Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

RALF KÄHLER HANS-CHRISTIAN HEGE

Visualization of Time-Dependent Adaptive Mesh Refinement Data

ZIB-Report 03-16 (June 2003)

Visualization of Time-Dependent Adaptive Mesh Refinement Data

Ralf Kähler [*†‡] Hans-Christian Hege[*]

Abstract

Analysis of phenomena that simultaneously occur on quite different spatial and temporal scales require adaptive, hierarchical schemes to reduce computational and storage demands. For data represented as grid functions, the key are adaptive, hierarchical, time-dependent grids that resolve spatio-temporal details without too much redundancy. Here, so-called AMR grids gain increasing popularity. For visualization and feature identification/tracking, the underlying continuous function has to be faithfully reconstructed by spatial and temporal interpolation. Well designed interpolation methods yield better results and help to reduce the amount of data to be stored.

We address the problem of temporal interpolation of AMR grid data, e.g. for creation of smooth animations or feature tracking. Intermediate grid hierarchies are generated by merging the cells on all refinement levels that are present in the key frames considered. Utilizing a clustering algorithm a structure of nested grids is induced on the resulting collection of cells. The grid functions are mapped to the intermediate hierarchy, thus allowing application of appropriate interpolation techniques.

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: AMR, non-conforming hexahedral grids, multi-resolution techniques

[*]Zuse Institute Berlin (ZIB)
[†]MPI for Gravitational Physics (AEI)
[‡]supported by BMBF/DFN, project GriKSL TK 602

1 Introduction

Multi-scale phenomena are abundant in many application fields like material science, fluid dynamics, geophysics, meteorology and astrophysics. Representing and numerically simulating such processes is a challenging task since quite different scales have to be resolved, requiring enormous amounts of storage and computational power. Usually hierarchical representations are used, like wavelet decompositions or hierarchical grids. An important aspect is adaptivity, i.e. local adjustment of the spatio-temporal resolution to the details to be resolved. A standard representation therefore are hierarchical, locally refined grids.

So-called AMR grids, a specific variant of nested structured grids, gain increasing popularity. This grid representation of functions was originally developed by Berger and Oliger in the context of a numerical multilevel technique for solving hyperbolic partial differential equations [7]. A variant of this approach, called *structured* Adaptive Mesh Refinement, is applied in many domains like hydrodynamics [6], meteorology [3] and in particular astrophysics [10, 2]. AMR grids are used also on their own: data from experimental sources like 3D image data, e.g. from confocal microscopy, can be efficiently represented using AMR grids [16].

Locally refined hierarchical grids become imperative, if *time-dependent* 3D data are considered. Hence an increasing number of scientists is in need of appropriate analysis techniques that help to interpret hierarchical time-dependent 3D grid data. For analysis purposes, like animation, feature identification, or feature tracking, the underlying non-discrete time-dependent function has to be faithfully reconstructed from the grid function. This is done by spatial and temporal interpolation.

Appropriate interpolation methods may improve the results of data analysis greatly. Furthermore, they help to further reduce the amount of grid data to be stored and handled. For instance in large numerical simulations one may store less time steps and nevertheless be able to create smooth animations that display the underlying process without discontinuities or cracks. Even if numerical methods are available that provide *dense* output by maintaining and evaluating internally some interpolants, one would *not* store all these data due to the immense storage requirements. Only information should be stored that can not be recovered by spatio-temporal interpolation methods.

In this article we describe an approach that allows the generation of dense output for time-dependent AMR data defined on axis-aligned, structured, hexahedral meshes. The algorithm is fast and therefore enables a 'on-the-fly' application during the visualization session. Although the algorithms are presented in the context of AMR data, they also apply to axis-aligned, unstructured non-conforming hexahedral meshes, that are used in finite element (FEM) simulations. This mesh type is supported e.g. by the numerical libraries deal.II [4] and UG [5].

2 Related Work

The number of papers dealing with rendering methods for AMR data has increased in the last couple of years. A back-to-front cell-sorting algorithm for AMR hierarchies utilized for volume rendering was presented by Max [18]. Norman et al. describe their approach of resampling AMR data to uniform as well as unstructured grids [19]. Ma presented a parallel volume renderer for AMR data [17]. Weber et al. proposed an approach for crack-free isosurface extraction as well as a software and hardware accelerated cell-projection algorithm for AMR data [25, 24, 26]. Kähler et al. presented an approach for accelerated texture based volume rendering of large, sparse datasets by representing non-transparent regions using AMR data structures [16] and a hardware-supported, texture based volume rendering algorithm for AMR data that directly employs the hierarchical structure of this data type [15, 14]. Park et. al. presented a splatting approach in [20]. Techniques for rendering remote AMR data are described in [13].

Visualization methods for time-dependent adaptive simulations have been presented by Polthier et. al. [21], Happe et. al. [12] and Schmidt et. al. [22]. These approaches have in common that they assume the existence of two unstructured grids and associated grid functions at each time-step where 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.

In principle this approach could also be applied in the case of AMR data. But for realistic AMR simulations, which often contain dozens of refinement levels, and typically evolve several scalar and vector quantities, this would result in huge amounts of data between two root level updates. This is because the update frequency usually doubles between two consecutive levels of refinement, i.e. increases exponentially. AMR simulations code therefore often store data only for time-steps that correspond to root level updates. Besides this storage problem, it is not clear how to proceed for higher order interpolation in time that require more than keyframes.

We therefore decided to create intermediate grid hierarchies in order to connect keyframes hierarchies. An imaginable approach is to identify corresponding subgrids present in the keyframes, for example by utilizing feature tracking algorithms like presented by Silver et. al. [23], and then interpolate their position and sizes for the intermediate steps. But this would in general result in overlapping grids on the same levels and it is also not clear how to proceed in case a subgrid has no corresponding partner on the next frame. Instead we followed up with an approach that

- generates an intermediate grid hierarchy by merging the cells on each refinement levels that are present in the keyframes,
- uses a clustering algorithm to induce a nested grid structure on the resulting collection a cells,
- projects the grid functions of each keyframe to such an 'merged' grid hierarchy and finally
- generates the intermediate grid functions by interpolating between each set of corresponding data samples on these merged hierarchies.

3 AMR Data Structure

Let $\Omega \subset \mathbb{R}^3$ denote the data domain. In general it may be composed of a set of non-overlapping, aligned rectangular domains. In order to simplify the notation we assume that Ω consists of just one rectangular region.

3.1 Globally Refinement

Let Ω be discretized by a hierarchy of axis-aligned, structured rectilinear grids $(\Omega^l)_{l=0,1,...,l_{max}}$ with decreasing mesh size. Index l denotes the *refinement level*, starting with 0 for the coarsest level. The size of cells on the coarsest grid is given by $\mathbf{h}^0 = (h_0^0, h_1^0, h_2^0)$, and the cell sizes on finer grids are recursively defined by $\mathbf{h}^l := (h_0^{l-1}/r_0, h_1^{l-1}/r_1, h_2^{l-1}/r_2)$, where $\mathbf{r} = (r_0, r_1, r_2), r_i \in \mathbb{N}^+$ denotes the so-called refinement factor. In principle the refinement factor may be different for each refinement level, but in order to ease notation we assume that it is constant. Let $n_i + 1$ the number of vertices along the i-th coordinate axis of the coarsest discretization Ω^0. The vertices of Ω^l are given by

$$\mathbf{x}_{ijk}^l := (ih_0^l, jh_1^l, kh_2^l)$$

with $i = 0, 1, ..., n_0 r_0^l$ and similar for j, k. Note that l in the expression r_0^l is an exponent, in contrast to the rest of this article. Thus the position of a vertex $\mathbf{x}_{ijk}^l \in \Omega^l$ corresponds to the vertex $\mathbf{x}_{ri,rj,rk}^{l+1} \in \Omega^{l+1}$. The grid cells $\Omega_{ijk}^l \subseteq \Omega^l$ are defined by

$$\Omega_{ijk}^l := \left\{ \mathbf{x} \in \Omega \mid \mathbf{x} = \mathbf{x}_{ijk}^l + \sum_{m=0}^{2} \alpha_m \mathbf{e}^m,\ \alpha_m \in [0, h_m^l] \right\},$$

where $\mathbf{e}^0 = (1,0,0), \mathbf{e}^1 = (0,1,0), \mathbf{e}^2 = (0,0,1)$ denote the standard basis vectors in $\mathbb{R}^3$. Each coarse cell can be decomposed into a set of cells of the next finer discretization

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

and similar for $\hat{j}, \hat{k}$. The cells $\Omega^{l+1}_{\hat{i}\hat{j}\hat{k}}$ are called a refinement of the coarse cell Ω^l_{ijk}.

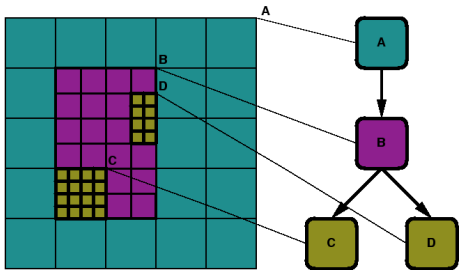

Figure 1: 2D example of an grid hierarchy H with refinement factor $\mathbf{r} = (2,2)$. The root grid Γ^0 is partially refined by one subgrid Γ^1_0, which is further refined by 2 subgrids Γ^2_0 and Γ^2_1.

3.2 Local Spatial Refinement

In the AMR approach the whole computational domain is covered by a coarse grid $\Gamma^0 := \Omega^0$, usually called *root grid* since the whole hierarchy is represented by a tree. The equations are initially approximated on this coarse grid and the coarse solution is inspected by an error estimator detecting cells that require higher resolution. These cells are clustered into disjoint, axis-aligned rectangular regions, which define new *subgrids*, consisting of cells of the next finer discretization Ω^1. Unlike in the FEM approach these subgrids do not replace, but overlap the coarser cells, giving rise to a level-of-detail representation of the interesting parts of the computational domain. Then the equations are evolved on the finer subgrids and this refinement procedure recursively continues until all cells fulfill the simulation specific error criteria. Notice that cells are either totally refined by cells of the next finer grid, or remain totally unrefined, and that adjacent cells may differ by more than one refinement level. Let the m-th subgrid on Ω^l be denoted by

$$\Gamma^l_m = \left\{ \Omega^l_{ijk} \subseteq \Omega^l \mid i = p^m_0, ..., p^m_0 + n^m_0 \text{ and similar for } j,k \right\},$$

where $\mathbf{p}^m$ is the integer offset vector of this subgrid, and n_i^m defines the number of cells per coordinate axis i. The union of level l subgrids is denoted as $\Lambda^l := \bigcup_{m=0}^{n_l} \Gamma_m^l$. By construction these unions are nested, i. e. $\Lambda^{l+1} \subseteq \Lambda^l \subseteq \Omega^l$. The grid hierarchy H is defined as the union of all levels $H := \bigcup_{l=0}^{l_{max}} \Lambda^l$. Fig. 1 shows a 2D example.

Associated with each refinement level are one or more grid functions. We restrict the discussion to real-valued scalar functions in the following, since the generalization of the following to complex or vector-valued functions is straightforward. The functions are usually defined per vertex (*vertex centered*)

$$\mathcal{F}^l : \Lambda^l \to \mathbb{R} \quad with \quad \mathbf{x}_{ijk}^l \in \Lambda^l \mapsto \mathcal{F}^l(\mathbf{x}_{ijk}^l) = f_{ijk}^l,$$

or per cell (*cell-centered*)

$$\overline{\mathcal{F}}^l : \Lambda^l \to \mathbb{R} \quad with \quad \Omega_{ijk}^l \in \Lambda^l \mapsto \overline{\mathcal{F}}^l(\Omega_{ijk}^l) = \overline{f}_{ijk}^l.$$

If not explicitly distinguished, $\mathcal{F}^l$ will be used to denote both cases in the following.

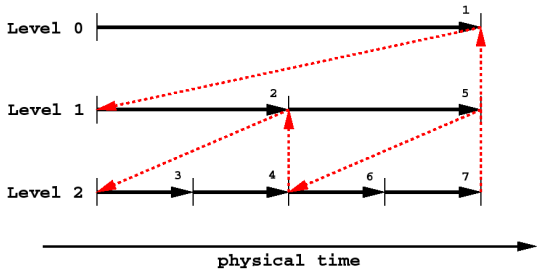

Figure 2: Order of temporal evolution of a grid hierarchy with an overall temporal refinement factor of $r_t = 2$.

3.3 Temporal Refinement

Besides the refinement in space, adaptive numerical schemes usually also perform a refinement in time. This means that the spatially refined levels are updated more frequently. A temporal refinement factor $r_t \in \mathbb{N}^+$ between a pair of two consecutive refinement levels $(\Lambda^l, \Lambda^{l+1})$ indicates that Λ^l is evolved one large step Δt_l, and next Λ^{l+1} is evolved r_t times with step sizes of $\Delta t / r_t$, see Figure 2. In the following we denote by $\Lambda^{l,t}$ and H^t the union of level l subgrids and the grid hierarchy respectively at time t.

In general the time steps of the refined levels do not have to be equally distant, but it has to be ensured that after an integer number of updates the coarse and the fine level match up

again. After the coarse level is evolved according to its step size, the spatial refinement and regridding procedure described above is carried out on the next finer levels. This implies that the topology of the grid hierarchy usually changes with respect to time. In general the structure of the whole grid hierarchy (except for the root grid) may change after a time step corresponding to the coarsest level.

4 Grid Generation

It is this change of the underlying grid structure that complicates the interpolation of intermediate time steps during the visualization phase. We can propose the problem as follows

> Given grid hierarchies and associated grid functions $(H^{t_m}, \mathcal{F}^{l,t_m}_{l=0,...,n_{t_m}})$ at discrete time steps $t_0, t_1, ..., t_{max}$, with different topology in general, generate intermediate grid hierarchies H^t and interpolated grid functions $\mathcal{F}^{l,t}$ for $t \in]t_i, t_{i+1}[$.

We will now describe the construction of the intermediate grid hierarchies. The number of keyframes required for this depends on the order of the interpolation function that is applied to obtain the associated grid function, see Section 5. In the following we make to assumptions which are usually fulfilled by the numerical schemes:

- the root grid structure $\Lambda^{0,t}$ remains constant for all time-steps,
- the spatial refinement factors between two consecutive unions of subgrids $(\Lambda^{l,t}, \Lambda^{l+1,t})$ do not change with respect to time.

The first assumption is not essential and might be omitted as long as it is guaranteed that the cells on the coarsest discretizations Ω^{0,t_m} at the given time steps are not shifted or rotated with respect to each other. The intermediate grid hierarchy for some time step is generated by merging the level l subgrids for each refinement level in the keyframe hierarchies:

$$\begin{aligned} H^t &= H^{t_m} \cup H^{t_{m+1}} \cup ... \cup H^{t_{m+n}} \\ &= \bigcup_{l=0}^{\bar{l}_{max}} \left(\Lambda^{l,t_m} \cup \Lambda^{l,t_{m+1}} \cup ... \cup \Lambda^{l,t_{m+n}} \right) \end{aligned}$$

where $\bar{l}_{max}$ denotes the maximum number of refinement levels present in the set of time-steps considered. However, by merging the corresponding levels, in general we loose the structure of disjoint subgrids present in the keyframe hierarchies, as illustrated in Figure 3.

The resulting intermediate unions of level l subgrids could be stored with explicit connectivity information. But in terms of memory efficiency and performance it is advantageous

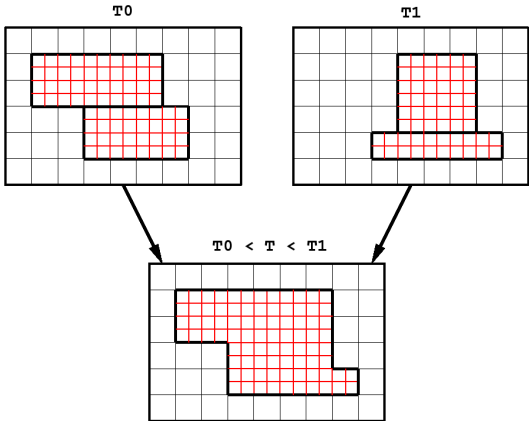

Figure 3: Top: Coarse grid and level 1 subgrids of two keyframes. Bottom: Coarse grid and level 1 union of subgrids for intermediate time-steps (in general created by merging corresponding level l subgrids of each keyframe).

to also subdivide these unions of level l subgrids into sets of disjoint rectangular subgrids, since the rendering algorithms work faster on blocks with implicitly given (trivial) connectivity. So for each $\Lambda^{l,t}$ we demand a partition into axis-aligned, non-overlapping rectangular subgrids $(\Gamma_i^{l,t})_{i=0,,,k_l}$, such that $\Lambda_{l,t} \subseteq \bigcup_{i=0}^{k_l} \Gamma_i^{l,t}$.

4.1 The Clustering Algorithm

An efficient and fast algorithm for clustering cells into axis-aligned regions was suggested by Berger et. al. [8], adopting signature-based methods used in computer vision and pattern recognition. We will describe the basic ideas briefly in this section. For more details the reader might refer to [8].

Assume that the information about which cells of subgrid Γ_m^l shall be clustered is encoded by the binary function defined on the index domain of Γ_m^l:

$$S : [p_0^m, .., p_0^m + n_0^m] \times [p_1^m, ..] \times [p_2^m, ..] \rightarrow \{0, 1\}$$

with $S(i, j, k) = 1$ if Ω_{ijk}^l is marked for clustering. For each slice perpendicular to the i_j, i_k and j_k planes the number of cells that need refinement is computed and stored in so called signature lists. For example the entry for slice number i parallel to the j_k plane is

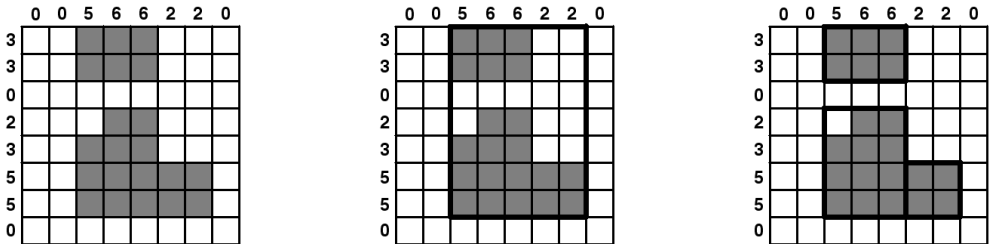

Figure 4: *2D example of the clustering procedure: (a) In a first step signature lists are computed. (b) Exterior rows and columns with zero entries are pruned off. (c) Interior zero entries and inflection points indicate splitting edges.*

given by

$$S_{j\_k}(i) = \sum_{j=p_1^m}^{(p_1^m+n_1^m)} \sum_{k=p_2^m}^{(p_2^m+n_2^m)} S(i,j,k)$$

and similarly for the two other orientations. A 2D example is shown in Figure 4 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 (Figure 4 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 discrete Laplacian second derivative

$$\Delta_{j\_k}(i) = S_{j\_k}(i+1) - 2S_{j\_k}(i) - S_{j\_k}(i-1),$$

and similar for $\Delta_{i\_k}(j)$, $\Delta_{i\_j}(k)$, of each signature list is computed and the biggest inflection point is taken as the splitting plane (see Figure 4 c). This procedure is repeated recursively on the newly created subregions until one of the following halting criteria is satisfied:

- The subregion exceeds some *efficiency* ratio, i. e. the ratio of the number of its 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.*

Notice that according to these criteria the clusters usually contain a number of cells that are not marked, in order to keep the number of created subgrids low. Usually an efficiency

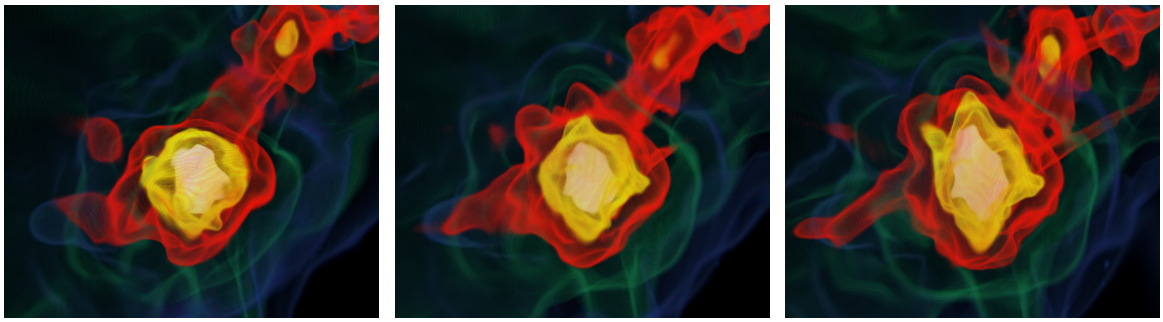

Figure 5: Left and right: keyframes of an AMR simulation portraying the evolution of the first stars in the universe. Middle: interpolation between these keyframes.

ratio of 95% and a minimal extension of 4 yields good results in terms of number of grids and additional memory overhead for theses additional cells, see Section 6.

In principle the clustering procedure could be applied at once to all subgrids on the same level, but this would result in high memory requirements for the signature lists, due to the huge number of cells contained in the minimal bounding box enclosing the higher resolved levels. Thus we perform the clustering procedure per subgrid rather than per level. Per assumption the root level remains unchanged, and the root grid of the intermediate grid hierarchy is given by $\Gamma_0^{0,t} := \Lambda^{0,t_m} = ... = \Lambda^{0,t_{m+n}}$. Now suppose we want to generate the subgrids of a grid $\Gamma_q^{l,t} \subset \Lambda^{l,t}$. The signature list is initialized as follows:

$$S(i,j,k) = \begin{cases} 1, & \text{if } \Omega^{l+1}_{ri,rj,rk} \subset \left(\Gamma_q^{l,t} \cap \left(\bigcup_{s=m}^{m+n} \Lambda^{l+1,t_s}\right)\right) \\ 0, & \text{otherwise.} \end{cases}$$

That is, a cell in $\Gamma_q^{l,t}$ is marked for clustering, if it is refined by the next finer level in at least one of the time-steps $t_m, ..., t_{m+n}$. This signature list is passed to the clustering algorithm that generates a set of subgrids $\Gamma_0^{l+1,t}, \Gamma_1^{l+1,t}, ...$ contributing to the next level $\Lambda^{l+1,t}$. This procedure is recursively repeated for each of the newly created subgrids, until the maximum level is reached, which is defined as the maximum of the number of levels for $H^{t_m}, ..., H^{t_{m+n}}$.

5 Data Interpolation

In this section we describe how the intermediate grid functions $\mathcal{F}^{l,t} \colon \Lambda^{l,t} \mapsto \mathbb{R}$, respectively $\overline{\mathcal{F}}^{l,t} \colon \Lambda^{l,t} \mapsto \mathbb{R}$ associated with the hierarchy H^t are constructed. Two cases have to be distinguished for each for each cell $\Omega^l_{ijk} \in \Lambda^{l,t}$

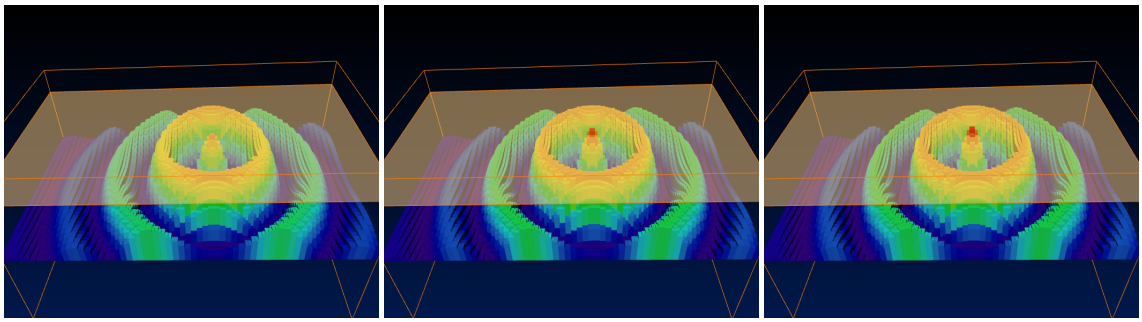

Figure 6: Comparison of three different time interpolation schemes for an analytical dataset showing a damped cosine wave traveling along the x-axis with constant velocity. For linear interpolation oscillations occur, whose minimal peak amplitude during the considered time interval is depicted by the semi-transparent plane. The oscillations decrease for the hermite interpolation and vanish for the flux-based approach. The cell-centered data are rendered using constant spatial interpolation.

- $(\Omega^l_{ijk} \in \Lambda^{l,t_s}) \, \forall \, t_s \, \in [t_m, ..., t_{m+n}]$,
- $\exists \, t_s \in [t_m, ..., t_{m+n}] \quad \text{with} \quad (\Omega^l_{ijk} \notin \Lambda^{l,t_s})$.

In the first case the value $f^{l,t}_{ijk}$ can be obtained as a function of the given functions values f^{l,t_s}_{ijk} at the $n+1$ keyframes. Since in the second case in at least one of the given hierarchies no corresponding cell on the refinement level l exists, we have to apply some form of interpolation on the grid function on the coarser levels of resolution. Let us assume that this is the case for Λ^{l,t_s} in the following. Because of the nesting property of the levels, the cell is covered by at least one coarser cell. Let $\Omega^{\tilde{l}}_{\tilde{i}\tilde{j}\tilde{k}}$ denote the cell on the finest level $\tilde{l} \leq l$ that contains Ω^l_{ijk}. In the *cell-centered* case we perform a nearest neighbor interpolation, i. e. $\overline{f}^{l,t_s}_{ijk} := \overline{\mathcal{F}}^{l,t_s}(\Omega^{\tilde{l}}_{\tilde{i}\tilde{j}\tilde{k}})$. In the *vertex-centered* case f^{l,t_s}_{ijk} is obtained by trilinear interpolation within the cell $\Omega^{\tilde{l}}_{\tilde{i}\tilde{j}\tilde{k}}$.

So for each cell at the intermediate grid hierarchy it has to be determined which of the two cases holds and the associated data samples have to be obtained. For larger hierarchies this can be an expensive operation, even if data structures like kD-trees are employed to speed up this procedure. In order to accelerate this procedure we resample the grid functions $\mathcal{F}^{l,t_s}$ onto a grid with the topology of the intermediate hierarchy in a first step. This involves some interpolation in order to obtain data values for fine cells or vertices that are not present in the given set of hierarchies. After this preprocessing the grid function can be computed

quite fast, since now for each subgrid $\Gamma^{l,t}$ there exists a corresponding subgrid Γ^{l,t_s}. This is especially advantageous if more than one time-step for the same subset of keyframes has to be be generated.

We employ three different schemes for the interpolation in time. The first one is piecewise linear, i .e.

$$f_{ijk}^{l,t} = \frac{t_{s+1} - t}{t_{s+1} - t_s} f_{ijk}^{l,t_s} + \frac{t_s - t}{t_s - t_{s+1}} f_{ijk}^{l,t_{s+1}} \tag{1}$$

The second one is $\mathcal{C}^1$ continuous *cubic hermite interpolation*, see for example [11]. So besides the function values for t_s and t_{s+1}, also the first derivatives at these time steps are taken into account. This uniquely determines a polynomial of degree 3, that can be written as

$$\begin{aligned} f_{ijk}^{l,t} := f_{ijk}^{l,t_s}\, H_0^3(t) \quad &+ \quad \frac{d}{dt} f_{ijk}^{l,t_s}\, H_1^3(t) + \\ f_{ijk}^{l,t_{s+1}}\, H_2^3(t) \quad &+ \quad \frac{d}{dt} f_{ijk}^{l,t_{s+1}}\, H_3^3(t), \end{aligned}$$

where $\{H_0^3, ..., H_3^3\}$ are the *cubic Hermite Polynomials*. Since the first derivatives of the grid functions are usually not available during the visualization phase, we estimate them using the osculatory method [9]. The value of the first derivative at t_s is approximated by the slope of the unique polynomial of degree 2 that takes the function values at t_{s-1}, t_s, t_{s+1}. This implies that we need the grid function values for 4 successive time-steps in order to obtain an approximation of the first derivatives at t_s and t_{s+1}.

It is possible to obtain more precise values for the first derivatives of the grid functions in the case that the grid function represents some conserved quantity that fulfills a conservation law of the form

$$\frac{\partial}{\partial t} \rho(\vec{x}, t) = div\ \vec{j}(\vec{x}, t), \tag{2}$$

where $\rho(\vec{x}, t)$ denotes the density of the conserved quantity and $\vec{j}(\vec{x}, t)$ is the associated current. An common example is the case of mass conservation in hydrodynamic simulations. Here $\rho(\vec{x}, t)$ denotes the mass density and the current is computed from the density and the velocity field $\vec{v}\,(\vec{x}, t)$ via $\vec{j} = \rho\, \vec{v}$. Since usually in hydrodynamic simulations besides the mass, respectively density fields, also the associated velocity vector-fields are stored, one can compute the derivative of the scalar field according to Eq. 2. The divergence of the current is approximated by the flux of the mass through each of the cells faces

$$div\ \vec{j} = div\ (\rho\, \vec{v}) = \frac{1}{V_{cell}} \sum_{i=0}^{5} \rho_i\, A_i\, \vec{n}_i\, \vec{v}_i$$

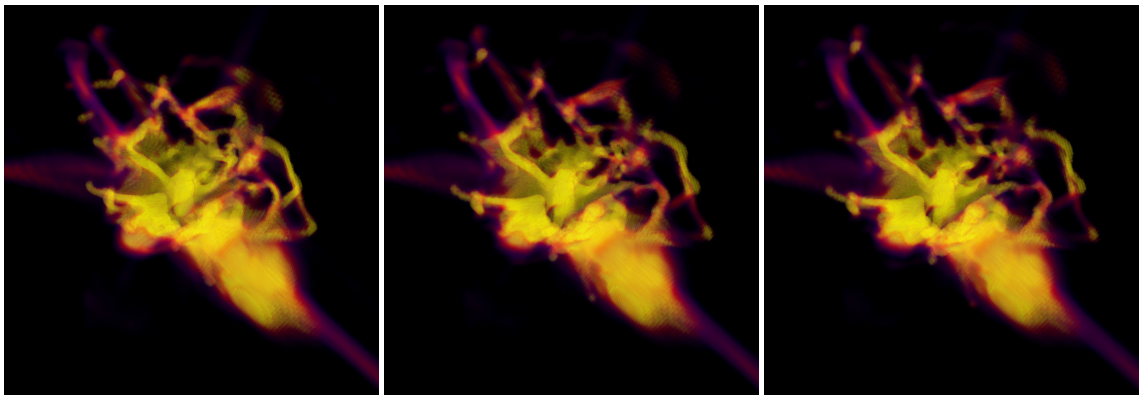

Figure 7: Three interpolated frames from a simulation describing a supernova explosion visualized by texture-based volume rendering.

where $\rho_i, \vec{v}$ denote the density and velocity fields evaluated at the i-th face and $A_i, V_{cell}, \vec{n}_i$ the face area, its volume and the outward-oriented face normals.

6 Results & Discussion

The algorithm has been implemented in Amira [1], an object-oriented, expendable 3D data visualization system developed at ZIB. We applied the algorithm to three time-dependent AMR datasets. The performance was tested on a SGI Onyx3 on a single 500 MHz MIPS R14000 processor.

Dataset 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 grid per time-step. Dataset 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 linear and Hermite interpolation, with estimated first derivatives, see Section 5. Figure 5 and 7 show some volume rendered images of the sequences. The resulting animations show slight oscillations in some parts for linear interpolation, which decrease for Hermite interpolation. Besides that the resulting animations are smooth.

For illustrating the differences of animation quality of the different interpolation schemes we choose an analytical example as dataset III. It shows a damped cosine oscillation that moves along the x-axis with constant velocity. The $64 \times 32 \times 32$ root grid is refined two times and contains about 1100 sub grids at each of the 20 keyframes. We compared the animation quality of linear, Hermite and flux-based interpolation by generating 8 intermediate frames per pair of data steps. Figure 6 show some volume rendered images of the resulting sequences.

	increase # cells	grid generation	interpolation
Dataset I (linear)	7%	3.4 sec	0.2 sec
Dataset I (Hermite)	12%	6.6 sec	0.8 sec
Dataset II (linear)	11%	2.2 sec	0.1 sec
Dataset II (Hermite)	15%	3.5 sec	0.3 sec
Dataset III (linear)	20%	1.8 sec	0.1 sec
Dataset III (Hermite)	30%	4.2 sec	0.3 sec
Dataset III (flux-based)	20%	3.5 sec	2.0 sec

Figure 8: The first two columns denote the increase in the number of cells for the intermediate time steps relative to the given keyframes. The third column states the times for generating the intermediate grid and projecting the given grid functions. This has to be carried out only if the keyframes change. The last column gives the time for interpolation of the intermediate grid function.

The resulting animation show disturbing oscillations for linear interpolation. The oscillations decrease for Hermite interpolation with estimated derivatives and vanish for the flux-based interpolation, where the knowledge of the velocity vector fields are taken into account, as described in Section 5.

Information about performance and memory requirements is given in Figure 8. The number of sub grids in the merged hierarchies is decreased by about 20 % compared to the number of subgrids present in the stored hierarchies. As can be seen in the table the amount of additional memory requirements and the times for grid generation where highest for the Hermite interpolation, since 4 keyframes had to be merged in this case. But the space increase was still less than 30 % in all examples. The middle row depict the times for grid generation and keyframe projection, which has to be carried out only if the keyframes change. Again it was highest for Hermite interpolation, but with less than 7 sec even for the 2000 grid data set it still admits a on-the-fly generation during the visualization phase. Due to the keyframe projection step the times for the interpolation (right row) are short, which is advantageous if more than one intermediate frame is generated for a constant set of keyframes.

7 Conclusions & Future Work

In this paper we addressed the problem of temporal interpolation of AMR grid data, e.g. for creation of smooth animations or feature tracking. 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 are mapped to the intermediate hierarchy and interpolated using different schemes, like linear, Hermite or flux-based interpolation. The algorithms are fast and allow on-the-fly generation of interpolated frames.

There are several ways to extend the presented algorithm. Higher order interpolation schemes could be implemented for spatial interpolation during the prolongation step. Further it would be interesting to combine the presented approach with feature tracking algorithms. Also it seems promising to adapt the order of temporal interpolation to the rate of change of the underlying data. It might be beneficial to use lower order temporal interpolation for subgrids with slowly varying data and higher order interpolation for subgrids with rapidly changing data (which is usually the case for the higher resolved levels).

8 Acknowledgments

We thank Tom Abel (Pennsylvania State University, State College), Stuart Levy (National Center for Supercomputing Applications (NCSA), Urbana) and Greg Bryan (University of Oxford, Oxford) for providing the cosmological datasets and for fruitful discussions. This work was supported by BMBF/DFN, project GriKSL TK 602.

References

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

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

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

[4] Wolfgang Bangerth, Ralf Hartmann, and Guido Kanschat. `deal.II` *Differential Equations Analysis Library, Technical Reference*. `http://www.dealii.org`.

[5] P. Bastian, K. Birken, K. Johannsen, S. Lang, N. Neuss, H. RentzReichert, and C. Wieners. Ug – a flexible software toolbox for solving partial differential equations, 1997.

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

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

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

[9] K. Brodlie and P. Mashwama. Controlled interpolation for scientific visualization. *Scientific Visualization: Overviews, Methodologies and Techniques*, pages 253–276, 1997.

[10] G. L. Bryan, T. Abel, and M. L. Norman. Achieving extreme resolution in numerical cosmology using adaptive mesh refinement: Resolving primordial star formation. In *Proc. of Supercomputing 2001*, 2001.

[11] P. Deuflhard and A. Hohmann. *Numerical Analysis in Modern Scientific Computing: An Introduction.* Springer, 2003.

[12] 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.

[13] 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.

[14] R. Kähler, D. Cox, R. Patterson, S. Levy, H.-C. Hege, and T. Abel. Rendering the first star in the universe - a case study. pages 537–540. IEEE, 2002.

[15] R. Kähler and H. C. Hege. Texure-based volume rendering of adaptive mesh refinement data. *The Visual Computer*, (18):481–492, 2002.

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

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

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

[19] 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.

[20] S. Park, C. L. Bajaj, and V. Siddavanahalli. Case study: Interactive rendering of adaptive mesh refinement data. pages 521–524. IEEE, 2002.

[21] 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.

[22] 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.

[23] D. Silver and X. Wang. Volume tracking. In Roni Yagel and Gregory M. Nielson, editors, *IEEE Visualization '96*, pages 157–164, 1996.

[24] G. H. Weber, H. Hagen, B. Hamann, K. I. Joy, T. J. Ligocki, K.-L. Ma, and J. Shalf. Visualization of adaptive mesh refinement data. In *Proceedings IS&T/SPIE Electronic Imaging 2001, volume 4302*, San Jose, CA, USA, 2001.

[25] 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.

[26] 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.