an object-space-based volume rendering method. It is similar to ray casting, a widely used image-space-based approach. In both methods, rays are traced through the volume accumulating light along each ray. Ray casting does this on a per-pixel basis, constructing rays for each pixel. Cell-projection-based methods construct all ray segments for the current cell and merge them with existing ray segments.

Usually, a priority queue is maintained for each pixel that collects all ray segments contributing to that pixel. Figure 2 shows the fundamental idea of the cell-projection approach. The boundary faces of all cells are divided into two groups, *front-facing* faces (with normals directed toward the viewer) and *back-facing* faces (with normals

directed away from the viewer). First, the front-facing faces are scan-converted into a buffer. For each pixel influenced by the cell, this buffer holds a depth corresponding to an entry parameter value, called t_{in}, along the ray and an interpolated scalar value at that position. Subsequently, the back-facing faces are scan-converted. For each generated pixel, the depth corresponding to the exit parameter value, called t_{out}, along the ray and an interpolated scalar value are computed. Entry parameter value t_{in} and corresponding scalar value are read from the buffer, and the ray segment reaching from t_{in} to t_{out} is constructed. This ray segment is then inserted into the ray-segment queue of the corresponding pixel. If the ray segment is adjacent to existing ray segments in the ray-segment queue, it is merged with them. Two ray segments are adjacent when the union of their parameter intervals along the ray ($[t_{0,\text{in}}, t_{0,\text{out}}]$ and $[t_{1,\text{in}}, t_{1,\text{out}}]$) is continuous, *i.e.*, when either $t_{0,\text{in}} = t_{1,\text{out}}$ or $t_{0,\text{out}} = t_{1,\text{in}}$ holds. When all cells are processed, each ray segment buffer contains only one ray segment corresponding to the ray originating from the pixel.

If the cells are sorted, *e.g.*, using the scheme of Max,[11] and rendered in either back-to-front or front-to-back order, the queue for collecting the ray segments is not necessary. Instead, newly generated ray segments are always adjacent to already computed ray segments and can be composited directly in the frame buffer. However, in our implementation, we use the queue-based approach and extend it as described in the following section.

3.2. Progressive Refinement Cell Projection for AMR Data

We render an AMR hierarchy by subsequently cell-projecting its constituent grids. Two schemes are possible: Bottom-up rendering starts with rendering the finer grids and proceeds with filling gaps by rendering the corresponding portions of the coarser grids. In a top-down approach, a coarse grid is rendered first. The result can be displayed and used as an intermediate visualization. The rendered image is refined by proceeding to the finer grids and replacing portions of the already rendered image with a version at higher resolution. A simple bottom-up rendering scheme starts by cell-projecting the grids of the finest level. Grids of the coarser levels are subsequently cell-projected. While rendering these coarser grids, special care must be taken of those cells overlapped by a finer grid. Ray segments for the regions covered by these cells are already computed. Thus, these cells must be skipped during the rendering process to obtain a correct result. This is done by using an *intersection map*. The intersection map contains an entry for each cell in the grid that specifies the index of the grid that overlaps this cell, should such a grid exist. During the rendering process, the intersection map entry for the current cell is read. If it specifies an overlapping grid, the cell is skipped. A straightforward extension is to render only a subset of refining grids. In this case, a cell is skipped only when the grid overlapping that cell is already rendered.

By adding supplemental information to each ray segment that is inserted into the ray-segment queues, it is possible to use a top-down rendering approach for AMR hierarchies. For each ray segment, indices of two grids are stored: One index specifies the grid in which the segment was created, *i.e.*, the grid to which the cell belongs. The other index specifies by which grid in the child level the segment is affected, *i.e.*, it specifies rendering which grid would define a more accurate representation of that ray segment. This is the index of the grid refining the cell in which the ray segment was created, and it can be obtained by reading the corresponding entry in the intersection map. Ray segments are only merged when they are adjacent (see previous section) were created in the same and are affected by the same grid. Figure 3 shows three rays traced from a specific viewpoint. When the coarsest grid G_0 is rendered, all ray segments are tagged as being created in grid G_0. Ray A is not affected by any grid, since it does not intersect any refined grid cells. Ray B is divided into three segments; the first is not affected by another grid, the second is affected by grid G_1, and the third is not affected by any grid. After rendering grid G_0 completely, the ray-segment queue for ray A contains exactly one ray segment. The queues for rays B and C contain three and four segments, respectively. Ray B is comprised of a segment from the viewpoint to grid G_1 (solid line segment), a segment that is contained in grid G_1 (dashed line segment), and the segment after leaving grid G_1 (solid line segment). Ray C consists of a segment from the viewpoint to grid G_1 (solid line segment), one segment that is contained in grid G_1 (dashed line segment), a segment that is contained in grid G_2 (solid line segment), and a segment after leaving grid G_2 (dashed line segment). Since ray segments can only be merged when they are affected by the same grid, these segments cannot be merged. We note that, even though ray C intersects also grid G_3, this intersection is not considered until grids G_1 and G_2 are rendered. Only grids of the next-finer level are considered as affecting a given ray segment.

Using this approach of partitioning rays into segments that are affected by finer grids and those that are not, it is straightforward to refine an already rendered image by rendering a finer grid. Before the finer grid is rendered, all ray segments affected by that finer grid are erased from the ray-segment queues. When the finer grid is rendered, the

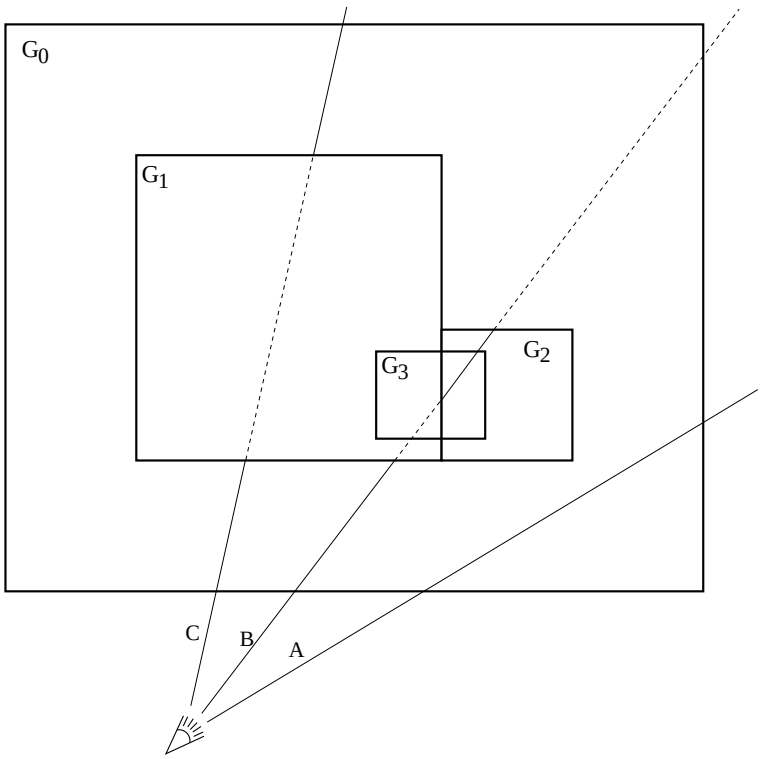


Figure 3. Progressive rendering of AMR hierarchy.

gaps in the ray segments resulting from this step are filled with more accurate ray segments, resulting in an improved image.

3.3. Interpolation Issues

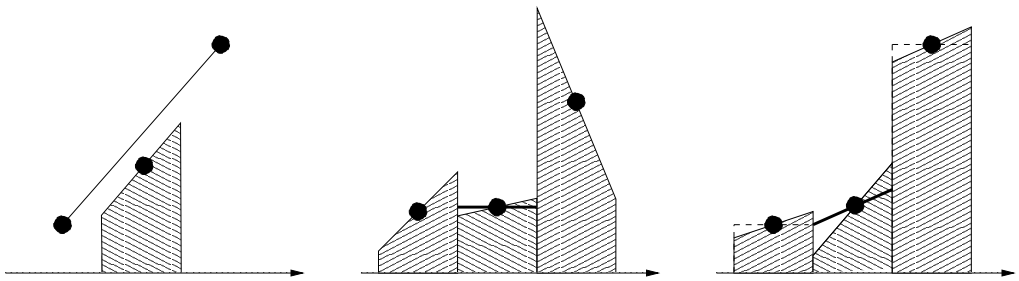
One way to render grids of an AMR data set is to use no interpolation at all and use only original data values. This approach assigns the value at a cell's center to the whole extent of the cell. Scientists who perform simulations favor this approach, as it might make it easier to identify problems in a simulation.

If an interpolation scheme is used, it should be compatible with the interpolation scheme applied during the simulation. Unfortunately, this is complicated by the fact that the underlying finite-difference method has no "inherent" interpolation scheme. Differentiation is approximated by differencing function values of neighboring grid points. The simulation does not use any values besides those at cell centers. AMR, however, uses an interpolation scheme to initialize values of newly created grids. We have implemented an interpolation scheme based on the piecewise linear method (PLM), commonly used in AMR simulations. PLM can be used with simulations of arbitrary dimension. We outline the basic method for the 2D case. PLM estimates the partial derivatives at the center of a grid cell $\mathbf{k} = (k_0, k_1)$ using the difference between grid-cell values, *i.e.*,

$$\left.\frac{\partial}{\partial x_0} f\right|_{\mathbf{x}=\mathbf{p_k}} = \frac{v_{k_0+1,k_1} - v_{k_0-1,k_1}}{2\delta_0} \quad \text{and} \quad \left.\frac{\partial}{\partial x_1} f\right|_{\mathbf{x}=\mathbf{p_k}} = \frac{v_{k_0,k_1+1} - v_{k_0,k_1-1}}{2\delta_1} \,, \tag{1}$$

where δ_0 and δ_1 denote the cell size in the corresponding directions. Figure 4(a) illustrates this approximation for the 1D case (considering one component). The scalar value associated with the cell center is denoted by $v_{\mathbf{k}}$ and its position, *i.e.*, the center of cell $\mathbf{k}$, denoted by $\mathbf{p_k}$. Scalar values in a cell are approximated linearly by adding the first derivative multiplied by the distance from the cell center to the scalar value at the cell's center:

$$PLM(\mathbf{x}) = v_{\mathbf{k}} + (\mathbf{x} - \mathbf{p})\nabla f\Big|_{\mathbf{x}=\mathbf{p_k}} = v_{\mathbf{k}} + (x_0 - p_0)\left.\frac{\partial}{\partial x_0} f\right|_{\mathbf{x}=\mathbf{p}} + (x_1 - p_1)\left.\frac{\partial}{\partial x_1} f\right|_{\mathbf{x}=\mathbf{p}} . \tag{2}$$



(a) (b) (c)

Figure 4. Computing interpolated values using PLM. (a) The first derivative is approximated as the difference between the values at the centers of the two adjacent cells divided by their distance. Values within the cell are interpolated linearly by adding the approximated derivative multiplied by the distance from the cell center to the value at the cell center. (b) If the cell value is a local extremum, the first derivative is set to zero (**heavy** solid line) instead of using the difference between adjacent cell values (solid line). (c) If necessary, the absolute value of the approximated derivative (solid line) is reduced (heavy solid line) to restrict the values in a grid cell between the average levels (dashed lines) of its neighbors.

Using the first derivative according to Equation 1 can result in the introduction of new extrema. PLM prevents this from happening by imposing a limit on the absolute value of the first derivative ("van Leer Limiting[18]"). This is done by extending Equation 1 in the following way: First, a decision is made whether a local extremum is present at the cell center of the current cell. (A local extremum in direction 0 is present when the two differences $v_{(k_0,k_1)} - v_{(k_0-1,k_1)}$ and $v_{(k_0+1,k_1)} - v_{(k_0,k_1)}$ have different signs, *i.e.*, when

$$v_{(k_0,k_1)} - v_{(k_0-1,k_1)} v_{(k_0+1,k_1)} - v_{(k_0,k_1)} \geq 0 \text{ .)} \tag{3}$$

In this case, the corresponding partial derivative is set to zero (Figure 4(b)). Otherwise, the absolute value of the slope is set to

$$\left| \frac{\partial}{\partial x_0} f \Big|_{\mathbf{x}=\mathbf{p_k}} \right| = \min \left(\left\{ \left| \frac{v_{(k_0+1,k_1)} - v_{(k_0-1,k_1)}}{2\delta_0} \right|, \left| 2\frac{v_{(k_0+1,k_1)} - v_{(k_0,k_1)}}{\delta_0} \right|, \left| 2\frac{v_{(k_0,k_1)} - v_{(k_0-1,k_1)}}{\delta_0} \right| \right\} \right) . \tag{4}$$

The last two terms ensure that interpolated values in a cell lie between the average values (*i.e.*, the values at the cell centers) of adjacent cells (Figure 4(c)). When the condition stated in Equation 3 holds, all three terms in the min-expression have the same sign. The sign of the approximated derivative is set to this sign. The other direction, x_1, is handled analogously. This approach generalizes to higher dimensions. (If the values in the 2D case are interpreted as height values over the grid, the interpolant defines a plane with slopes $\frac{\partial}{\partial x_0} f|_{\mathbf{x}=\mathbf{p_k}}$ and $\frac{\partial}{\partial x_1} f|_{\mathbf{x}=\mathbf{p_k}}$ in x_0– and x_1–direction, respectively, passing through the scalar value at the cell center.)

PLM is well suited for interpolation of border values for a new grid. When used for visualization purposes, PLM can cause problems, since it is not continuous at cell boundaries. Even though for most viewing angles integration smoothes out artifacts in volume renderings, distracting artifacts can still result when the view direction is along one of the three major axes and two neighboring rays pass through two different "stacks" of cells. Each cell intersected by the ray has a significantly different color value compared to the color value of the neighboring cells. This effect is accumulated by the integration along the ray.

4. HARDWARE-ACCELERATED VOLUME RENDERING OF AMR DATA

4.1. Interactive Rendering of AMR Data

Visualizations based on direct volume rendering require the specification of meaningful viewpoints as well as transfer functions emphasizing features in a data set. Even though approaches for the automatic creation of transfer functions

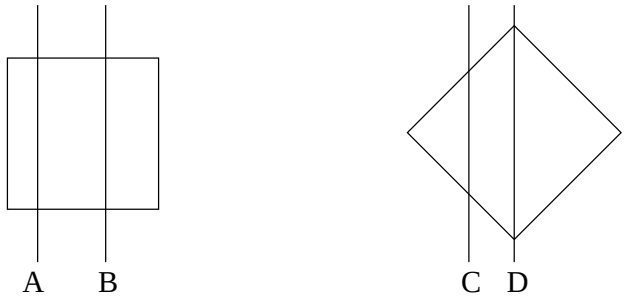


Figure 5. Different lengths of ray traversing cell. (a) Rays axis-aligned. (b) Rays not axis-aligned.

are known, there is still a need for generating customized transfer functions. This is often done on a trial-and-error basis. Thus, it is desirable to have a means for visualizing a volumetric data set at interactive speed. In general, this greatly enhances our understanding of data. This is especially the case for immersive environments. Immersive environments pose demanding requirements for high frame rates. At least 30 frames per second (for stereo-viewing mode) are necessary to achieve the impression of 3D immersion. Therefore, we have adopted hardware-accelerated volume rendering schemes for AMR data.

4.2. Rendering a Single Grid

It is possible to simulate cell projection in hardware. We project the boundary faces of all cells that are directed toward the viewer. By restricting our approach to orthographic projections, we ensure that for all cells the same faces are front-facing and back-facing. Thus, it is sufficient to determine the front-facing faces by considering only one cell and use the determined faces to define the "footprint" of cells. This is similar to splatting using the outline of a single cell as convolution kernel. All front-facing faces are rendered with constant color and opacity. This approach does not take varying cell length into account and produces artifacts in addition to the "blockieness" of the constant interpolation. This is illustrated in Figure 5. When a cell is viewed in axis direction, all rays traverse the same distance in the cell (equal to the length of the cell in that direction), shown in Figure 5(a). When the cell is not viewed in an axis direction, the length of the segment of the ray within this cell depends on the position of the ray. The quick-preview method ignores this fact. We note that the resulting artifacts are similar to the artifacts produced by a slicing approach that uses axis-perpendicular planes to composite volume images. If an image rendered by this method is viewed in the direction of an axis, the results are "correct." The more a viewing direction deviates from an axis direction, the more the individual planes are visible. The resulting renderings are similar to those one would obtain when rendering all three sets of planes at the same time rather than choosing the set that is most perpendicular to the viewing direction. This results in slightly different artifacts. Our cell based-approach allows us to render only cells whose opacity is above a given threshold and thus generate less geometry.

4.3. AMR Partition Tree

Structured-rectilinear grids allow the use of hardware-based schemes to accelerate rendering. AMR data consists of structured-rectilinear grids. However, when viewed over the whole domain there are changes in resolution. In order to apply hardware-accelerated schemes to AMR data, we partition a given data set into blocks of constant resolution using a generalized k-D tree.[1] Each grid is partitioned into two types of blocks of adjacent cells: A block consists completely of cells for which no refined representation is available or all cells of the given block are available in a higher resolution. In the latter case, the block contains a pointer to the finer representation. The resulting blocks are rendered from back to front. During rendering it is determined, for each block, whether it is rendered at low or high resolution to achieve interactive frame rates. It is possible to render only parts of a data set for improved efficiency. In this case, the positions of the higher-level grids indicate where interesting features in a data set exist. (This is possible since these grids are present in regions characterized by high variation, which are exactly the regions that are likely to be of greater interest.)

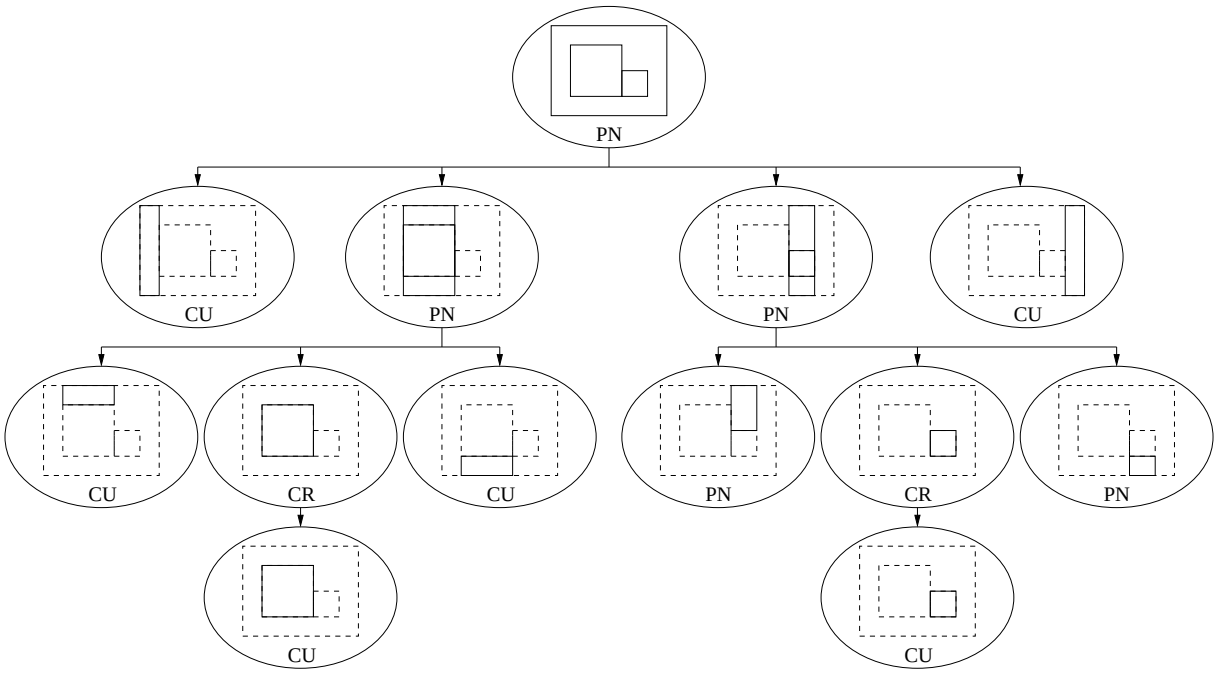


Figure 6. Partition tree for hierarchy shown in Figure 1. Completely refined grid-part nodes are denoted by "CP," unrefined grid-part nodes are denoted by "CU," and partition nodes are denoted by "PN."

.

The partition tree consists of nodes of different types. All nodes share certain common information, regardless of type. Each node specifies a region of an AMR grid. Thus, each node contains at least a reference to this AMR grid and information regarding which cells of the grid are used by this node. These regions are always rectilinear and axis-aligned, allowing their specification by the minimum and maximum cell indices. All cells with indices between these indices belong to the region described by this node. This information alone is sufficient for the "unrefined-grid-part" node. Nodes of this type describe regions in an AMR grid for which no further refinement information is available, regions that can always be rendered at the fixed resolution of the current level. The "completely-refined-grid-part" node describes a region that is completely available at a higher resolution and thus indicates a transition between two hierarchy levels. The corresponding region of the AMR grid is completely available at a higher resolution. Thus, it is possible to render the resolution of the current level or alternatively render it at a higher resolution. This higher-resolution representation is a pointer to a sub-tree describing that region. A "partition" node splits a given region into "stripes" along an axis. Each stripe is an unrefined-grid-part node, a completely-refined-grid-part node, or a partition node (with a different partition direction).

Figure 6 shows the partition tree for the AMR hierarchy from Figure 1. To render the complete hierarchy this partition tree is traversed. Partition nodes are handled by rendering their children in correct order, *i.e.*, by traversing their children lists back-to-front. (We use orthographic projection.) Thus, back-to-front traversal can be achieved by checking the component of the view-direction vector corresponding to the partition direction and rendering the child nodes in a direction according to the sign of the component. Unrefined-grid-part nodes are handled by rendering the corresponding region of the AMR grid using a hardware-accelerated approach. Completely-refined grid-part nodes are rendered by traversing their sub-trees.

4.4. Adaptive Tree Traversal

By using heuristics it is possible to adaptively traverse the partition tree and render images that take certain criteria into account. We differentiate two types of criteria governing the traversal of the partition tree: *Viewpoint-dependent* criteria aim at minimizing rendering time while retaining quality of generated images. We employ two criteria to modify tree traversal. During the traversal of a partition node, a check is performed for each child whether the region corresponding to that node is visible, *i.e.*, whether it intersects the viewing frustum. If the answer is negative,

the corresponding child is skipped, thus culling the corresponding part of the volume. Since an intersection test for bounding box and viewing frustum is too expensive to be performed in real time, we use a heuristic: We project an arbitrary diagonal of the region (the line segment spanning from one vertex of a 3D cell to the opposite-corner vertex) to view coordinates and check whether the bounding box of that diagonal intersects the view region. (In normalized view coordinates this region is given as $[-1,1] \times [-1,1] \times [-1,1]$.) When a completely refined-grid-part node is traversed, a choice must be made: Should the corresponding region be rendered at the resolution of the current level or should it be rendered at a higher resolution? The decision is based on an estimated footprint of the voxel. If each cell of a refining grid covers only sub-pixels, *i.e.*, only fractions of pixels, it is sufficient to render the data with the resolution of the parent level. Again, to perform this decision efficiently enough for real-time rendering we estimate this region by the bounding box of the projected diagonal of a cell.

Considering viewpoint-dependent criteria alone only avoids unnecessary rendering time but does not neccesarily ensure interactive rendering speed. When it is not possible to use the full resolution of an AMR data set and achieve interactive frame rates, greater speed can be obtained by sacrificing rendering quality. Currently, we specify a traversal depth of the partition tree for interactive rendering. For each frame, the time necessary to render it is measured, and an achieved frame rate estimated. If the estimated frame rate is too low, the traversal depth is decreased; if it is higher than the desired frame rate, the traversal depth, and thus the quality of the rendered image, is increased.

5. RESULTS

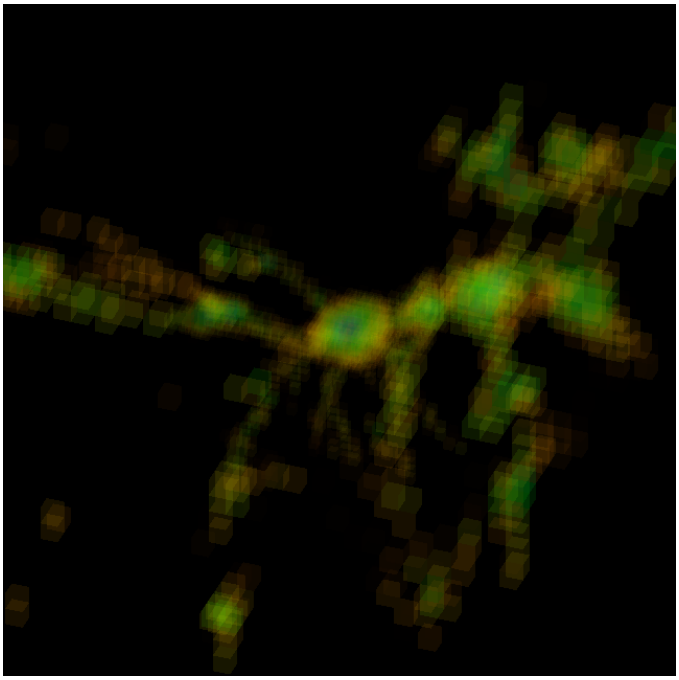
Figure 7 shows images used for interactive preview purposes for AMR data. On an SGI Onyx with InfiniteReality2 graphics, a frame rate of five frames per second can be achieved while using almost the entire AMR hierarchy (five to six of eight levels). Even on an Indigo2 with SolidImpact graphics four to five of eight levels can be rendered with five frames per second, since our method requires no hardware acceleration of textures. Using an NVIDIA GeForce board, only two to three levels can be rendered at five frames per second. The result is shown in Figure 7(a). For merely choosing a viewpoint this quality is sufficient. The best possible quality using the hardware-accelerated approach is shown in Figure 7(b). This image utilizes all levels in the AMR hierarchy and uses the view-dependent criteria merely to avoid unnecessary computation time. This image requires about two seconds on an SGI Onyx with InfinityReality2 graphics and about ten seconds on a PC with NVIDIA GeForce board.

6. CONCLUSIONS AND FUTURE WORK

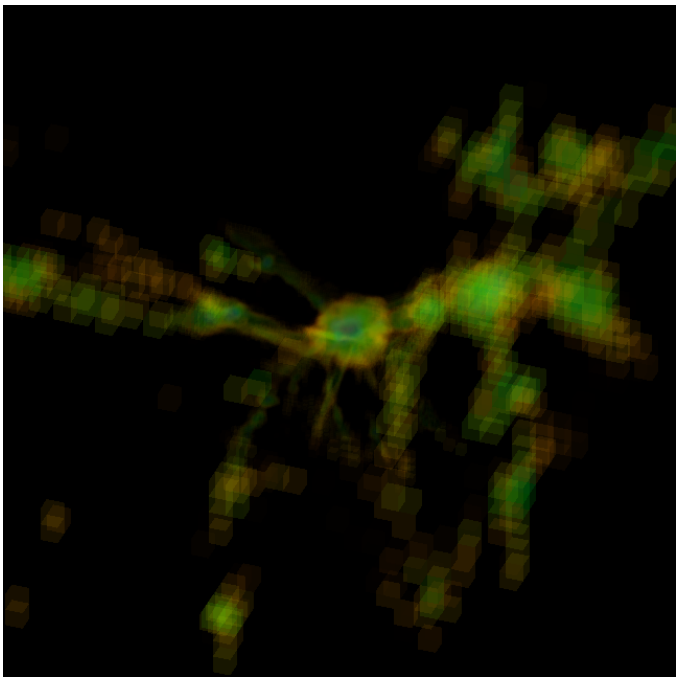
By utilizing the inherent hierarchical structure of AMR data, we can improve efficiency of AMR data visualization substantially. We have presented a hardware-accelerated approach for volume rendering of AMR data at interactive frame rates and a high-quality cell projection method. The high-quality rendering method allows us to progressively render the refining grids, producing images of increasing quality. We pay special attention to the interpolation scheme to make it compatible with the interpolation scheme used in the simulation itself to generate "accurate" images. Our methods thus facilitate the evaluation of simulation results as they enable scientists to identify numerical problems more easily.

The hardware-accelerated and the cell-projection-based approaches offer numerous possibilities for further improvement. The quality of images produced by the cell-projection-based approach can be improved by considering, for example, other illumination models. Furthermore, higher-order interpolation schemes might improve results, even though the integration along rays already tends to "smooth out" discontinuities. Our recent work on the extraction of isosurfaces[19] could be combined with volume rendering to yield higher-quality images, without discontinuities at boundaries between different hierarchy levels. (The current scheme can produce discontinuities at boundaries between individual cells.)

Concerning the hardware-accelerated approach, it would be interesting to compare it to existing approaches, most notably the slice-plane approach mentioned in this paper. The relatively poor performance of the used NVIDIA GeForce board (compared to the rendering speeds on SGIs) gives rise for additional research. Furthermore, the AMR hierarchy can be used to guide a viewer for navigation through a data set and/or render only regions of interest. Our approach of adapting the traversal depth tends to oscillate. More work is necessary to reduce this effect. Combining viewpoint-dependent and framerate-dependent criteria could assign priorities to different nodes and determine regions where quality is to be sacrificed for rendering speed, yielding a more flexible, adaptive traversal strategy for AMR partition trees.



(a)



(b)

Figure 7. Images generated by hardware-accelerated volume renderer (data set courtesy of Greg Bryan, Massachusetts Institute of Technology, Theoretical Cosmology Group, Cambridge, Massachusetts). (a) Data set rendered during user interaction based on two coarsest-level grids in AMR hierarchy. (b) Data set rendered at full resolution.

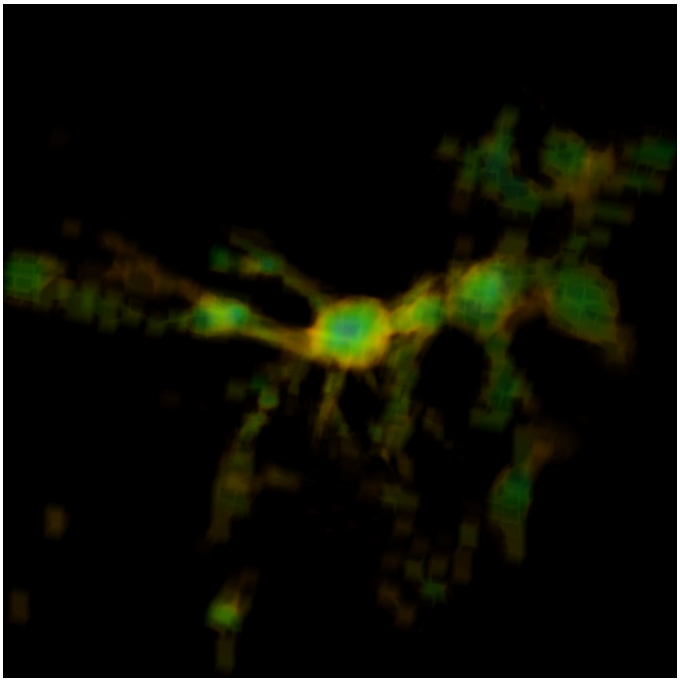


Figure 8. Final high-quality rendering using viewpoint and transfer function used for Figure 7.

7. ACKNOWLEDGMENTS

This work was supported by the Directory, Office of Science, Office of Basic Energy Sciences, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098; the Lawrence Berkeley National Laboratory; the National Science Foundation under contracts ACI 9624034 (CAREER Award), through the Large Scientific and Software Data Set Visualization (LSSDSV) program under contract ACI 9982251, and through the National Partnership for Advanced Computational Infrastructure (NPACI); the Office of Naval Research under contract N00014-97-1-0222; the Army Research Office under contract ARO 36598-MA-RIP; the NASA Ames Research Center through an NRA award under contract NAG2-1216; the Lawrence Livermore National Laboratory under ASCI ASAP Level-2 Memorandum Agreement B347878 and under Memorandum Agreement B503159; the Los Alamos National Laboratory; and the North Atlantic Treaty Organization (NATO) under contract CRG.971628.

We also acknowledge the support of ALSTOM Schilling Robotics and SGI. We thank the members of the LBNL Visualization Group; the LBNL Applied Numerical Algorithms Group; the Visualization and Graphics Research Group at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis, and the AG Graphische Datenverarbeitung und Computergeometrie at the University of Kaiserslautern, Germany.

REFERENCES

1. Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975.
2. Marsha Berger and Phillip Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82:64–84, May 1989. Lawrence Livermore National Laboratory, Technical Report No. UCRL-97196.
3. Marsha Berger and Joseph Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512, March 1984.
4. Greg L. Bryan. Fluids in the universe: Adaptive mesh refinement in cosmology. *Computing in Science and Engineering*, 1(2):46–53, March/April 1999.
5. Philippe Lacroute. Fast volume rendering using a shear-warp factorization of the viewing transformation. Ph.D. dissertation CSL-TR-95-678, Stanford University, Computer Science Department, Computer Systems Laboratory, September 1995.

6. Eric C. LaMar, Bernd Hamann, and Kenneth I. Joy. Multiresolution techniques for interactive texture-based volume visualization. In David Ebert, Markus H. Gross, and Bernd Hamann, editors, *IEEE Visualization '99*, pages 355–361, 543 (color plate), IEEE Computer Society Press, Los Alamitos, California, 1999.
7. Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988. (See also corrigendum[8,21]).
8. Marc Levoy. Letter to the editor: Error in volume rendering paper was in exposition only. *IEEE Computer Graphics and Applications*, 20(4):6–6, July/August 2000.
9. Kwan-Liu Ma. Parallel rendering of 3D AMR data on the SGI/Cray T3E. In *Proceedings of Frontiers '99 the Seventh Symposium on the Frontiers of Massively Parallel Computation*, pages 138–145, IEEE Computer Society Press, Los Alamitos, California, February 1999.
10. Kwan-Liu Ma and Thomas W. Crockett. A scalable parallel cell-projection volume rendering algorithm for three-dimensional unstructured data. In James Painter, Gordon Stoll, and Kwan-Liu Ma, editors, *IEEE Parallel Rendering Symposium*, pages 95–104, IEEE Computer Society Press, Los Alamitos, California, November 1997.
11. Nelson L. Max. Sorting for polyhedron compositing. In Hans Hagen, Heinrich Müller, and Gregory M. Nielson, editors, *Focus on Scientific Visualization*, pages 259–268. Springer-Verlag, New York, New York, 1993.
12. Nelson L. Max. Optical models for volume rendering. *IEEE Transactions on Computer Graphics*, 1(2):99–108, 1995.
13. Jörg Meyer. *Interactive Visualization of Medical and Biological Data Sets*. Shaker Verlag, P.O. Box 1290, D-52013 Aachen, Germany, 2000. (Ph.D. disseration, AG Graphische Datenverarbeitung und Computergeometrie, Department of Computer Science, University of Kaiserlautern, Germany, 1999).
14. Michael L. Norman, John M. Shalf, Stuart Levy, and Greg Daues. Diving deep: Data management and visualization strategies for adaptive mesh refinement simulations. *Computing in Science and Engineering*, 1(4):36–47, July/August 1999.
15. Christof Rezk-Salama, Klaus Engel, M. Bauer, Günther Greiner, and Thomas Ertl. Interactive volume rendering on standard PC graphics hardware using multi-textures and multi-stage-rasterization. In *Proceedings SIGGRAPH/Eurographics Graphics Hardware Workshop 2000*, 2000.
16. Paolo Sabella. A rendering algorithm for visualizing 3D scalar fields. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22(4), pages 51–58, 1988.
17. Allen Van Gelder and Kwansik Kim. Direct volume rendering with shading via three-dimensional textures. In Roger Crawfis and Charles Hansen, editors, *1996 Volume Visualization Symposium*, pages 23–30, ACM Press, New York, New York, October 1996.
18. Bram van Leer. Towards the ultimate conservative difference scheme. IV. A new approach to numerical convection. *Journal of Computational Physics*, 23:276–299, March 1977.
19. Gunther H. Weber, Oliver Kreylos, Terry J. Ligocki, John M. Shalf, Hans Hagen, and Bernd Hamann. Extraction of crack-free isosurfaces from adaptive mesh refinement data. (presentated at the NSF/DoE Lake Tahoe Workshop on Hierarchical Approximation and Geometrical Methods for Scientific Visualization, Tahoe City, California, October 2000; see `http://graphics.cs.ucdavis.edu/~hvm00/program.html`).
20. Lee Westover. Footprint evaluation for volume rendering. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24(4), pages 367–376, August 1990.
21. Craig Wittenbrink, Tom Malzbender, and Mike Goss. Letter to the editor: Interpolation for volume rendering. *IEEE Computer Graphics and Applications*, 20(5):6–6, September/October 2000.