Visualization of Time-Dependent Remote Adaptive Mesh Refinement Data

Ralf Kaehler*
Zuse-Institute Berlin (ZIB) and
MPI for Gravitational Physics (AEI)

Steffen Prohaska*
Zuse-Institute Berlin (ZIB)

Andrei Hutanu†
Louisiana State University (LSU)

Hans-Christian Hege*
Zuse-Institute Berlin (ZIB)

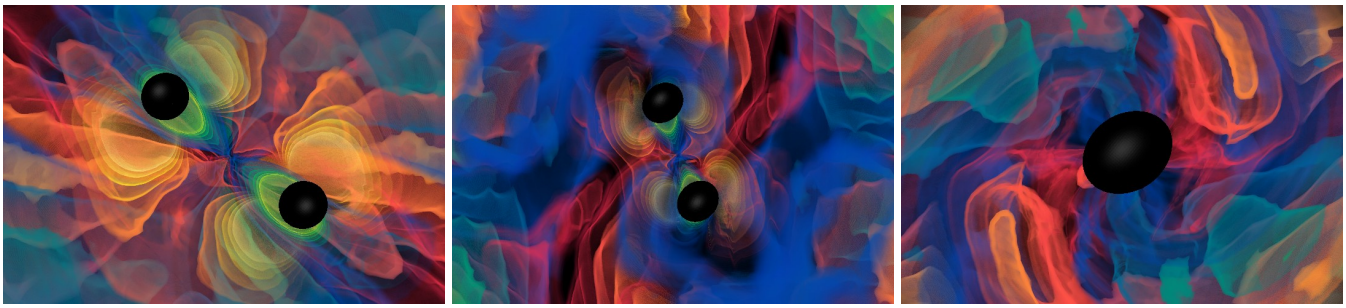

Figure 1: Three interpolated time steps of an AMR simulation of a black hole collision. The shaded isosurfaces depict the event horizons of the black holes, and the volume rendered scalar field shows the gravitational waves that are emitted during the merger.

Abstract

Analysis of phenomena that simultaneously occur on different spatial and temporal scales requires adaptive, hierarchical schemes to reduce computational and storage demands. Adaptive Mesh Refinement (AMR) schemes support both refinement in space that results in a time-dependent grid topology, as well as refinement in time that results in updates at higher rates for refined levels.

Visualization of AMR data requires generating data for absent refinement levels at specific time steps. We describe a solution starting from a given set of "key frames" with potentially different grid topologies.

The presented work was developed in a project involving several research institutes that collaborate in the field of cosmology and numerical relativity. AMR data results from simulations that are run on dedicated compute machines and is thus stored centrally, whereas the analysis of the data is performed on the local computers of the scientists. We built a distributed solution using remote procedure calls (RPC). To keep the application responsive, we split the bulk data transfer from the RPC response and deliver it asynchronously as a binary stream. The number of network round-trips is minimized by using high level operations. In summary, we provide an application for exploratory visualization of remotely stored AMR data.

CR Categories: I.3.4 [Computer Graphics]: Graphics Utilities—Graphics packages; I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types; I.3.8 [Computer Graphics]: Applications

Keywords: Time-Varying Data Visualization, Visualization over Networks, Multiresolution Visualization

*e-mail:{kaehler,prohaska,hege}@zib.de
†e-mail:ahutanu@cct.lsu.edu

1 Introduction

Multi-scale phenomena are abundant in many application fields such as material science, fluid dynamics, geophysics, meteorology and astrophysics. Representing and simulating these processes numerically is a challenging task since different scales must be resolved, requiring enormous amounts of storage and computational power. An important aspect is adaptivity, i.e. local adjustment of the spatio-temporal resolution to the details to be resolved, particularly when time-dependent 3D data is considered. Standard representations for such data are hierarchical, locally refined grids. Therefore, an increasing number of scientists are in need of appropriate visualization techniques in order to help them interpret data defined on such grid structures.

One of the first steps involved in this analysis is exploratory visualization, which requires highly interactive systems. Examples of common tasks involved in this include inspection of a data set to track errors and results; specifying 3D parameters (i.e. sub-volumes or camera paths) or time range for detailed analysis; and specifying parameters for a batch visualization task.

Fulfilling these requirements if all data is stored locally may be achievable. However, in many cases the data is not stored locally, but on dedicated supercomputers. Often, only a tiny fraction of the data is required to perform the visualization and analysis tasks. Transferring a complete data set to a local machine would be a waste of bandwidth and time. This can be avoided by distributing the application which would allow direct access to remotely stored data. The primary goal is for the system to use network resources efficiently while maintaining interactivity. Changes to existing software should be kept minimal. These goals can be attained through thorough analysis of the application and proper choice of distribution.

We present such a solution for a specific type of locally refined hexahedral grids, AMR grids. These grids are employed in a numerical scheme that was originally developed by Berger and Oliger in the context of multi-level techniques for solving hyperbolic partial differential equations [5]. A variant of this approach, *structured* Adaptive Mesh Refinement (SAMR), has gained increasing popularity over the last few years and is presently applied in various domains, such as computational fluid dynamics [3], relativistic as-

trophysics [31, 22] and cosmology [9, 1].

Visualizing time-dependent AMR data is challenging. One of the reasons is that the topology of the AMR hierarchy is not constant over time, but is altered frequently based on local regridding procedures. This implies that during visualization, the missing data must be approximated by interpolation between grid layouts with potentially different topology.

This paper reports on work that has been carried out in collaboration between international research groups. Our application scenario can be characterized as:

- The data is the result of time-dependent, structured AMR simulations in the fields of numerical relativity and cosmology.
- The simulations store various scalar-, vector- and tensorfield quantities. Recent simulations contain several thousand time steps of several field variables defined on up to 600,000 separate subgrids, resulting in storage demands of hundreds of gigabytes per simulation run.
- The data is stored on dedicated supercomputers with limited graphic capabilities and needs to be analyzed interactively by several research groups in the US and Europe.

We describe a visualization system that meets these requirements. It allows to extract and transfer single time steps from remote, time-dependent AMR data to local clients, taking full advantage of local graphics hardware for rendering. Our approach:

- generates an intermediate grid hierarchy by merging the cells on all refinement levels that are present in the key frames,
- induces a nested grid structure on the resulting collection of cells,
- generates intermediate grid functions by interpolating between each set of corresponding data samples on these merged hierarchies,
- executes these steps on a data server and efficiently transfers interpolated data to the client as a binary data stream, and
- controls the data server through remote procedure calls, implemented using SOAP.

2 Related Work

Several papers published in the last few years deal with rendering methods for AMR data. Norman et al. describe an approach of resampling AMR data to uniform and unstructured grids [24]. Weber et al. proposed an approach for crack-free isosurface extraction, as well as, a parallel software and a hardware accelerated cell-projection algorithm for AMR data [38, 40, 39]. Kaehler et al. presented an approach for accelerated texture-based volume rendering of large, sparse data sets by representing non-transparent regions using AMR data structures [19] and a hardware-supported, texture-based volume rendering algorithm for AMR data that directly employs the hierarchical structure of this data type [18, 17]. Park et al. presented a splatting approach [25].

Visualization methods for time-dependent data on unstructured grids have been presented by Polthier et al. [26], Happe et al. [13] and Schmidt et al. [30]. These approaches require the existence of two grids and associated grid functions at each time step at which the underlying grid structure is adapted: the solution before and after grid refinement, respectively grid coarsening. This ensures that on each pair of consecutive time steps the interpolation can be carried out on identical grids.

Applying all these techniques to remotely stored data could be achieved by image-based streaming approaches, as in VNC [29] or Vizserver [34]. These technologies require little or no software development and provide a solution in many use cases; however, they may be limited by missing or low end graphics hardware on the data server. Another restriction is that they do not support loading multiple data sets from different locations in the same visualization session. If the latency of the network is high, image-based streaming approaches suffer from low frame rates and low responsiveness. A round-trip time for overseas connections through the Internet is approximately 150 ms. As one round-trip is required to pass new viewing parameters and return an image the frame rate is limited to 6 fps. Taking into account the processing time of the application; the image readback and encoding; the transfer to the client; and decoding and display at the client the frame rate drops to 1–2 fps. Because the local hardware capabilities are not used, stereo projection or a multi-wall immersive environment like a CAVE cannot be driven directly.

Hybrid approaches like image based rendering assisted volume rendering [23], for example Visapult [7], are able to hide network latency. Billboards are generated on a server and transferred to the client, where they are composited by graphics hardware. Local geometry may also be integrated in this process. This approach was presented for volume rendering. It is not obvious how to extend it to arbitrary visualization methods. For very dynamic user interactions, for example changes to a completely different viewpoint or unanticipated parameter changes, network round-trips are involved in updating the visualization and network latency will become apparent.

Remote file access could be utilized to distribute an application in a simple way using one of the many existing architectures. Using local graphics hardware for rendering guarantees high interactivity. Data sizes are limited by the capabilities of the local hardware. Using the same API for local and remote file access (SRB [4], HDF5 [28]) is convenient but unawareness of the network latency is costly. If multiple accesses (each consuming at least one network round-trip) are necessary to transfer the data of interest, only a small fraction of the available network bandwidth can be utilized.

Systems that do provide high-level selection operations are preferable but they are often either limited to a specific class of selection operations (RIO [10], OpenDAP [11]), data objects (DataCutter [8], Active Data Repository [20]) or both (parallel netCDF [21]). GridFTP [2] can be used to implement user-defined, high-level selection operations but does not provide direct support for specifying complex interfaces.

If low interaction is sufficient—for example the visualization parameters may be known in advance for a class of files—presenting results in a web portal may be useful [16]. This may also be the case if sub sets of the data selected for rendering grow beyond the capabilities of local hardware. Utilizing more capable hardware near the data source for rendering may be the only option. Image-based streaming of hybrid approaches as discussed above can be used to provide restricted interactivity.

We will briefly review the AMR scheme in Section 3. The generation of the intermediate hierarchies and the interpolation is discussed in Section 4 and 5. Next the remote data access is described (Section 6) and we conclude with presenting and discussing results (Section 7).

3 Adaptive Mesh Refinement

The basic idea of AMR is to combine the simplicity of structured grids with the advantages of local grid adaption. In this approach the computational domain is covered by a set of coarse, structured subgrids. During the computation, local error estimators are utilized to detect cells that require higher resolution. These cells are covered by a set of rectangular subgrids. Unlike in finite element approaches, these subgrids do not replace, but rather overlay the

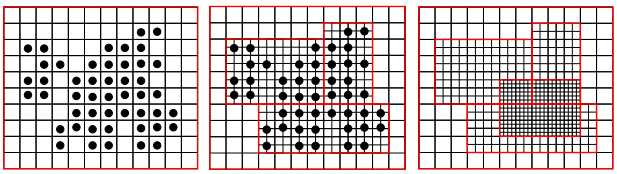

Figure 2: Refinement process for AMR schemes: Cells that require refinement are determined using local error criteria (a) and clustered into separate subgrids (b), which cover the regions with higher resolution. This process is recursively continued until each cell fulfills the error criteria (c).

refined regions of the coarse base grid.

The equations are advanced on the finer subgrids and this refinement procedure recursively continues until all cells fulfill the considered error criterion, giving rise to a hierarchy of nested refinement levels, as shown in Figure 2.

An advantage of AMR is that each subgrid can be viewed as an separate, independent grid with a separate storage space. This allows to process subgrids almost independently during integration and hence the approach is well suited for parallel processing.

In the following we will restrict the discussion to the special case called *structured adaptive mesh refinement (SAMR)*, in which the subgrids can not have arbitrary orientations, but are rather aligned with the major axes of the coordinate system, though the described techniques can also be applied to the general case.

In the next subsection we will briefly introduce some notations that are used in the remainder of this paper.

3.1 Notations

Let $\Omega \subset \mathbb{R}^3$ denote the data domain, which is discretized by a hierarchy of axis-aligned grids $(\Omega^l)_{l=0,1,\ldots,l_{max}}$ with decreasing mesh spacings. The index l numbers the *refinement level*, starting with 0 for the coarsest level. Let the mesh spacing of the coarsest grid be given by $\mathbf{h}^0 = (h_0^0, h_1^0, h_2^0)$. The mesh spacings of the finer grids are recursively defined by $\mathbf{h}^l := (h_0^{l-1}/r, h_1^{l-1}/r, h_2^{l-1}/r)$, where the positive integer r denotes the so-called *refinement factor*. In principle this factor can differ for each direction and each level, but in order to simplify the notation we assume that it is constant.

Further let the vertices of Ω^l be denoted by $\mathbf{x}^l_{ijk}$ The grid cell denoted by $\Omega^l_{ijk} \subseteq \Omega^l$ contains the vertices $\mathbf{x}^l_{i,j,k}, \mathbf{x}^l_{i+1,j,k}, \mathbf{x}^l_{i,j+1,k}, \ldots$. In the AMR approach, cells are either completely refined by cells of the next finer grid , or remain completely unrefined. Each coarse cell can be decomposed into a set of r^3 cells of the next finer discretization

$$\Omega^l_{ijk} = \bigcup_{\hat{i},\hat{j},\hat{k}} \Omega^{l+1}_{\hat{i}\hat{j}\hat{k}} \text{ with } \hat{i} = ri, ri+1, \ldots, ri+r;\ \hat{j},\hat{k} = \ldots,$$

so the cells $\Omega^{l+1}_{\hat{i}\hat{j}\hat{k}}$ provide a refinement of the coarse cell Ω^l_{ijk}. Since AMR grids can be represented as a tree of nested levels, the coarse base grid Ω^0 is also called *root* level in this context.

The m-th *subgrid* of Ω^l will be denoted by Γ^l_m, compare Figure 3. The union of all level l subgrids $\Lambda^l := \bigcup_{m=0}^{n_l} \Gamma^l_m$ is called *refinement level* l or just level l. By construction these levels are nested: $\Lambda^{l+1} \subseteq \Lambda^l \subseteq \Omega^l$. We will denote the whole grid hierarchy, i.e. the union of all refinement levels by $\mathscr{H} := \bigcup_{l=0}^{l_{max}} \Lambda^l$.

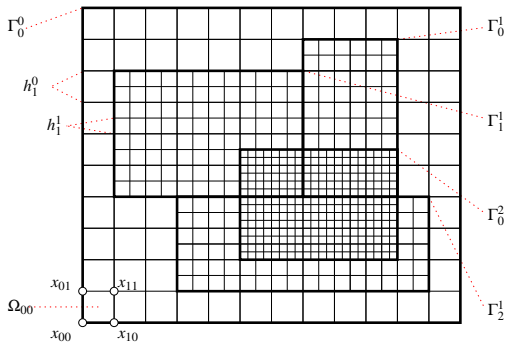

Figure 3: Two-dimensional example of a structured AMR grid. The root level Γ_0^0 is refined by three subgrids $\Gamma_0^1, \Gamma_1^1, \Gamma_2^1$ that generate the refinement level Λ^1. Λ^1 itself is refined by one subgrid Γ_0^2.

3.2 Temporal Refinement Scheme

For numerical solvers of partial differential equations with explicit time-integration, stability conditions demand that the time step size Δt of the scheme corresponds to the mesh size Δx, in the sense that the time step decreases as the mesh spacings decreases. Hence a global time step for all subgrids in an AMR hierarchy would be determined by the cell size of the highest resolved level present in the hierarchy, resulting in a large computational overhead for the coarser levels.

This is the reason for the fact that besides the spatial refinement, AMR schemes for solving partial differential equations additionally perform a refinement in time. That means the spatially refined levels are updated more frequently than the coarser ones. The order in which the levels are advanced in time is shown in Figure 4.

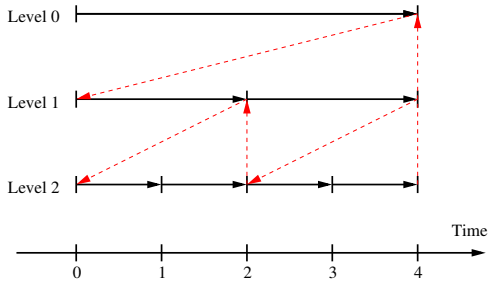

Figure 4: Order of temporal integration of a grid hierarchy with an overall temporal refinement factor of $r_t = 2$.

First, the coarse level Λ^0 is advanced for a large time step. Next the integration routine is recursively called for the refined levels $\Lambda^1, \ldots, \Lambda^{l_{max}}$, and these subgrids are advanced with a decreasing time step size. The integration of the finer levels is followed by the so-called *restriction step*, which updates the coarse grid function by the more accurate values of the finer ones.

In the last step the solution is inspected and the grid structure is adapted based on the local error criterion. This implies that the topology of $\Lambda^l, \ldots, \Lambda^{l_{max}}$ might change after each integration step on a level l. In general the structure of the whole hierarchy (except for the root level) is modified at time steps at which the root level grid is updated.

The time steps of the refined levels need not necessarily be equally distant, but it must be ensured that after an integer number of updates the times of all levels in the hierarchy match up again.

In the following we denote the union of level l subgrids at a certain time by $\Lambda^l(t)$ and the grid hierarchy by $\mathscr{H}(t)$.

4 Generation of Intermediate Grids

The resulting change of the underlying grid structure complicates the interpolation of intermediate time steps during the visualization phase. Consider again the example of Figure 4. For time steps 0 and 4 data at all refinement levels is available, but for time step 1, only the second level contains subgrids.

So in order to visualize this discrete time step the data for level 0 and level 1 must be generated from the stored information. For level 0, data is given for time 0 and 4, whereas for level 1 there is data at time 0 and 2. If for each level the subgrids before and after regridding are available, then interpolation is not problematic, since for each node in t_a^l there exists a counterpart in t_b^l. But as discussed above, often just a fraction of the computed time steps is stored, for example the time steps that correspond to root level updates[1].

In this case it is not obvious how to carry out the interpolation, since in general there are no corresponding grid nodes in this situation. We may state the problem as follows:

> Given sets of subgrids on refinements levels $\Lambda^l(t_a^l), \Lambda^l(t_b^l)$ with $l = 0, 1, ..., l_{max}$ at discrete time steps t_a^l, t_b^l, with potentially different topology, we need to generate a grid hierarchy $\mathscr{H}(t)$, as well as an interpolated grid function $f^l(t)$ for an intermediate time $t \in [\max_l t_a^l, \min_{l'} t_b^{l'}]$. [2]

We will first discuss how the intermediate grid structure is generated. We make the following assumptions, which are usually fulfilled for time-dependent AMR grids:

- The root level structure remains constant for all time steps, and
- the spatial refinement factors between two consecutive levels do not change in time.

In a first step, the refinement levels of the intermediate grid are generated. This is done by merging the subgrids for each level of the key frames:

$$\Lambda^l(t) = \Lambda^l(t_a^l) \cup \Lambda^l(t_b^l).$$

By merging of corresponding levels, in general we loose the subgrid structure present in the key frame hierarchies, as illustrated in Figure 5. The resulting collection of cells on each level could be stored as an unstructured hexahedral grid with explicit connectivity information. But in terms of memory efficiency and performance it is advantageous to reintroduce a structure of disjoint subgrids on these unions of cells, since many rendering algorithms, like for example hardware-accelerated volume rendering, operate more efficient on blocks with implicit connectivity.

So for each $\Lambda^l(t)$ of the intermediate hierarchy $\mathscr{H}(t)$, we require a partition into axis-aligned, non-overlapping rectangular subgrids $\Gamma^l(t_i)$, such that $\Lambda_l(t) \subseteq \bigcup_i \Gamma_i^l(t)$.

This partition could be generated by computing the pairwise intersection of the grids on $\Lambda^l(t_a^l)$ with the grids at $\Lambda^l(t_b^l)$, but this might result in a large number of small, degenerated grids. In a second step one could do some post-processing to merge smaller grids to larger ones.

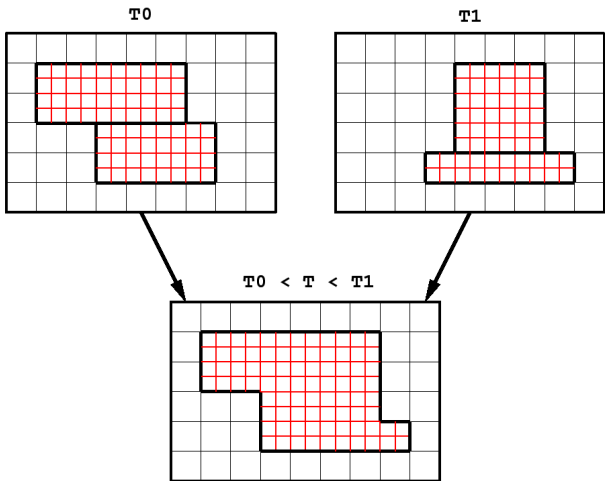

Figure 5: Top: Coarse grid and level 1 subgrids of two key frames. Bottom: Coarse grid and level 1 union of subgrids for intermediate time steps (created by merging corresponding level l subgrids of each key frame).

We decided to utilize a clustering algorithm published by Berger [6]. Besides the fact that it generates efficient partitions in terms of the number of grids, it has the further advantage that the resulting grids are arranged in a kD-tree style. This is beneficial for visualization algorithms like direct volume rendering, since the kD-tree allows an efficient processing of the separate grids in a view-consistent order, see [18]. We will briefly describe the basic ideas of the clustering in the next section, for more information the reader may refer to [6].

4.1 A Clustering Algorithm

Assume that the information about which cells are selected for clustering is encoded by the binary function

$$S^l(i,j,k) = \begin{cases} 1, & \text{if } \Omega_{ijk}^l \text{ is marked for clustering} \\ 0, & \text{otherwise.} \end{cases}$$

In a first step the number of cells that need refinement is computed for each slice perpendicular to the three major coordinate planes and stored in so called signature lists.

A two dimensional example is shown in Figure 6 (a). In the next step exterior zero-entries in these lists are detected and pruned off in order to place a minimal bounding box around the marked cells, as shown in Figure 6 (b). Any interior zero entry in these lists indicates a potential splitting index, i.e. a position at which the given volume is subdivided into two smaller subregions. If all signatures are non-zero, the second derivative of each signature list is computed and the largest inflection point is chosen as the splitting plane, compare Figure 6 (c). This procedure is repeated recursively on the newly created subregions until one of the following halting criteria is satisfied:

- The subregion exceeds the *efficiency* ratio, i.e. the ratio of the number of cells tagged for clustering to its total number of cells is greater than a preselected threshold.
- Further subdivision of the region would result in grid dimensions smaller than some *minimal extension*.

[1] In order to allow a faithful interpolation of the data we assume that the spatial and temporal sampling rate of the data is at least twice the bandwidth of the signal, according to Shannons sampling theorem [33].

[2] The number of key-frames t_i^l required for this depends on the order of the interpolation function that is applied to obtain the associated grid function. In order to ease the notation we assume here that linear interpolation is applied, compare Section 4.3. We further assume that the maximal refinement level is the same for both times, which is no loss of generality, since we can always generate additional levels for one of the time steps by projecting data from the next coarser levels, such that both hierarchies have the same depth.

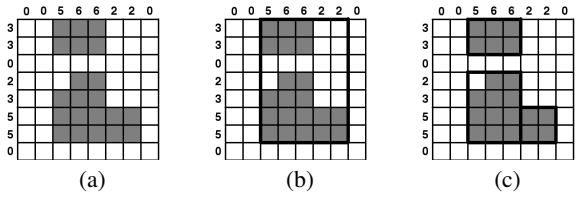

Figure 6: 2D example of the clustering procedure: (a) In a first step the signature lists are computed. (b) Exterior rows and columns with zero entries are pruned off. (c) Interior zero entries and inflection points indicate splitting edges.

4.2 Application of the Clustering Algorithm

In a first step the cells that are contained in the root level $\Lambda^0(t)$ of $\mathscr{H}(t)$ are clustered via this algorithm, yielding a set of axis-aligned subgrids $\Gamma_i^0(t)$. In principle the clustering could also be applied directly for refined levels, but this would result in high computational efforts for computing the signature lists, since the number of the cells contained in the minimal bounding box enclosing the higher resolved levels can grow exponentially.

Thus we perform the clustering procedure recursively on each newly created subgrid rather than for the whole level at once. That is for each subgrid $\Gamma_i^0(t)$ only the cells Ω_{ijk}^1 on the first level of refinement $\Lambda^1(t)$ that are contained in the bounding box of $\Gamma_i^0(t)$ are clustered. This procedure is recursively repeated for each of the subgrids on $\Lambda^1(t)$, until all levels are processed. By this procedure we obtain an AMR hierarchy $\mathscr{H}(t)$ that consists of nested, disjoint structured subgrids.

4.3 Temporal Interpolation of Grid Functions

Next the grid functions $f^l(t)\colon \Lambda^l(t) \to \mathbb{R}$ associated with the intermediate grid hierarchy $\mathscr{H}(t)$, are constructed. Two cases must be distinguished for each cell $\Omega_{ijk}^l \in \Lambda^l(t)$:

$$\Omega_{ijk}^l \in \Lambda^l(t_a^l) \cap \Lambda^l(t_b^l) \quad \text{or} \quad \Omega_{ijk}^l \notin \Lambda^l(t_a^l) \cap \Lambda^l(t_b^l).$$

In the first case the data values associated with $\Omega_{ijk}^l(t)$ can simply be computed from the set of grid functions at t_a^l and t_b^l, see discussion below. In the second case, in at least one of the given levels $\Lambda^l(t_a^l)$ or $\Lambda^l(t_b^l)$ no corresponding cell on the refinement level l exists. In order to obtain a function value for these nodes, we interpolate the grid function on the (already processed) grid on the next coarser level of resolution $\Lambda^{l-1}(t)$. We currently support constant interpolation for cell-centered data and tri-linear interpolation for vertex-centered data.

We employ two different temporal interpolation schemes. The first one is C^0-continuous piecewise linear interpolation. The second one is C^1-continuous cubic Hermite interpolation. So besides the function values for t_s and t_{s+1}, also the first derivative at these time steps contributes to the interpolant

$$\begin{aligned} f_{ijk}^l(t) := f_{ijk}^l(t_a^l)\, H_0^3(t) \quad &+ \quad \left(\frac{d}{dt} f_{ijk}^l(t_a^l)\right) H_1^3(t) + \\ f_{ijk}^l(t_b^l)\, H_2^3(t) \quad &+ \quad \left(\frac{d}{dt} f_{ijk}^l(t_b^l)\right) H_3^3(t). \end{aligned}$$

$\{H_0^3, ..., H_3^3\}$ denote the *cubic Hermite Polynomials*. In case the first derivatives of the grid functions are not available during the visualization phase, we approximate these by the slope of the quadratic polynomials through t_a, resp. t_b and the preceding, resp. succeeding time step.

This implies that we require the grid function values at four successive time steps in order to obtain an approximation of the first derivatives at t_a and t_b.

In a last step the data on the coarser levels must be updated by the filtered data from the higher resolved levels. This is necessary since the sampling rate on the refined levels is higher and therefore data on these levels can resolve features, that are not captured correctly in the interpolation on the corresponding coarser representation of this region. Currently the grid generation process is carried out on-the-fly, but parts of it could be done in a preprocessing step.

5 Remote Data Access

The presented algorithms were implemented using partial file access to a local file. In a second step the visualization application was distributed.

As discussed in Section 2, image-streaming does not deliver sufficiently high frame-rates. Remote file access, as e.g. discussed in [28], utilizes only a small fraction of the available network bandwidth; especially if latency becomes large.

Our distribution decision was determined by two major goals: simplicity and minimization of the number of network round-trips. We decided to distribute the application based on the remote procedure call (RPC) paradigm, i.e. replacing local function calls by synchronous request-response calls to a server.

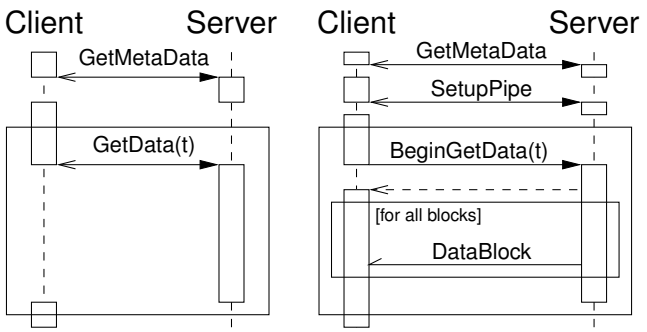

Figure 7: Sequence diagrams for the simple (left) and advanced (right) client/server. The simple server returns the intermediate grid hierarchy and all interpolated data in the response to GetData(t). The advanced server returns only the intermediate grid hierarchy as a response to BeginGetData(t). Interpolated data samples are delivered as asynchronous one-way DataBlock messages.

In a following step, described below, we reduced the size of the initial server response by delivering large parts of the response outside the RPC message as a stream of binary data blocks. We use the gSOAP toolkit [37] to generate stubs for client/server SOAP calls. SOAP is an XML-based RPC standard using HTTP as its transport layer [35]. It is commonly associated with 'web services.'

In order to visualize remote AMR data, two major options are available for distributing the application: to carry out the grid generation and interpolation procedure on the remote side and transfer the resulting intermediate hierarchy, or to transfer the keyframes of the data first, and perform the interpolation on the local visualization client.

This decision depends on whether it is more important to minimize bandwidth usage or interpolation time. Based on the fact that in our setting, it takes more time to transfer data than to execute the interpolation we decide to minimize overall data transfer over the network.

When a user inspects a new data set for the first time and wants to get an impression of the overall temporal evolution of the data typically only a small number of time steps compared to the total

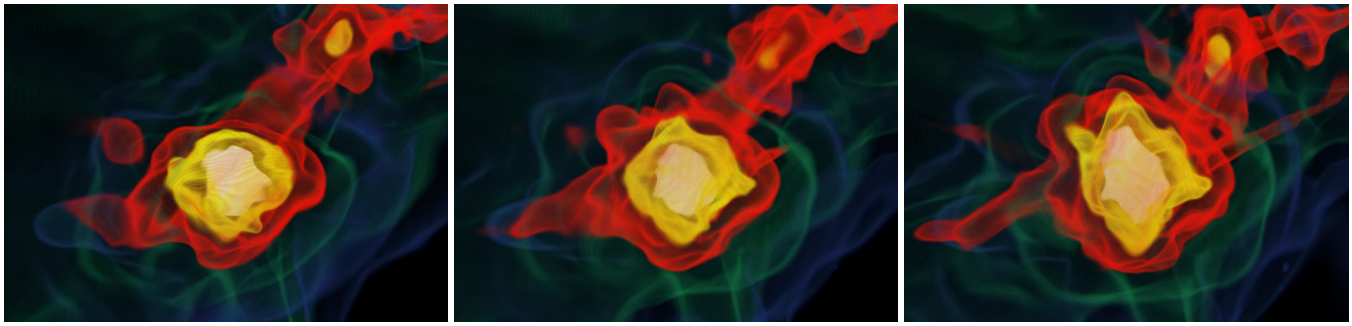

Figure 8: Two key frames and an interpolated intermediate frame generated from data set I, a galaxy-formation simulation, via texture-based volume rendering.

number of time steps in the file is requested and the distance between time steps is larger than the distance between two root level updates. In this case the bandwidth requirements are smaller if the interpolation is carried out on the remote side.

In a first request, the client asks for meta data about the data set, like the bounding box, data type, and available time range. The remaining session consists of requests for hierarchies at certain times. The server performs all required processing and delivers a grid with interpolated data for the requested time.

In the simple version of the server all the interpolated data samples are included in the SOAP response (see Figure 7). This results in large SOAP messages that may take long to be returned to the client. The client blocks until all data for the requested time step has been received. Most of the response is made up of the binary samples of the grid function, whereas the description of the grid hierarchy requires only a small percentage of the message.

In our current version, the sample data is replaced by an identifier in the SOAP message and delivery is deferred until after the RPC returned (see Figure 7). Since the RPC response only contains the description of the hierarchy the remote call will return much faster. The blocks are then delivered through a dedicated pipe transporting one-way messages [27].

While we are currently using TCP, this separation will also allow us to easily add new and useful protocols like SABUL/UDT [12] or special protocols for dedicated networks, like Reliable Blast UDP [14] or TCPXX [32].

The client displays the AMR hierarchy right after the RPC returned. Data samples are filled in asynchronously as they arrive. Regions for which higher resolution data is expected can be highlighted as the full AMR hierarchy is available. This may be useful information to warn the user from drawing conclusions based on low resolution data. Interpolated data on these levels may be misleading (as discussed at the end of Section 4 At any time, the client can initiate a request for another point in time. Such a request triggers on the server side a cancel of the transfer of pending blocks from the last request. Thus, a user can change the requested time step before the data has been completely transferred and network resources are not wasted.

In both versions of the server, only one network round-trip is required per request and delivery of a hierarchy at a certain time.[3] As a consequence, the network bandwidth can be fully utilized during delivery of the hierarchy.

6 Results

The algorithms have been implemented as extensions to Amira [36], an object-oriented, expandable 3D data visualization system developed at ZIB. We applied the algorithm to different time-dependent AMR data sets.

We tested the performance of the grid generation and interpolation operation for two local files stored on a SGI Onyx3 on a single 500 MHz MIPS R14000 processor. Data set I is a result from a cosmological simulation of the formation of stars in the early universe with a root grid resolution of 128^3 cells, 8 levels of refinement and about 2000 grids per time step. Data set II depicts a supernova explosion with 8 levels of refinement and about 1600 grids per time step. We took 10 data dumps and generated 8 intermediate frames for each pair using both linear and Hermite interpolation, with estimated first derivatives, see Section 4.3. Figure 8 and 9 show some volume rendered images of the sequences.

Details about performance and memory requirements are shown in Figure 10. Since four key frames need to be merged for the Hermite interpolation the amount of additional memory requirements and the times for grid generation were highest. The space increase introduced by the merging procedure was about 7 % and 15 % for the examples. The middle column depicts the times for grid generation. Again it was highest for the Hermite interpolation, but it was less than 7 seconds even for the 2000 grid data set. This amount of time is acceptable for on-the-fly generation during the visualization phase.

We performed network measurements between ZIB, Berlin, Germany and LSU, Baton Rouge, USA. The machines we used were a 3.2 GHz single-processor P4, 1.5 GByte RAM, 160 GB 7200 rpm HDD at LSU as the server and a Dual 2.0 GHz AMD Opteron 246 at ZIB as the client. The network round-trip time was approximately 150 ms. An untuned TCP connection delivered between 5–6 MByte per minute (measured with Iperf [15]).

We used one scalar quantity stored in a HDF5 file of 22 GBytes size. The AMR hierarchy, interpolated at a certain time, consists of roughly 5.5 MBytes of data stored on approximately 600 subgrids.

We measured two values that we consider useful for giving an impression on how our system performs. After selecting a time step, it took 1–3 seconds to send the request, load and process the data on the server, and send the hierarchy description back to the client.[4] The latency, until the hierarchy is returned to the client, is mainly bound by loading and processing the data on the server. Local access is not notably faster. Giving detailed numbers is difficult as processing times vary depending on caching of the file system and the HDF5 library.

After receiving the RPC response, the client starts displaying the hierarchy and updates the display as the sample data arrive. All binary samples were fully received after 60–75 seconds. Compressing the data sent through the binary channel, using standard zlib compression, resulted in only a 5% decrease in total data size. Our

[3] The use of HTTP Keep-Alive avoids per-request connection setup.

[4] We used gSOAP's integrated zlib compression of XML messages.

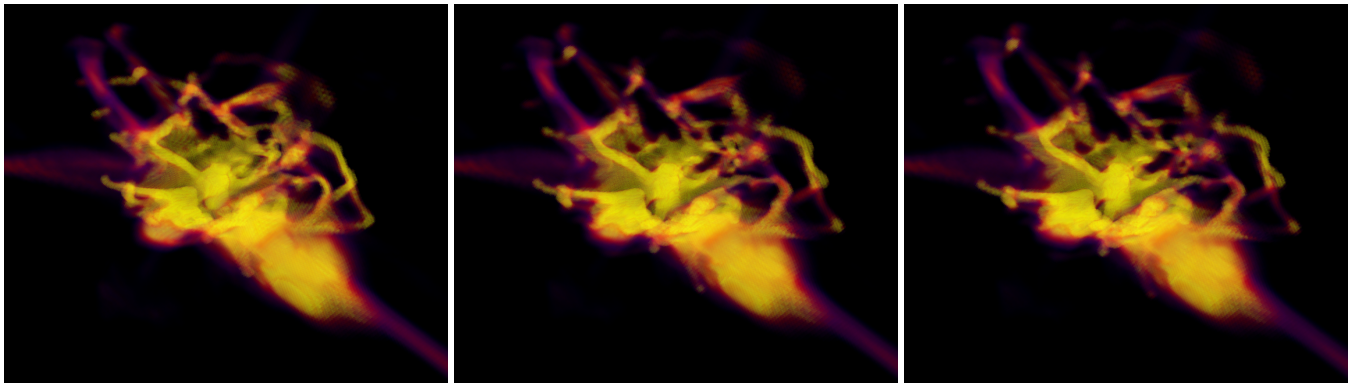

Figure 9: Three interpolated frames from data set II, a simulation describing a supernova explosion visualized by texture-based volume rendering.

	increase	grid generation	interpolation
Data I (linear)	7%	3.4 sec	0.2 sec
Data I (Hermite)	12%	6.6 sec	0.8 sec
Data II (linear)	11%	2.2 sec	0.1 sec
Data II (Hermite)	15%	3.5 sec	0.3 sec

Figure 10: The first column denotes the increase in the number of cells for the intermediate time steps relative to the given key frames. The second column states the times for generating the intermediate grid. The right column gives the time for interpolation of the intermediate grid function.

application uses the full available bandwidth that was measured by Iperf.

7 Conclusions

In this paper we addressed the problem of visualizing remote, time-dependent data defined on AMR grids. In order to handle the problem of varying grid topology during time evolution, intermediate grid hierarchies are generated by merging the grids on the corresponding refinement levels. In a second step a nested grid structure is induced, employing a clustering algorithm. Finally the grid functions can be mapped to the intermediate hierarchy and interpolated using conventional schemes like linear or Hermite interpolation.

We use remote procedure calls (RPC), implemented as SOAP messages, to distribute the application using a client/server model. The RPC paradigm allows to easily distribute an application at any function call. Choosing a high-level call to minimize the number of network round-trips is crucial to efficiently utilize a high-latency network. In our case round-trips only occur when the requested time is changed. By separating the bulk data transfer from the AMR hierarchy description and transferring sample data in an asynchronous binary stream, we are able to achieve low response times in the RPCs; the separated binary stream utilized the full network bandwidth of a TCP connection.

We consider it sufficient for an interactive application to deliver a first response to parameter changes, for example selection of a time step, within seconds, and a complete result within minutes, while always sustaining a high update rate of the rendered image when changing the view point. Under these assumptions, the size of data that can be displayed by our application is either limited by the network bandwidth, the amount of data transferred within minutes, or by the capability of local hardware to handle the data. With increasing data size, or with low-capacity networks image streaming or image based rendering methods may become the only available option for visualization. Interactivity is then restricted by the network latency. But as long as the network is fast enough and local resources are capable of handling the data, we believe that using remote data access is the best way to build a remote visualization application if interactivity is a major concern.

8 Future Work

For larger hierarchies it might be beneficial to carry out parts of the grid generation in a preprocessing step. It is unfeasible in general to store a merged grid for each time data is updated on the highest resolved level. Alternatively one could generate two sets of grids per time step for a each refinement level, such that for each intermediate time the interpolation of data can be carried out on corresponding subgrids. The first part could be done in a preprocessing step, whereas the second one has to be carried out on the fly.

The binary data transport layer could be replaced by network protocols that are optimized for high bandwidth over large distances or for optical networks. It could also be extended to receive data blocks from more than one source, which may be a scenario in Grid or Cluster Computing. We plan to investigate these points in the future. We further plan to allow the user to reduce the amount of transmitted data by requesting only those subgrids that are contained in some region of interest.

9 Acknowledgments

We thank Tom Abel (Stanford University), Stuart Levy (National Center for Supercomputing Applications) and Greg Bryan (University of Oxford) for providing the cosmological data sets and for fruitful discussions. The black hole collision data set was computed by Ryoji Takahashi (Louisiana State University). Special thanks to Charlotte Fruge (Louisiana State University) for proof-reading the manuscript. This work was supported by the Center for Computation Technology at LSU.

References

[1] T. Abel, G. Bryan, and M. L. Norman. The formation of the first star in the universe. *Science*, 295, issue 5552:93–98, 2002.

[2] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data management and transfer in high performance computational grid environments. In *Parallel Computing Journal,28(5)*, pages 749–771, 2002.

[3] A. S. Almgren, J. B. Bell, P. Colella, and T. Marthaler. A Cartesian grid projection method for the incompressible Euler equations in complex geometries. *SIAM Journal on Scientific Computing*, 18(5):1289–1309, 1997.

[4] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Resource Broker. *Proc. CASCON'98, Toronto, Canada*, 1998.

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

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

[7] W. Bethel, B. Tierney, J. lee, D. Gunter, and S. Lau. Using high-speed wans and network data caches to enable remote and distributed visualization. In *Proc. IEEE Supercomputing '00*, page 28, Washington, DC, USA, 2000. IEEE Computer Society.

[8] M. D. Beynon, R. Ferreira, T. Kurc, A. Sussman, and J. Saltz. DataCutter: Middleware for filtering very large scientific datasets on archival storage systems. In *Proc. Mass Storage Systems*, pages 119–133. IEEE Computer Society Press, Mar. 2000.

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

[10] I. Foster, D. Kohr, Jr., R. Krishnaiyer, and J. Mogill. Remote I/O: Fast access to distant storage. In *Proceedings of the Fifth Workshop on Input/Output in Parallel and Distributed Systems*, pages 14–25, San Jose, CA, 1997. ACM Press.

[11] J. Gallagher, N. Potter, T. Sgouros, S. Hankin, and G. Flierl. The data access protocol—DAP 2.0, 2004.

[12] Y. Gu, X. Hong, and R. L. Grossman. Experiences in design and implementation of a high performance transport protocol. In *Proceedings of the ACM/IEEE SC2004 Conference (SC'04)*, page 22, November 2004.

[13] T. Happe, K. Polthier, M. Rumpf, and M. Wierse. Visualizing data from time-dependent adaptive simulations. In *Proceedings of the Workshop on 'Visualization - Dynamics and Complexity'*, 1995.

[14] E. He, J. Alimohideen, J. Eliason, N. K. Krishnaprasad, J. Leigh, O. Yu, and T. A. DeFanti. Quanta: a toolkit for high performance data delivery over photonic networks. *Future Gener. Comput. Syst.*, 19(6):919–933, 2003.

[15] Iperf documentation. http://dast.nlanr.net/Projects/Iperf/.

[16] T. J. Jankun-Kelly, O. Kreylos, J. M. Shalf, K.-L. Ma, B. Hamann, K. I. Joy, and E. W. Bethel. Deploying web-based visual exploration tools on the grid. *IEEE Computer Graphics and Applications*, 23(2):40–50, 2003.

[17] R. Kähler, D. Cox, R. Patterson, S. Levy, H.-C. Hege, and T. Abel. Rendering the First Star in the Universe - A Case Study. In R. Moorhead, M. Gross, and K. I. Joy, editors, *Proceedings of IEEE Visualization*, pages 537–540. IEEE Computer Society Press, 2002.

[18] R. Kähler and H.-C. Hege. Texure-based Volume Rendering of Adaptive Mesh Refinement Data. *The Visual Computer*, 18(8):481–492, 2002.

[19] R. Kähler, M. Simon, and H.-C. Hege. Fast Volume Rendering of Sparse Datasets using Adaptive Mesh Refinement. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):341–351, 2003.

[20] T. Kurc, Ü. Çatalyürek, C. Chang, A. Sussman, and J. Saltz. Visualization of large data sets with the active data repository. *IEEE Comput. Graph. Appl.*, 21(4):24–33, 2001.

[21] J. Li, W.-K. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale. Parallel netCDF: A scientific high-performance I/O interface. In *Proc. Supercomputing Conference*, Phoenix, Arizona, Nov. 2003.

[22] S. L. Liebling. The singularity threshold of the nonlinear sigma model using 3D adaptive mesh refinement. *Class.Quant.Grav.*, 21(3995), 2004.

[23] K. Mueller, N. Shareef, K. Huang, and R. Crawfis. Ibr-assisted volume rendering. In *Proc. IEEE Visualization 2000, Late Braking Hot Topics*, page 45, 2000.

[24] 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.

[25] S. Park, C. L. Bajaj, and V. Siddavanahalli. Case study: Interactive rendering of adaptive mesh refinement data. In R. Moorhead, M. Gross, and K. I. Joy, editors, *Proceedings of IEEE Visualization*, pages 521–524. IEEE Computer Society Press, 2002.

[26] K. Polthier and M. Rumpf. A concept for time-dependent processes. In M. Göbel, H. Müller, and B. Urban, editors, *Visualization in Scientific Computing*, pages 137–153. Springer Verlag, Berlin, Heidelberg, New York, 1994.

[27] S. Prohaska and A. Hutanu. Remote data access for interactive visualization. In *13th Annual Mardi Gras Conference: Frontiers of Grid Applications and Technologies*, pages 17–22, 2005.

[28] S. Prohaska, A. Hutanu, R. Kähler, and H.-C. Hege. Interactive exploration of large remote micro-CT scans. In *Proc. IEEE Visualization '04*, pages 345–352. IEEE Computer Society, 2004.

[29] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, Jan./Feb. 1998.

[30] T. Schmidt and R. Rühle. On-line visualization of arbitrary unstructured, adaptive grids. In *Proceedings of Sixth Eurographics Workshop on Visualization in Scientific Computing*, pages 25–34. Springer, 1995.

[31] E. Schnetter, S. H. Hawley, and I. Hawke. Evolutions in 3D numerical relativity using fixed mesh refinement. *Class. Quantum Grav.*, 21(6):1465–1488, 2004.

[32] J. Shalf and W. Bethel. Cactus and Visapult: An Ultra-High Performance Grid-Distributed Visualization Architecture Using Connectionless Protocols. In *IEEE Computer Graphics and Applications*, volume 23(2), pages 51–59, March/April 2003.

[33] C. E. Shannon. Communication in the presence of noise. *Proc. Institute of Radio Engineers, Vol. 37, No. 1*, pages 10–21, 1949.

[34] Silicon Graphics, Inc., 1600 Amphitheatre Pkwy, Mountain View, CA 94043, United States. *OpenGL Vizserver 3.1 White Paper - Application-Transparent Remote Interactive Visualization and Collaboration*, Apr. 2003.

[35] SOAP specifications. http://www.w3.org/TR/soap/.

[36] D. Stalling, M. Westerhoff, and H.-C. Hege. Amira: A highly interactive system for visual data analysis. In C. D. Hansen and C. R. Johnson, editors, *The Visualization Handbook*, pages 749–767. Elsevier, 2005.

[37] R. van Engelen and K. Gallivan. The gSOAP toolkit for web services and peer-to-peer computing networks. In *CCGRID*, pages 128–135, 2002.

[38] 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.

[39] 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 *Proceedings of Vision, Modeling, and Visualization 2001*, pages 121–128, Stuttgart, Germany, 2001.

[40] G. H. Weber, M. Oehler, O. Kreylos, J. M. Shalf, W. Bethel, B. Hamann, and G. Scheuermann. Parallel cell projection rendering of adaptive mesh refinement data. In A. Koning, R. Machiraju, and C. T. Silva, editors, *Proceeding of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, pages 51–60, Los Alamitos, California, 2003. IEEE, IEEE Computer Society Press.