Chapter 5

Visualization of Structured Adaptive Mesh Refinement Data

Due to the growing popularity of structured AMR schemes for representing multi-scale phenomena in the last years, an increasing number of scientists is in need of appropriate interactive visualization techniques to interpret and analyze time-dependent simulation data defined on this kind of grids. Tools for both, 2D analysis to quantitatively convey the information within single slices and a 3D representation to quickly grasp the overall structure are required.

A naive method for rendering structured adaptive mesh refinement data is to resample the grid function onto a uniform mesh with a cell size corresponding to highest resolved level. The data for the new grid nodes that are not present in the original AMR grid, has to be obtained by interpolation. Though this approach has the advantage that it allows to apply all standard visualization routines available for uniform grids, it is unfeasible even for moderately resolved AMR data due to the enormous amount of memory requirements. As an example consider a time-dependent hierarchy with five levels of refinement, a root level subgrid with 128^3 cells, an overall refinement factor of 2 and a grid function with 4 bytes per node. Resampling such an hierarchy would result in an uniform grid that requires 256 GByte of memory for each time step.

An alternative approach is to convert AMR data to an unstructured, locally refined hexahedral grid with explicit connectivity information. Though not as disadvantageous as the approach mentioned above, it also increases the memory requirements for storing the data and it further does not take advantage of the regular structure of the subgrids, that allows for fast and efficient rendering.

Another drawback of these conversions is that information about the grid function on different levels of resolution that is present in the original data is lost. Scientists are often interested in comparing the grid functions on different levels of refinement in order analyze their codes and numerical algorithms, and/or inspect the behavior of the solution at the level boundaries.

Thus converting AMR data to standard grid types is neither a feasible nor a desirable strategy. It is much more advantageous to develop visualization techniques that can directly handle and utilize the hierarchical structure of AMR data and at the same time take advantage of the fact, that it consists of structured subgrids, in order to accelerate the rendering.

This will be our basic paradigm for all visualization approaches presented in the next sections. This chapter is organized as follows: In the next Section related work in the field of hierarchical visualization techniques is discussed. Next the topics of point location and interpolation for AMR data are addressed. In Section 5.4 we discuss the adaption of indirect volume rendering methods for AMR data, whereas in Section 5.5 direct volume rendering approaches for AMR data are presented.

5.1 Related Work

To our knowledge the first paper in the literature that deals with visualization of AMR data was published in '93 by Max [57]. Max describes a back-to-front cell-sorting algorithm for AMR grids and employs it for a software-based volume rendering approach and for contour surface extraction.

Norman et al. [60] describe an approach of resampling AMR data to uniform as well as unstructured grids, which allows them to apply standard rendering algorithms, though their approach is feasible only for AMR hierarchies with a small number of refinement levels, as discussed at the beginning of this chapter.

There exist quite a number of papers that deal with the problem of artifacts in triangulations that result from the standard marching cubes algorithm, if it is applied to non-conforming hexahedral grids. Shu et al. [81] use axis-aligned patches to fill the cracks in the multi-resolution isosurface mesh. A disadvantage of this approach is the visually noticeable change of the surface normal between triangles resulting from the marching cubes approach and the triangles of the planar patches.

Shekkar et al. [78] propose a crack-fixing scheme that shifts vertices of triangle edges located on the higher resolved face parallel to the face in order to align with triangle edges of adjacent lower resolved cells. By this some details present in the higher resolved regions might get lost and the resulting meshes contain T-vertices, which might lead to shading artifacts.

Westermann et al. [94] discuss two strategies for dealing with the crack problems for octrees. Their first approach utilizes a conforming split scheme in order to eliminate hanging nodes. Since this approach can lead to a noticeable increase of the number of generated triangles, they propose a second scheme in which coarse triangles which abut higher resolved cells are split up into triangle fans, in order to match the shape of the adjacent smaller triangles. Compared to [78] this has the advantage that the detail information of the higher resolved regions is preserved, though it is feasible only for grids, where adjacent cells differ by at most one level of resolution and hanging nodes that are dependent nodes. Fang et al. [28] extended the conforming split approach proposed in [94] to deal with vertex-centered data defined on unrestricted AMR grids. Their approach still requires that the data at hanging nodes are dependent nodes.

In contrast to the approaches mentioned above, Weber et al. [89] address the problem of isosurface generation for cell-centered data defined on AMR grids. They propose the use of several types of stitching cells to connect cells on different levels of resolution. This approach avoids resampling of cell-centered data, but has the drawback that it restricts the applicability of their algorithm to restricted AMR grids, which guarantee that refined levels are surrounded by at least one layer of cells from the next coarser level.

Two parallel, direct volume rendering approaches for octree data generated by the PARAMESH framework [55] are compared by Ma [54]. The first one performs a resampling of the data to a uniform grid, similar to [60], rendering subregions of the resulting grid on individual processors, whereas the other methods utilizes the octree hierarchy and distributes its leaf nodes.

In [90] Weber et al. apply the stitching scheme presented in [89] to a cell-projection algorithm. Park et al. [66] present a hierarchical splatting approach for AMR data. Kelley et al. [34] describe a framework for interactive, parallel volume rendering of remote AMR data. Sub-trees of the AMR hierarchy are distributed on individual processors and composed on the local rendering client. Weber et al. [91] further investigate load-balancing strategies for parallel volume rendering of AMR data.

Visualization methods for time-dependent simulations carried out on locally refined, unstructured grids, have been presented by Polthier et al. [68], Happe et al. [31] and Schmidt et al. [74]. These approaches have in common that they assume the existence of two unstructured grids and associated grid functions at each time step at which the underlying grid structure is changed: 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.

5.2 Point Location for AMR Data

Since in AMR grids subregions of the computational domain might be covered by multiple cells from different levels of resolution, we have to specify the point location operation, which is important for many visualization methods, as follows:

Given a maximal search depth $\tilde{l}$ with $0 \leq \tilde{l} \leq l_{max}$, locate the unique cell $\Omega^l_{ijk} \in \Lambda^l$ with $0 \leq l \leq \tilde{l}$, for which

$$(\boldsymbol{p} \in \Omega^l_{ijk}) \ \wedge \ (\boldsymbol{p} \notin \Lambda^{l+1} \cup \ldots \cup \Lambda^{l_{max}})$$

holds.

Notice that $\tilde{l}$ does not necessarily have to equal the highest resolved level. It is often beneficial to use coarser approximations of the grid function in some subregions, for example in order to accelerate rendering and/or preprocessing times. In this case it should be avoided to interpolate data samples on the highest available level for coarser representations. This will in general violate the sampling theorem and result in aliasing artifacts, since the sampling rate on the coarse levels is too low.

A direct approach for cell location is to inspect each subgrid on all refinement levels $0 \leq l \leq \tilde{l}$, starting at level $\tilde{l}$, until a subgrid that covers the considered location is determined. Due to the uniform structure of the subgrids, the associated local coordinates (u, v, w) are given by

$$(u, v, w) = (\frac{\mathbf{p}_x - \mathbf{m}_x}{\mathbf{d}_x} - [\frac{\mathbf{p}_x - \mathbf{m}_x}{\mathbf{d}_x}], ..., ...)$$

where $[.]$ denotes the floor function, $\mathbf{m}$ the minimal coordinate of the subgrids bounding box and $\mathbf{d}$ the cell size. This search strategy is not feasible for most visualization algorithms, like for example direct volume rendering via raycasting, which involve a large number of point location operations.

A way to speed up this process is to exploit the inherent hierarchical structure of AMR hierarchies. Simulation packages, like for example the AMR code ENZO [65], often do generate nested subgrids, so each Γ_j^l (except for the root level subgrids) is completely contained in exactly one subgrid $\Gamma_j^{l-1} = \mathcal{P}(\Gamma_j^l)$ on the next coarser level of refinement Λ^{l-1}. This tree-structure can be directly exploited for a hierarchical search strategy.

In case the subgrids are not arranged in a nested manner, this tree structure can always be induced in a preprocessing step, as indicated in the following pseudocode:

```
set minimal bounding box of all $\Lambda^0$ as $\Gamma^{-1}$;
insert all $\Gamma_i^0 \in \Lambda^0$ as $\mathcal{C}(\Gamma^{-1})$;
for all ($\Lambda^l$, $l = 0, ..., l_{max} - 1$) {
    for all ($\Gamma_i^l \in \Lambda^l$) {
        for all ($\Gamma_i^{l+1} \in \Lambda^{l+1}$ {
            if $((\Gamma_i^l \cap \Gamma_j^{l+1}) \nsubseteq (\partial\Gamma_i^l \cup \partial\Gamma_j^{l+1}))$
                set $(\Gamma_i^l \cap \Gamma_j^{l+1})$ as $\mathcal{C}(\Gamma_i^l)$
        }
    }
}
```

Though for large n this hierarchical approach is preferable to the linear search strategy mentioned above, it still suffers from the drawback that AMR hierarchies tend to be not well balanced, i. e. the degree of the nodes can vary considerably. Subgrids located on the coarser levels usually have a much larger number of descendants than the higher resolved ones. Subgrids with hundreds of subnodes are not unusual. Thus the traversal of the hierarchy becomes less beneficial. This in particular decreases the efficiency of local search strategies. Even if the next query requests a point that is located in the same subgrid, it is still necessary to inspect the potentially large number of the child nodes.

We address this problem by subdividing the computational domain into axis-aligned blocks $\mathcal{B}_m^l$ that enclose cells from the same resolution level, i. e. we decompose the grid domain into regions of cells that are either completely refined or unrefined

$$\Lambda^l = \bigcup_{m=0}^{n} \mathcal{B}_m^l \quad \text{for } 0 \leq l \leq l_{max},$$

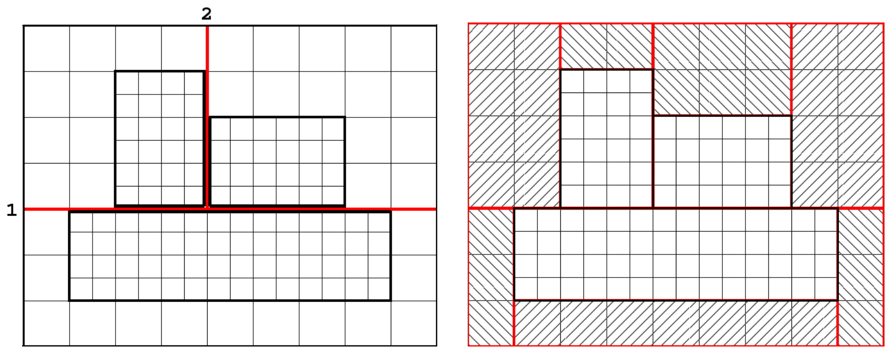

Figure 5.1: **Left:** 2D example of a grid, that contains 3 subgrids on the next level of resolution. In order to subdivide the volume into axis-aligned, non-overlapping regions containing only cells of the same refinement level, two split axes are determined. **Right:** Subregions that contain just one subgrid, are finally partitioned in up to 4 subregions (up to 6 in the 3D case)

such that

$$\left(\mathcal{B}_m^l \subseteq \Lambda^l \setminus (\Lambda^{l+1} \cup ... \cup \Lambda^{l_{max}})\right) \vee \left(\mathcal{B}_m^l \subseteq \Lambda^{l+1} \cup ... \cup \Lambda^{l_{max}}\right).$$

Regarding the discussion above this decomposition should further have the following properties

- $(\mathcal{B}_i^l \cap \mathcal{B}_j^l) = \emptyset \vee (\mathcal{B}_i^l \cap \mathcal{B}_j^l) \subset (\partial\mathcal{B}_i^l \cup \partial\mathcal{B}_j^l)$ for $i \neq j$, i. e. the interiors of the blocks on the same level should be disjoint,
- the subdivision should consist of a small number of blocks,
- the blocks should be nested, that is for each $\mathcal{B}_i^l$ exists exactly one $\mathcal{B}_j^{l-1}$ with $\mathcal{B}_i^l \subseteq \mathcal{B}_j^{l-1}$,
- and the nodes of this nested hierarchy should be of small degree.

In principle one could exploit an octree-data-structure for this purpose. The root node, covering the domain of the AMR root level, is recursively subdivided into 8 child nodes, until each leaf contains only cells on the same level of resolution. The drawback of this approach is that it usually leads to a large number of bricks, since often the partition has to carried out until the subregions contain exactly one cell. As an example consider a subgrid of $n \times n \times n$ cells, that is separated from the nearest boundary of its parent level by just one layer of cells. Here the octree subdivision scheme would generate $n \times n$ leaf nodes to cover this layer. Besides high storage requirements this large number of small nodes is disadvantageous for local search strategies.

In the following we propose a different approach that aims at generating a smaller number of blocks. The decomposition starts at the root level. In order to ease the discussion we assume that the root level consists of just one subgrid Γ_0^0. This is no loss of

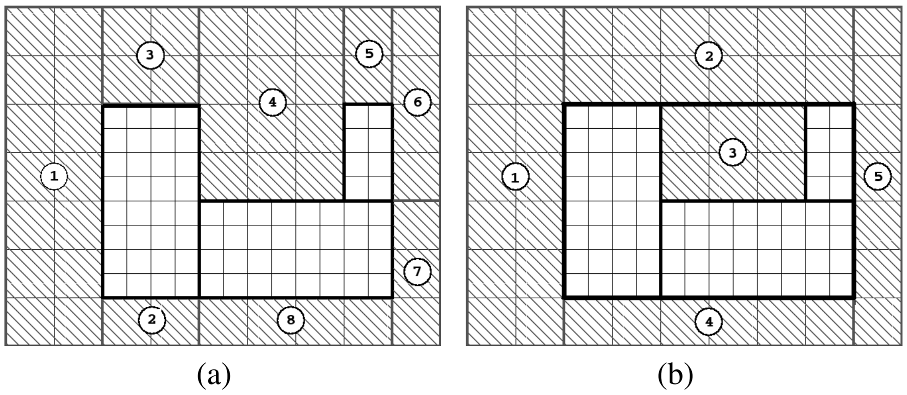

Figure 5.2: **(a):** Subgrid configuration where the unmodified decomposition leads to an unnecessarily high number of bricks. **(b):** By first enclosing the subgrids with a minimal bounding box, and decomposing the outer region, the number of created bricks is reduced.

generality, since in the case that the root level consists of multiple subgrids, we can introduce a virtual level that is simply defined by the subgrids minimal axis-aligned bounding box. We further assume that the subgrids are arranged in a nested structure, which always can be generated in a preprocessing step as discussed above.

The decomposition strategy is to recursively separate subgrids on the same level of resolution by axis-aligned hyperplanes. Starting at the root level, we perform a line-sweep along the three coordinate axis, in order to detect separating planes that do not intersect any of the subgrids as indicated in Figure 5.1. In the case that several separating hyperplanes $\mathcal{P}$ exist, the one that leads to the most balanced split is chosen. We define the most balanced split as the one that minimizes

$$|n_{low}(\mathcal{P}) - n_{up}(\mathcal{P})|,$$

where $n_{low}(\mathcal{P})$ and $n_{up}(\mathcal{P})$ denote the number of subgrids $\Gamma \in \mathcal{H}$ that are located in the lower, respectively upper subregion defined by the plane $\mathcal{P}$ with respect to the considered coordinate axis. For some configurations each hyperplane intersects at least one subgrid, consider for example the configuration shown in Figure 4.4. In this case the plane that intersects the smallest number of subgrids is chosen.

This splitting procedure is recursively continued on each of the two newly generated subregions introduced by the plane $\mathcal{P}$, until each of them contains at most one subgrid or part of one subgrid. This configuration can easily be split up into disjoint blocks of refined and unrefined cells. For certain configurations the partition algorithm discussed so far, still produces unnecessarily many blocks. Figure 5.2 (a) shows an example. In such cases we compute the minimal bounding boxes of the subgrids and decompose the outer region, like indicated on the Figure 5.2 (b). Notice that this never increases the number of created blocks. For our datasets, this step reduced the total number of leaf bricks by up to 10 %.

The splitting procedure described above is continued on each block that consists of cells that are refined by grids on the next level of resolution, until the highest resolved level is reached. This resulting space decomposition results in blocks that are arranged in the manner of an adaptive kD-tree data structure, compare Section 2.4, consisting of three different types of nodes:

- The first type represents blocks that cover cells that are not further refined. These nodes are leaf nodes of the associated kD-tree.
- Nodes representing blocks that cover cells which are refined on the next level on resolution. These also store information about the partition axis and references to the two subnodes, associated with the next two subvolumes. We use nodes of this type to prune the traversal of the hierarchy once a certain level of resolution is reached.
- The third type of nodes is primarily used for tree traversal. It holds information about the next partition plane and references to the next two subnodes.

Information of the number of generated block for several application examples will be given in Section 5.5.5.

5.3 Interpolation

Assuming continuous fields, e. g. those that do not contain jumps, which are for example present at material boundaries or shock fronts, a prerequisite for artifact-free rendering is a globally continuous interpolation of the discrete data samples.

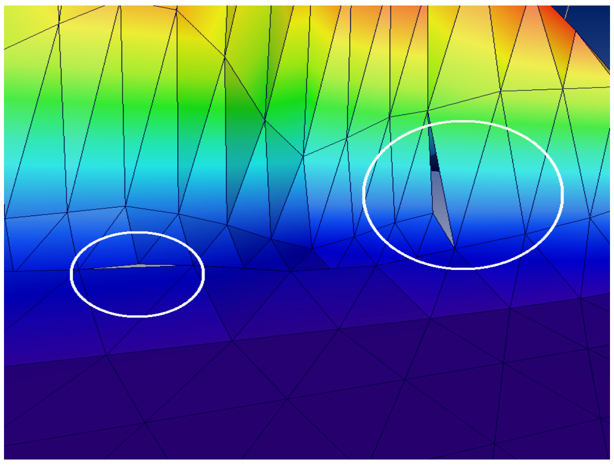

Figure 5.3: Cracks in a height field mesh, resulting from discontinuous interpolation.

For AMR data this requires special attention at the boundaries of the refinement levels. Problems might arise at hanging nodes, present at common faces of cells that belong to different resolution levels.

5.3.0.1 Vertex-Centered Data

Let us first discuss the case of vertex-centered data, for which typically piecewise trilinear interpolation is employed, if $\mathcal{C}^0$-continuity of the interpolant is desired. $\mathcal{C}^0$-continuous trilinear interpolation at common faces of adjacent cells of the same resolution level is guaranteed, since the interpolants on both sides degenerate to the same bilinear interpolant on the face. Continuity at boundaries of grid cells with different resolution requires the following:

Firstly, the data values on the boundary nodes that are no hanging nodes (white circles in Figure 5.4) have to coincide on the coarse and fine grid. This is usually ensured in numerical AMR schemes by the *restriction* step, in which the solution on the coarser grid nodes is updated by the more accurate grid function of the refined regions, compare Section 2.2. In general non-dependent hanging nodes (black circles in Figure 5.4) lead to discontinuities of the piecewise trilinear interpolant, resulting in visible artifacts for most visualization methods, as for example shown in Figure 5.3. There are two alternatives to deal with this problem: One is to replace the data values at hanging nodes by the result obtained via bilinear interpolation of the coarse cell faces, i. e. to convert them to dependent nodes. This typically introduces only minor corrections, because the difference between the fine and coarse solutions is usually small at level boundaries, since most error estimators detect regions that require refinement based on this difference. Further the number of hanging nodes is small compared to the number of all nodes in the grid hierarchy.

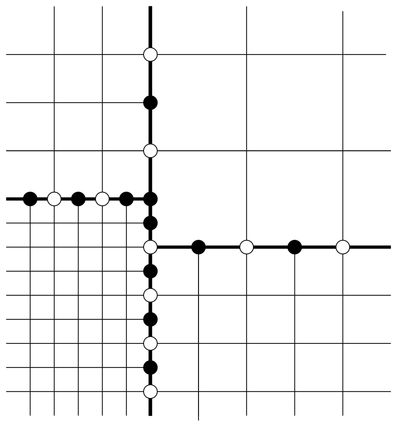
Figure 5.4: Different types of nodes at level boundaries.

Another method to obtain continuity which does not change the grid function, is to modify the underlying grid structure in order to construct a conforming grid. In this approach (*conforming split*), the coarse cells at level boundaries are replaced by a set of sub-cells of different topology, such that each internal face is shared by exactly two adjacent cells. In Subsection 5.4.3.3, we will present a conforming split scheme for AMR grids, that is capable of handling arbitrary level differences between adjacent cells and requires only two types of split cells.

We apply both techniques in the visualization routines discussed in the following. The first method is more advantageous for direct volume rendering approaches, since the

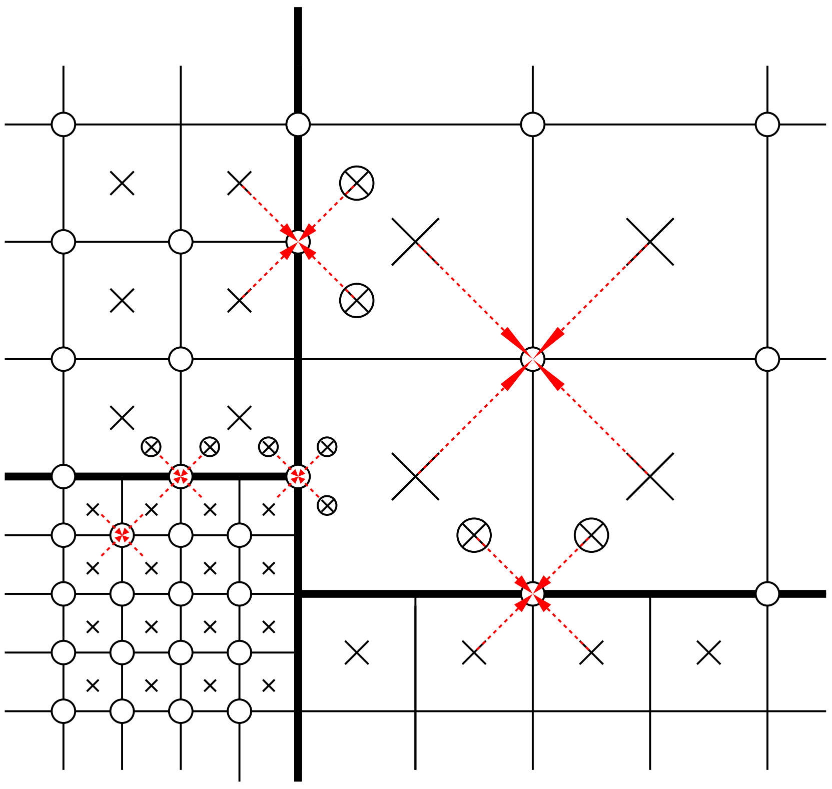

Figure 5.5: Resampling from cell-centered data to vertex-centered data: Data at the non-hanging vertices (circles) is obtained via trilinear interpolation between the adjacent cell-centered data samples (crosses) using a stencil width that corresponds to the mesh size of the highest resolved adjacent grid. At boundaries missing samples (circles with crosses) are generated via trilinear interpolation on the coarse grid. Data at hanging nodes is obtained via bilinear interpolation between the adjacent non-hanging nodes.

performance of the methods would be decreased significantly if the uniform structure of the subgrids is partially replaced by hybrid grid patches. For isosurface algorithms the second approach is appropriate, since the conforming split avoids cracks, which arise at level boundaries, even for a globally continuous interpolant, compare Section 5.4.3.

5.3.0.2 Cell-Centered Data

Let us now discuss the case of cell-centered data. Nearest-neighbor interpolation is the natural interpolation scheme this kind of data, since it allows a direct inspection of the data samples. We therefore support this kind of interpolation for cell-centered data in all visualization techniques discussed in the next sections. However, the ’blocky’ appearance of the resulting representations, that is especially pronounced on lower resolved regions, might obstruct the perception of the qualitative features of the data, consider for example the isosurface shown in Figure 5.10. Hence higher-order interpolation scheme are

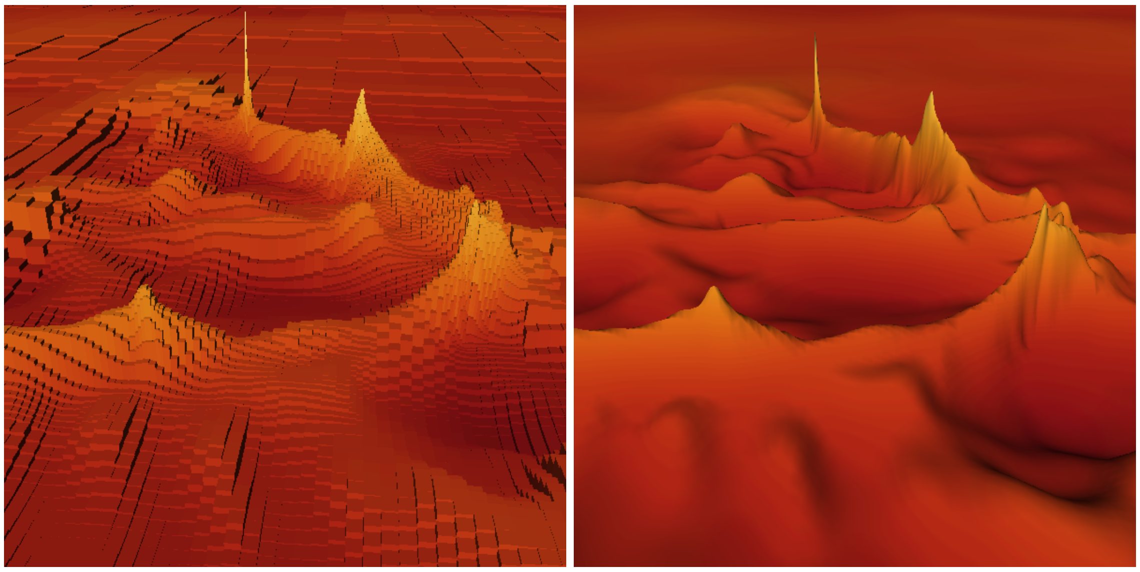
(a) (b)

Figure 5.6: Comparison of two height field meshes: a blocky mesh extracted from the original cell-centered data using nearest neighbor interpolation (a) and a smooth mesh extracted from the resampled vertex-centered data using tri-linear interpolation (b). *(dataset courtesy of M. Norman, National University of California)*

desirable also for cell-centered data.

One approach is to work on the *dual* grid, that has vertices at the locations of the data samples, i. e. at the cell centers, and to use trilinear interpolation on this grid structure. Problems arise at the boundaries of the levels, where the regular structure of the dual grid is disturbed. Since one of our requirements is to deal with AMR grids, that allow adjacent cells to differ by more than one level, i. e. unrestricted AMR hierarchies, a stitching cell approach to connect the cells of the dual grid like the one proposed by Weber et al. [89], compare Section 5.1, is not feasible for our purpose.

We therefore decided to resample the cell-centered data into a vertex-centered form, which is better suited for trilinear interpolation, as discussed above. In order to determine the value of the grid function at the grid nodes, we need to construct an interpolation rule for the grid function based on the cell data. According to Figure 5.5, we have to distinguish the following cases:

- Data for vertices $\boldsymbol{x}_{ijk}$ in the interior of a refinement level Λ^l is obtained by trilinear interpolation of the eight data samples $\bar{f}$ at the adjacent cell centers, i. e. (assuming cubic cells for simplicity)

$$f(\boldsymbol{x}_{ijk}) := \frac{1}{8} \sum_{l,m,n=-1,1} \bar{f}\left(\boldsymbol{x}_{ijk} + \frac{1}{2}(lh_0, mh_1, nh_2)\right), \qquad (5.1)$$

where $\mathbf{h}$ is the grid spacing on Λ^l.

- For boundary nodes $\boldsymbol{x}_{ijk} \in \partial\Lambda^l$ that are not hanging nodes, the same interpolation kernel as for interior nodes is used. In this case the width of the kernel is determined by the highest resolved cell that shares the vertex $\boldsymbol{x}_{ijk}$. Data samples at locations $\boldsymbol{x}_{ijk} + \frac{1}{2}(\pm h_0, \pm h_1, \pm h_2,)$ that are not present on this level are obtained by trilinear interpolation on the highest resolved refinement level, that contains the point.
- In order to achieve a $\mathcal{C}^0$-continuous trilinear interpolant the data associated with the hanging nodes are obtained by bilinear interpolation between the data at the adjacent non-hanging nodes.
- Boundary values on the root level Λ^0 are generated via extrapolation.

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

5.4 Indirect Volume Rendering

In this section we will discuss the modifications to some popular indirect volume rendering algorithms that are necessary when they are applied to structured AMR data. The first part of the section deals with approaches for displaying data within planar slices and in the second part a variant of the marching cubes algorithm, suitable for unrestricted, refined hexahedral grids will be presented.

5.4.1 Planar Slices

Our approach for rendering data within planar slices utilizes 2D textures, in order to allow interactive performance even if a large number of extracted slices is displayed simultaneously. For each subgrid that is intersected by the slice $\mathcal{S}$, a separate 2D texture is allocated and initialized with the extracted data samples. Multiple rendering of regions that are covered by subgrids on different resolution levels would result in noticeable artifacts, in particular if semi-transparent colormaps are applied. In order to avoid this, we employ the stencil-buffer, offered by standard graphics APIs.

First the textures for the highest resolved level are processed. Each rendering of a texture updates the stencil buffer, thus preventing the associated regions in the frame buffer from being overwritten by subsequently processed textures from the coarser levels, as indicated in the following pseudocode:

```
/* enable stencil test */
glEnable(GL_STENCIL_TEST);
/* suppress rendering in regions where stencil is 1 */
glStencilFunc(GL_NOTEQUAL, 0x1, 0x1);
glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);
/* reset buffer */
glClearStencil(0x0);
```

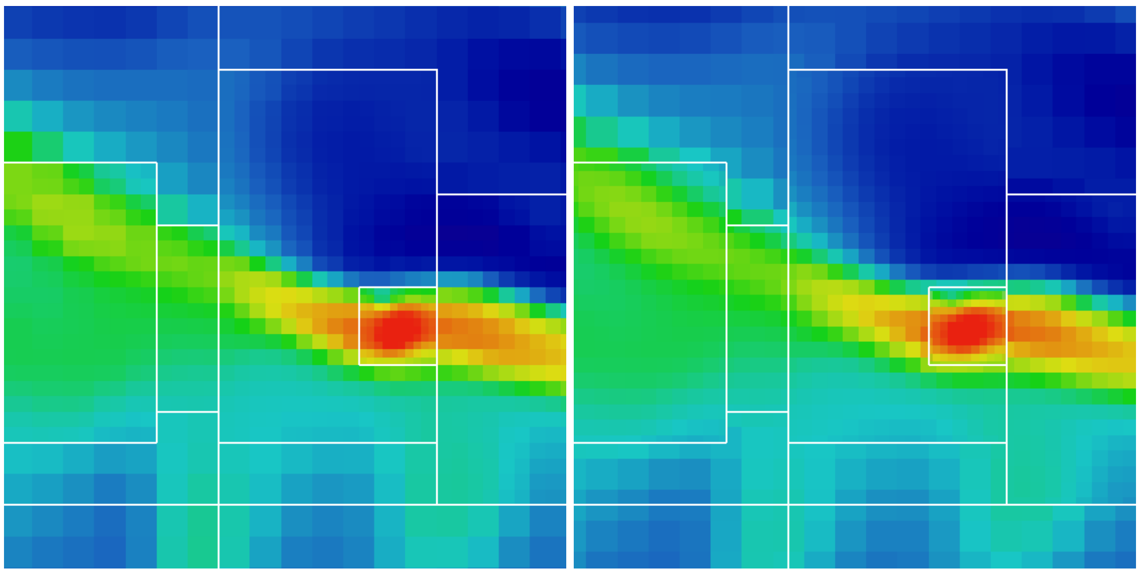
(a) (b)

Figure 5.7: Texture offsets: For cell-centered data (a) textures are aligned with cell centers, whereas for vertex-centered data (b), texels are aligned with the vertices. The latter one allows smooth transitions between the different texture patches if bi-linear interpolation is employed and the “hanging” texels are adjusted. In (b) nearest-neighbor interpolation was used, in order to better depict the texel offsets.

```
for ($l = l_{max}, ..., 0$) {
    for all ( $\Gamma_i^l \in \Lambda_l$ with $(\Gamma_i^l \cap \mathcal{S}) \neq \emptyset$) {
        render texture of $(\Gamma_i^l \cap \mathcal{S})$;
    }
}
```

For cell-centered data we apply nearest-neighbor interpolation, and the texels are aligned with the centers of the squares that result from the intersection between the slice and the cells, compare Figure 5.7 (a). For vertex-centered data the texels are aligned with grid nodes, respectively with the intersections between the cell edges and the slice, as indicated in Figure 5.7 (b). In this case the textures are rendered using bilinear interpolation. In order to ensure a continuous interpolation across boundaries of textures associated with different levels, we compute the boundary texels that belong to hanging nodes by bilinear interpolation between the adjacent coarse texels, compare Figure 5.8.

5.4.2 Height Fields

Let us now discuss the extraction of height fields for AMR data. For cell-centered data we display them as axis-aligned bars with top-faces perpendicular to the considered data

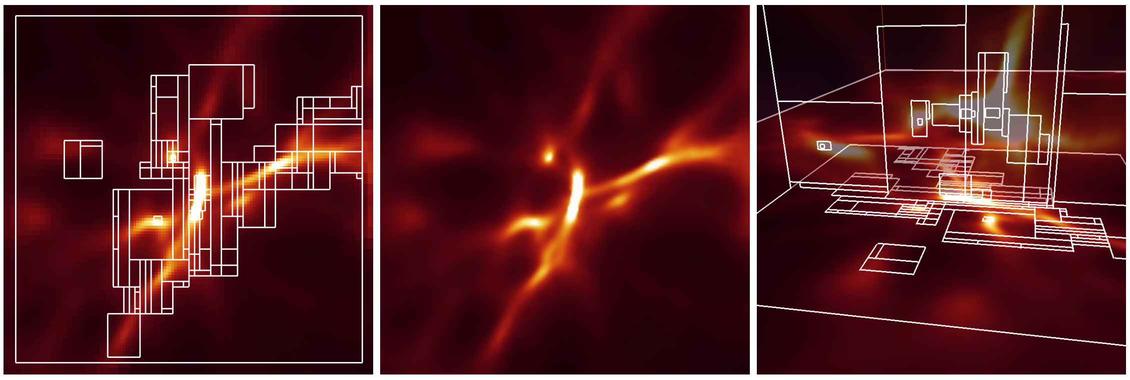

Figure 5.8: Texture-based rendering of AMR data within orthogonal slices. In the left image nearest neighbor interpolation was applied. The subgrid boundaries are indicated by the white lines. In the middle image the textures are interpolated using bilinear interpolation. Since the 'hanging texels' are adjusted appropriately, no interpolation artifacts at the interface between textures of different resolution occur. Utilizing the stencil buffer allows to render semi-transparent textures without changes in the overall transparency in regions that are covered by multiple subgrids, as shown in the right image. *(dataset courtesy of M. Norman, National University of California)*

slice. The height is proportional to the value of the grid function at the projection of the top-face onto the considered slice as shown in Figure 5.6 (a). For vertex-centered data we approximate the resulting surface by a triangular mesh. The data of cells in the interior of a refinement level, which are intersected by the considered slice, is represented by two triangles, compare Figure 5.9 (a).

Problems arise at coarse cells that abut on a boundary of a higher level of resolution. Although adjusting the data at the hanging nodes to obtain slave nodes, as discussed above, ensures a continuous interpolation across the boundary, the resulting surface still contains T-vertices. These can result in shading artifacts, compare Figure 5.9 (c).

So instead of modifying the grid function at hanging nodes, we rather decided to remove these nodes by replacing the triangles of coarse boundary cells by sets of smaller triangles, which provide a transition from the coarse triangles of the interior cells to the higher resolved ones from the adjacent finer level. Therefore we introduce a new central vertex and connect it to the triangle nodes from the adjacent cells that lie on the coarse cell faces, as shown in Figure 5.9 (b).

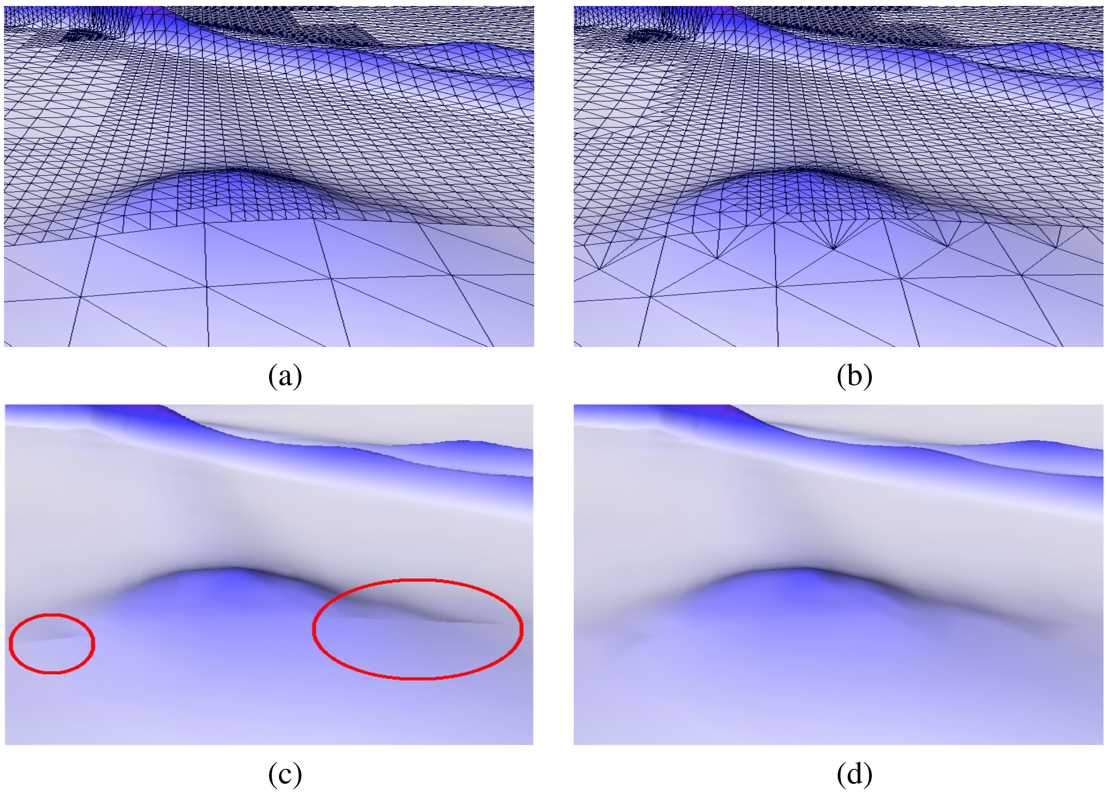

Figure 5.9: Image (a) shows a height field approximation by a triangular mesh that contains hanging nodes. These can lead to shading artifacts, as depicted in image (c). Replacing coarse triangles at level boundaries by strips of smaller triangles removes the hanging nodes (b) and the shading artifacts (d). *(dataset courtesy of M. Norman, National University of California)*

5.4.3 Isosurfaces

5.4.3.1 Cell-Centered Data

For cell-centered AMR data in combination with nearest-neighbor interpolation we generate the isosurface by primitives that are coplanar to one of the three coordinate planes. The construction is straight-forward:

- If two neighboring cells on the same level of resolution have associated data values below and above the iso-value, a surface element with the shape of the common face is generated.
- If the two cells belong to different levels of resolution, the smaller face segment of the higher resolved cell is chosen.

An example of a resulting surface is shown in the following Figure:

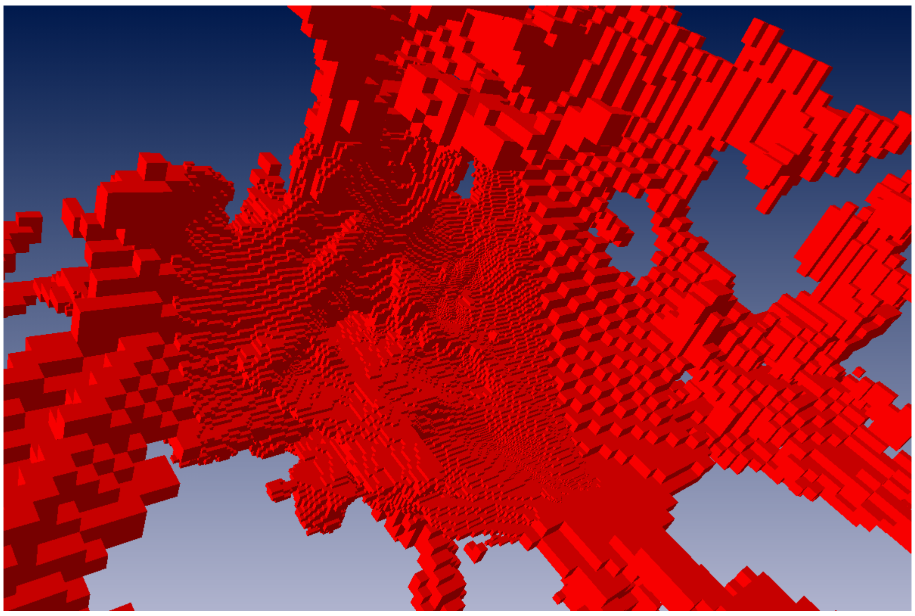

Figure 5.10: Isosurface for cell-centered AMR data. Surface primitives represent isosurface patches between two adjacent cells with data values below and above the iso-value. *(dataset courtesy of G. Bryan, Princeton University)*

5.4.3.2 Vertex-Centered Data

For vertex-centered data and trilinear interpolation the application of the standard marching cubes algorithm is problematic, since it generates artifacts at the level boundaries, visible as cracks in the isosurface, compare Figure 5.11 (a). In contrast to the display of height field surfaces discussed in the last subsection, the problem remains even for globally $\mathcal{C}^0$-continuous, piecewise trilinear interpolants.

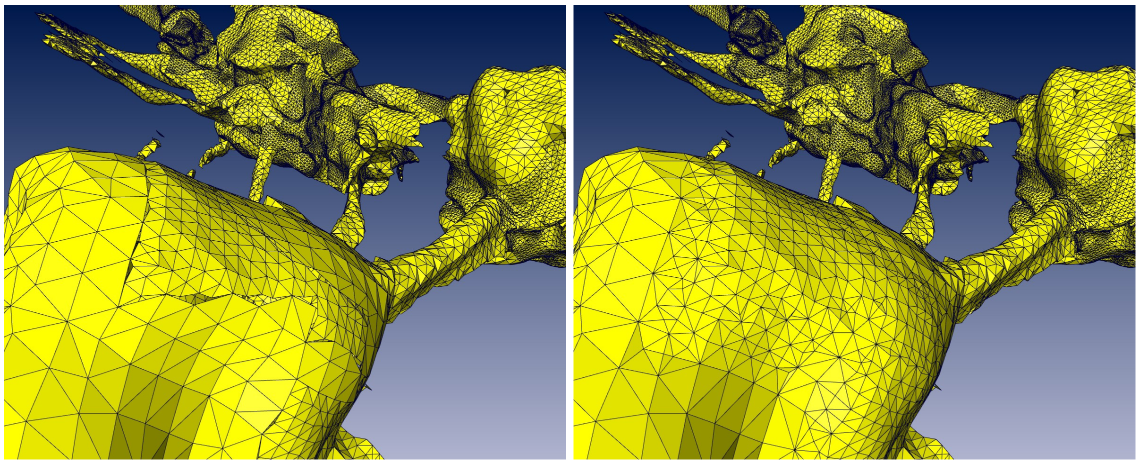
(a) (b)

Figure 5.11: The left image shows cracks in the triangulation generated by the marching cubes algorithm applied to an SAMR dataset. The right image shows a conforming mesh extracted from the hybrid grid that is obtained by replacing coarse cells at level boundaries by sets of tetrahedral and pyramidal cells. *(dataset courtesy of G. Bryan, Princeton University)*

The origin of the artifacts is the following: Depending on the signature of the face vertices, the one-dimensional isocontour of the intersection between the isosurface of the trilinear interpolant and the face consists of up to two hyperbolas, defined by

$$s(t) = \frac{\big(f(\boldsymbol{x}_{00}) - f(\boldsymbol{x}_{10})\big)t - f(\boldsymbol{x}_{00}) + v_{iso}}{\big(f(\boldsymbol{x}_{00}) - f(\boldsymbol{x}_{10}) + f(\boldsymbol{x}_{11}) - f(\boldsymbol{x}_{01})\big)t + f(\boldsymbol{x}_{01}) - f(\boldsymbol{x}_{00})},$$

where $\boldsymbol{x}_{ij}$ are the locations of the face nodes.

On the coarse face each of these hyperbolas is approximated by one line segment, whereas on the higher resolved faces a finer approximation is generated, since the bilinear interpolant is sampled at a larger number of locations, compare Figure 5.12 (a). Though for a continuous interpolant the starting-, and end-points of the line-sets, which lie on the edges of the coarse cell, coincide, the polylines will in general not be congruent in the interior of the face. This is only the case, if the hyperbolas degenerate to a straight line, i. e. if

$$f(\boldsymbol{x}_{00}) - f(\boldsymbol{x}_{10}) + f(\boldsymbol{x}_{11}) - f(\boldsymbol{x}_{01}) = 0,$$

which is holds if the two vectors

$$(1, 0, f(\boldsymbol{x}_{10}) - f(\boldsymbol{x}_{00})), (1, 0, f(\boldsymbol{x}_{01}) - f(\boldsymbol{x}_{11}))$$

are coplanar.

In case the values of the hanging nodes are not dependent nodes, the artifacts are even more severe. Consider for example a central hanging node with a signature that differs

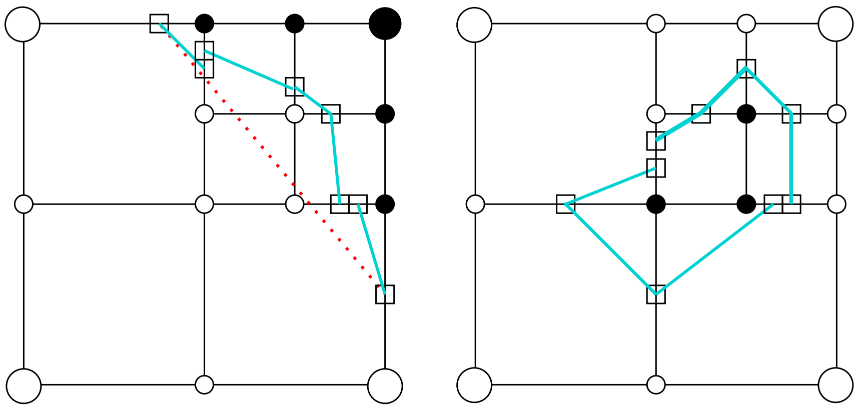
Figure 5.12: Reason for cracks at interfaces between fine and coarse grid cells: On the left side the isocontour on the coarse face is approximated by a straight line, whereas on the refined faces a larger number of isopoints (rectangles) is generated. In general these additional points do not lie on the coarse isoline, which leads to cracks in the resulting isosurface for the three-dimensional case. The figure on the right shows a situation for hanging nodes that are not dependent nodes. In this example no isocontour is generated on the coarse face, though a (discontinuous) isocontour results from the line segments of the refined faces.

from the one of the adjacent, non-hanging nodes. On the refined side the marching cubes algorithm would generate a closed isocontour, whereas on the coarse side no line-segment is present at all, as shown in Figure 5.12 (b).

5.4.3.3 Conforming Split

According to the discussion above we need a generalization of the marching cubes algorithm, that

- can handle non-restricted AMR hierarchies,
- can handle hanging nodes, that are not dependent nodes,
- and generates meshes without T-vertices, in order to avoid shading artifacts.

Due to these constraints we decided to generalize the conforming split approach discussed in [94] to handle locally refined hexahedral grids, with adjacent cells that differ by an arbitrary number of refinement levels.

Coarse cells that abut on a level boundary are replaced by two types of cells, namely tetrahedra and pyramids. Therefore a new node is generated in the center of these cells, with a data value obtained by trilinear interpolation. For faces that are shared by a cell on the same level of resolution, one pyramidal cell with the new vertex in the center as its apex and the face as its base is generated.

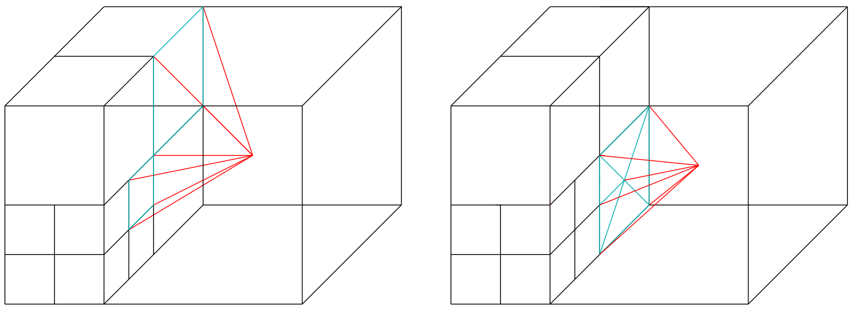
Figure 5.13: Illustration of the conforming split scheme. Pyramidal cells are employed to remove hanging nodes in the interior of the coarse face (left image), while tetrahedral cells are inserted according to the configuration of hanging nodes at the cell edges (right image).

For faces at the level boundary, a set of pyramids is generated, depending on the number of hanging nodes at the interior of the face. These pyramidal share the newly generated center node as a common apex, compare left image in Figure 5.13.

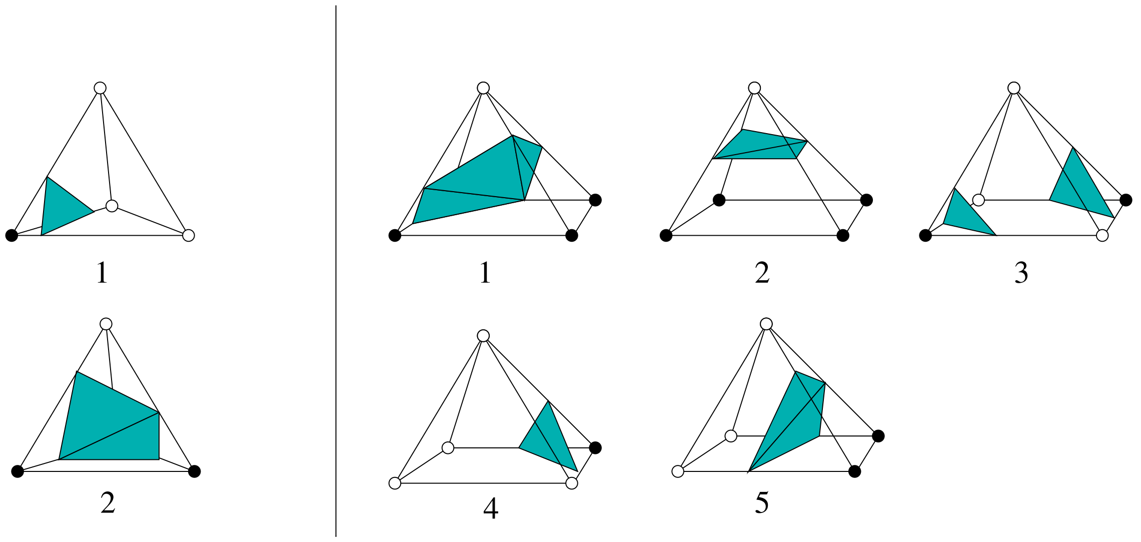

Figure 5.14: The 16 possible vertex configurations for tetrahedral cells can be reduced to 3 topologically different ones. Two of them lead to triangulations, as shown on the left side. Analogously the topologically different cases for pyramidal cells are shown on the right.

In the case that the hanging nodes that are not dependent nodes, it is further necessary to remove hanging nodes that are located on the edges of the cells. Depending on the configuration of these nodes the pyramidal cells are further split up into sets of tetrahedra as illustrated in the right part of Figure 5.13. Notice that this is necessary also for coarse cells that share only an edge with a refined cell.

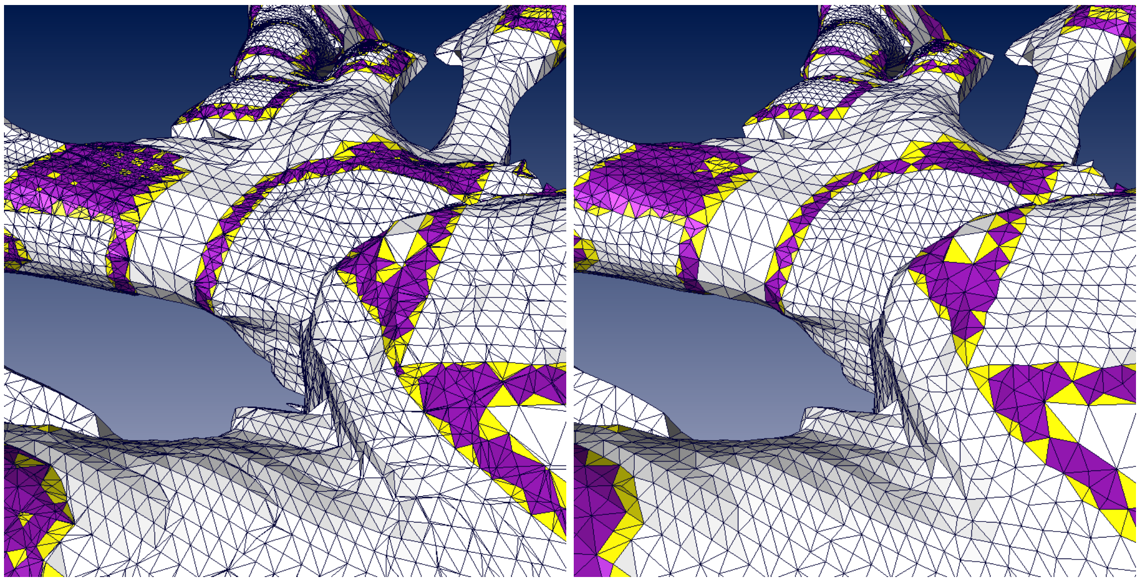

Figure 5.15: Triangles resulting from different cell types. White triangles - hexahedral cells; yellow triangles - tetrahedral cells; blue triangles - pyramid cells. The right image shows the simplification of the triangulation of the left side using the compact cubes algorithm [58]. *(dataset courtesy of G. Bryan, Princeton University)*

During the isosurface extraction step each cell is processed separately. The topologically different cases for tetrahedral and pyramidal cells are shown in Figure 5.14. Figure 5.15 shows a rendering, in which triangles generated from different types of cells are highlighted in different colors.

A drawback of the approach discussed so far is that due to the increased number of cell primitives in the hybrid grid, also the number of generated triangles is increased, if compared to isosurfaces (with cracks), that result from the unmodified, standard marching cubes. In order to reduce the number of triangles, we employ the *compact cubes* algorithms [58] in a post-processing step.

5.4.3.4 Results and Discussion

We applied the algorithm to two AMR datasets - the results from galaxy cluster formation simulations. The first one consists of 984 subgrids on 7 levels of refinement; the second one contains 1939 subgrids, distributed on 8 refinement levels. The measurements were performed on a 2.0 GHz PENTIUM4-SYSTEM equipped with a 128 MBytes ATI-RADEON 9200 graphics board. The results are given in Table 5.16.

	preprocessing	standard MC	standard CC	adaptive MC	adaptive CC
Dataset I	(7.0+5.0) s	107,000 (1.6 s)	58,000 (1.8 s)	164,000 (2.1 s)	72,000 (2.3 s)
Dataset II	(10.0+12.0) s	332,000 (4.4 s)	190,000 (4.6 s)	527,000 (4.8 s)	240,000 (5.0 s)

Figure 5.16: The table lists the preprocessing times for the detection of the location of hanging nodes, as well as the generation of the cell configurations for the conforming split, which has to be performed only once per dataset. The next three entries state the times for the actual extraction of the isosurface and the number of generated triangles for the standard marching cubes that only operates on the original hexahedral cells and leads to cracks at the boundaries, as well as for the adaptive variant discussed above, in both cases with and without triangle reduction utilizing the compact cubes approach.

The number of triangles resulting from the adaptive algorithm was about 60% larger if compared to the isosurfaces with cracks resulting from the standard marching cubes algorithm. Post-processing both surfaces using the compact cubes algorithm, reduced the increase to 25%. The extraction times where comparable for the different methods. Figure 5.17 shows a rendering example.

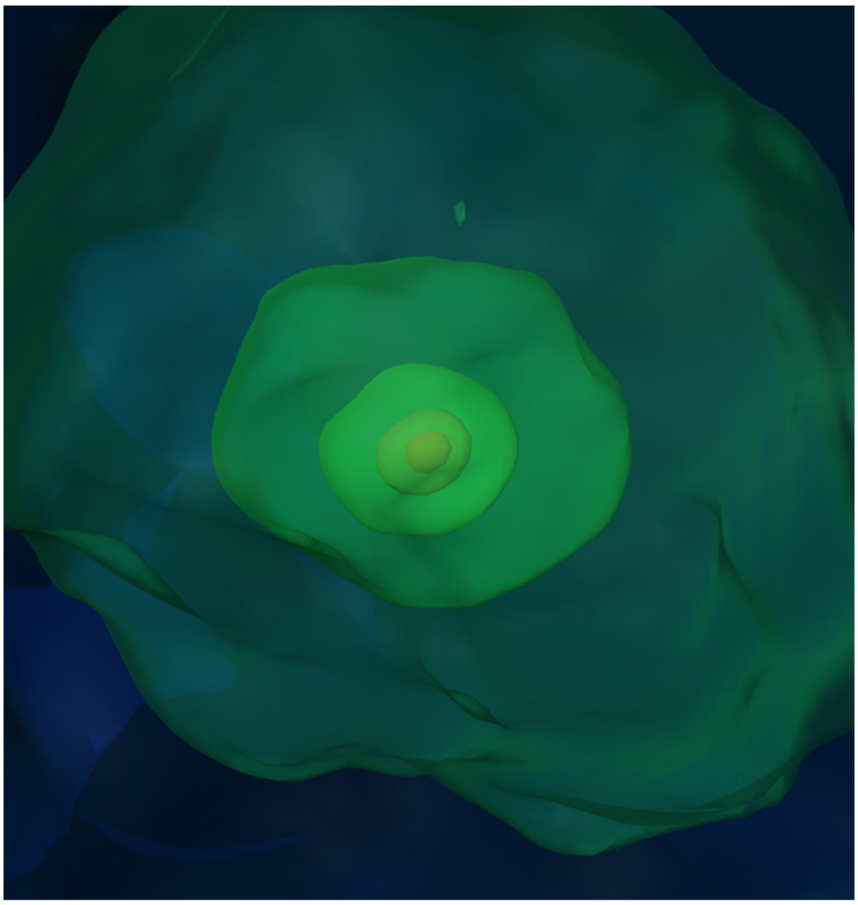

Figure 5.17: Nested semi-transparent isosurfaces depicting regions of constant gas density inside a proto-galaxy. *(dataset courtesy of T. Abel, Stanford University)*