Title
Extraction of Crack-Free Isosurfaces from Adaptive Mesh Refinement Data

Authors
Weber, Gunther H.
Kreylos, Oliver
Ligocki, Terry J.
et al.

Publication Date
2003

Peer reviewed

Extraction of Crack-free Isosurfaces from Adaptive Mesh Refinement Data

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

[1] Center for Image Processing and Integrated Computing (CIPIC), Department of Computer Science, One Shields Avenue, University of California, Davis, CA 95616-8562, U.S.A.
[2] AG Computergraphik, FB Informatik, University of Kaiserslautern, Erwin-Schrödinger Straße, D-67653 Kaiserslautern, Germany
[3] National Energy Research Scientific Computing Center (NERSC), Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, CA 94720, U.S.A.
[4] National Center for Supercomputing Applications (NCSA), University of Illinois, Urbana-Champaign, IL 61801, U.S.A.

Abstract. Adaptive mesh refinement (AMR) is a numerical simulation technique used in computational fluid dynamics (CFD). This technique permits efficient simulation of phenomena characterized by substantially varying scales in complexity. By using a set of nested grids of different resolutions, AMR combines the simplicity of structured rectilinear grids with the possibility to adapt to local changes in complexity within the domain of a numerical simulation that otherwise requires the use of unstructured grids. Without proper interpolation at the boundaries of the nested grids of different levels of a hierarchy, discontinuities can arise. These discontinuities can lead, for example, to cracks in an extracted isosurface. Treating locations of data values given at the cell centers of AMR grids as vertices of a dual grid allows us to use the original data values of the cell-centered AMR data in a marching-cubes (MC) isosurface extraction scheme that expects vertex-centered data. The use of dual grids also induces gaps between grids of different hierarchy levels. We use an index-based tessellation approach to fill these gaps with "stitch cells." By extending the standard MC approach to a finite set of stitch cells, we can define an isosurface extraction scheme that avoids cracks at level boundaries.

1 Introduction

AMR was introduced to computational physics by Berger and Oliger [3] in 1984. A modified version of their algorithm was published by Berger and Colella [2]. AMR has become increasingly popular in the computational physics community, and it is used in a variety of applications. For example, Bryan [4] uses a hybrid approach of AMR and particle simulations to simulate astrophysical phenomena.

Fig. 1 shows a simple two-dimensional (2D) AMR hierarchy produced by the Berger–Colella method. The basic building block of a d–dimensional Berger-Colella AMR hierarchy is an axis-aligned, structured rectilinear grid. Each grid g consists of n_j hexahedral cells in each axial direction. We treat this number as an integer resolution vector $\boldsymbol{n}$[1]. The grid spacing, i.e., the widths of grid cell in each dimension, is

[1] For convenience, we denote the j–th component of a vector $\boldsymbol{x}$ as x_j.

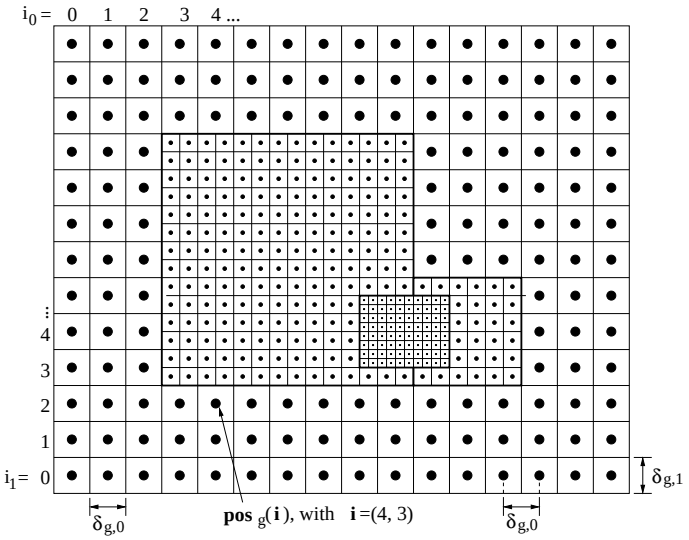


Fig. 1. AMR hierarchy consisting of four grids in three levels. The root level consists of one grid. This grid is refined by a second level consisting of two grids. A fourth grid refines the second level. It overlaps both grids of the second level. Boundaries of the grids are drawn as bold lines. Locations at which dependent variables are given are indicated by solid discs

constant in a specific direction and given as a vector $\boldsymbol{\delta}_g$, see Fig. 1. Fig. 1 illustrates that the distance between two samples is equal to the grid spacing. Each grid can be positioned by specifying its origin $\boldsymbol{o}_g$. The simulation method typically applied to AMR grids is a finite-difference method. Typically, a *cell-centered* data format is used, i.e., dependent function values are associated with the centers of cells. Thus, the dependent function value associated with a cell $\boldsymbol{i}_g$, with $0 \leq i_{g,j} < n_j$, is located at

$$pos_{g,j}(\boldsymbol{i_g}) = o_{g,j} + \left(i_j + \frac{1}{2}\right)\delta_j \quad , \tag{1}$$

see Fig. 1. Since sample locations are implicitly given by the regular grid structure, it suffices to store dependent data values in a simple array using a fixed ordering scheme, e.g., row-major order. We denote the region covered by a grid g by Γ_g.

An AMR hierarchy consists of several levels Λ_l comprising one or multiple grids. All grids in the same level have the same resolution, i.e., all grids in a level share the same cell width vector $\boldsymbol{\delta}_g = \boldsymbol{\delta}_{\Gamma_l}$. The region covered by a level Γ_{Λ_l} is the union of regions covered by the grids of that level. In most AMR data sets, only the root level covers a contiguous region in space, while all other levels typically consist of several disjoint regions.

The hierarchy starts with the *root level* Λ_0, the coarsest level. Each level Λ_l may be refined by a finer level Λ_{l+1}. A grid of the refined level is commonly referred to as a *coarse grid* and a grid of the refining level as a *fine grid*. The *refinement ratio*

r_l specifies how many cells of a fine grid contained in level l fit into a coarse-grid cell along each axial direction. The refinement ratio is specified as a positive integer rather than a vector, as it is usually the same for all axial directions. A refining grid can only refine complete grid cells of the parent level, i.e., it must start and end at the boundaries of grid cells of the parent level. A refining grid refines an entire level Λ_l, i.e., it is completely contained in Γ_{Λ_l} but not necessarily in the region covered by a single grid of that level. (This is illustrated in Fig. 1, where the grid comprising the second level overlaps both grids of the first level.) Thus, in many cases it is

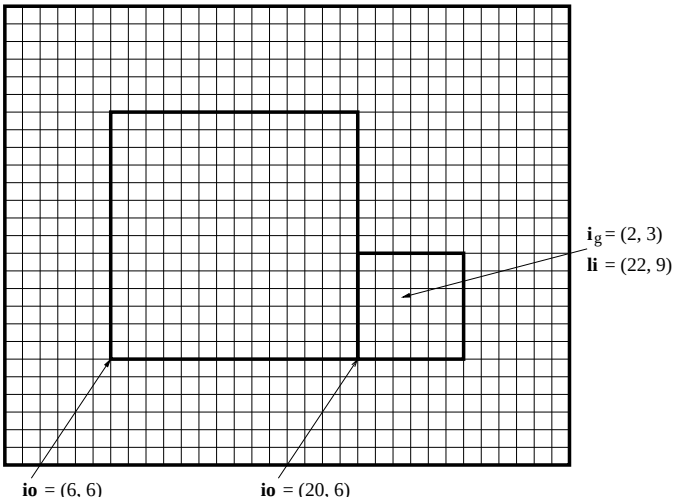


Fig. 2. To specify the index of a cell with respect to a level rather than a single grid, we assume that an entire level is covered by a level grid with a cell width equal to $\boldsymbol{\delta}_{\Gamma_l}$. Individual grids within a level cover rectangular sub-regions of that level grid. An index of a grid cell can either be specified with respect to the level grid ($\boldsymbol{li}$) or with respect to a grid containing that cell ($\boldsymbol{i}_g$). The integer origin of a grid is the level index of the grid cell with index **0** within the grid

convenient to access grid cells on a per-level basis instead of a per-grid basis.

To obtain the index of a cell within a level, we assume that a *level grid* with a cell width equal to $\boldsymbol{\delta}_{\Lambda_l}$ covers the entire domain Γ_{Λ_0}, see Fig. 2. This level grid starts at the minimum extent of a bounding box surrounding the root level $\boldsymbol{o}_{\Lambda_0}$, where

$$o_{\Lambda_0,j} = \min\{o_{g,j} | \boldsymbol{o}_g \in \Lambda_0\} \quad , \tag{2}$$

as shown in Fig. 3. Since all grids in a level are placed with respect to boundaries of grid cells in a parent level, grid cells in the level grid coincide with grid cells in a grid of that level or are outside the region covered by the level Γ_{Λ_l}. The *level index* $\boldsymbol{li}$ of a grid cell is its index in the level grid. Using the level index, the origin of a grid in a level can be defined as an *integer origin*. The integer origin $\boldsymbol{io}_g$ of a grid g

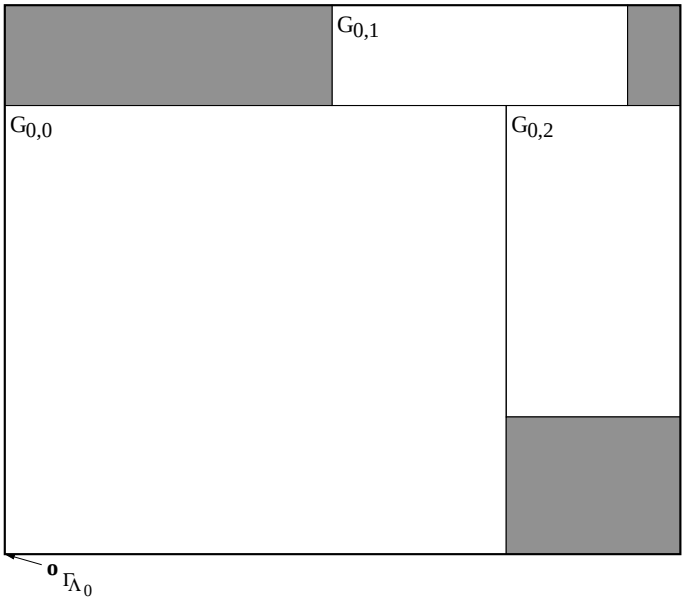


Fig. 3. The AMR hierarchy shown in this figure contains three grids in the root level. Regions that are not covered by any grid are shaded. The origin of a level grid is the minimum extent of the bounding box enclosing all grids in the root level

is the level index of the cell with index $\mathbf{0}$ within the grid, see Fig. 2. Its components are defined by

$$io_{g,j} = \frac{(o_{g,j} - o_{\Lambda_0,j})}{\delta_{\Gamma_l}} \quad \text{, where } g \in \Lambda_l \quad . \tag{3}$$

Since all grid cells of a fine grid must start and end at boundaries of cells of a coarse level, the components of the integer origin $io_{g,j}$ and the number of cells $n_{g,j}$ along an arbitrary axial direction of a grid g belonging to level l are always integer multiples of r_l. Individual grids in a level correspond to rectangular sub-regions of the level grid. To access a cell with a given level index $\boldsymbol{li}$, first the grid g that contains that level index has to be found, provided that such a grid exists. Second, the index $\boldsymbol{i}_g$ of the cell within that grid is obtained by subtracting this grid's integer origin $\boldsymbol{io}_g$ from $\boldsymbol{li}$.

Due to the hierarchical nature of AMR simulations, AMR data lend themselves to hierarchical visualization. One of the problems encountered when isosurfaces are extracted using an MC method is the cell-centered AMR format. MC methods expect dependent data values at a cell's vertices instead of data associated with a cell's center. If a re-sampling step is used to replace the values at a cell's center with values at its vertices, "dangling" nodes arise at level boundaries. These dangling nodes can cause cracks in the isosurface when using an MC method, even if a consistent interpolation scheme is used, i.e., one that assigns the same value to a dangling node

as is assigned to its location in the coarse level. To avoid re-sampling, we interpret the locations of the samples, see Eq. 1, as vertices of a new grid that is "dual" to the original one. This dual grid is then used in a standard MC approach.

The use of dual grids creates gaps between grids of different hierarchy levels. We fill those gaps in an index-based stitching step. Vertices, edges and faces of a fine grid are connected to vertices in the coarse level using a look-up table (LUT) for the possible refinement configurations. The resulting stitch mesh consists of tetrahedra, pyramids, triangular prisms and hexahedral cells. By extending an MC method to these additional cell types, we define an isosurface extraction scheme that avoids cracks in an isosurface at level boundaries.

The original Berger–Colella scheme [2] requires a layer with a width of at least one grid cell between a refining grid and the boundary of the refined level. Even though Bryan [4] eliminates this requirement, we still require it. This is necessary, as this requirement ensures that only transitions between a coarse level and the next finer level occur in an AMR hierarchy. Allowing transitions between arbitrary levels would force us to consider a large number of cases during the stitching process. (The number of cases would be limited only by the number of levels in an AMR hierarchy, since transitions between arbitrary levels are possible.) These requirements are equivalent to requirements described by Gross et al. [7] who also do not permit transitions between arbitrary levels.

2 Related Work

Relatively little research has been published regarding the visualization of AMR data. Norman et al. [14] convert an AMR hierarchy into finite-element hexahedral cells with cell-centered data that can be handled by standard visualization tools (like AVS [1], IDL [8], or VTK [15]), while preserving the hierarchical nature of the data. Ma [10] describes a parallel rendering approach for AMR data. Even though he re-samples the data to vertex-centered data, he still uses the hierarchical nature of AMR data and contrasts it to re-sampling it to the highest level of resolution available. Max [11] describes a sorting scheme for cells for volume rendering, and uses AMR data as one application of his method. Weber et al. [17] present two volume rendering schemes for AMR data. One scheme is a hardware-accelerated renderer for data previewing. This renderer partitions an AMR hierarchy into blocks of equal resolution and renders the complete data set by rendering blocks in back-to-front order. Their other scheme is based on cell projection and allows progressive rendering of AMR hierarchies. It is possible to render an AMR hierarchy starting with a coarse representation and refining it by subsequently integrating the results from rendering finer grids.

Isosurface extraction is a commonly used technique for visual exploration of scalar fields. Our work is based on the MC method, introduced by Lorensen and Cline [9], where a volume is traversed cell-by-cell, and the part of the isosurface within each cell is constructed using an LUT. The LUT in the original paper by Lorensen and Cline contained a minor error that can produce cracks in the extracted

isosurface. This is due to ambiguous cases where different isosurface triangulations in a cell are possible. Nielson and Hamann [12], among others, addressed this problem and proposed a solution to it. Van Gelder and Wilhelms [6] provided a survey of various solutions that were proposed for this problem. They showed that, in order to extract a topologically correct isosurface, more than one cell must be considered at a time. If topological correctness of an isosurface is not required, it is possible to avoid cracks without looking at surrounding cells. In our implementation, we use the LUT from VTK [15] that avoids cracks by taking special care during LUT generation.

Octree-based methods can be used to speed up the extraction of isosurfaces. Shekhar et al. [16] use an octree for hierarchical data representation. By adaptively traversing the octree and merging cells that satisfy certain criteria, they reduce the amount of triangles generated for an isosurface. Their scheme removes cracks in a resulting isosurface by moving fine-level vertices at boundaries to a coarser level to match up contours with the coarse level. Westermann et al. [19] modified this approach by adjusting the traversal criteria and improving the crack-removal strategy. Nielson et al. [13] use coons patches to avoid cracks between adjacent regions of different resolution. Instead of avoiding T-intersections or matching up contours they develop a Coons-patch based interpolation scheme for coarse-level cells that avoids cracks. Gross et al. [7] use a combination of wavelets and quadtrees to approximate surfaces, e.g., from terrain data. Considering an estimate based on their wavelet transform, their approach determines a level in the quadtree structure to represent a given region with a specific precision. Handling transitions between quadtree levels is similar to handling those between levels in an AMR hierarchy. Weber et al. [18] have introduced grid stitching to extract isosurfaces from AMR data sets. In this paper, we extent and elaborate on this work.

3 Dual Grids

The MC method assumes that data values are associated with cell vertices, but the prototypical AMR method produces values at cell centers. One possibility to deal with this incompatibility problem is to re-sample a given data set to a vertex-centered format. However, re-sampling causes “dangling nodes” in the fine level. Even if the same values are assigned to the dangling nodes as the interpolation scheme assigns to their location in the coarse level, dangling nodes can cause cracks when the MC method is applied, see [19]. We solve these problems by using a *dual grid* for isosurface extraction. This dual grid is defined by the function values at the cell centers, see Eq. 1. The implied connectivity information between these centers is given by the neighborhood configuration of the original cells. Cell centers become the vertices of the vertex-centered dual grid. The indices of a cell with respect to a grid or a level (level index), defined in Section 1, become indices of vertices in the dual grid.

The dual grids for the first two levels of the AMR hierarchy shown in Fig. 1 are shown in Fig. 4. We note that the dual grids have “shrunk” by one cell in each axial

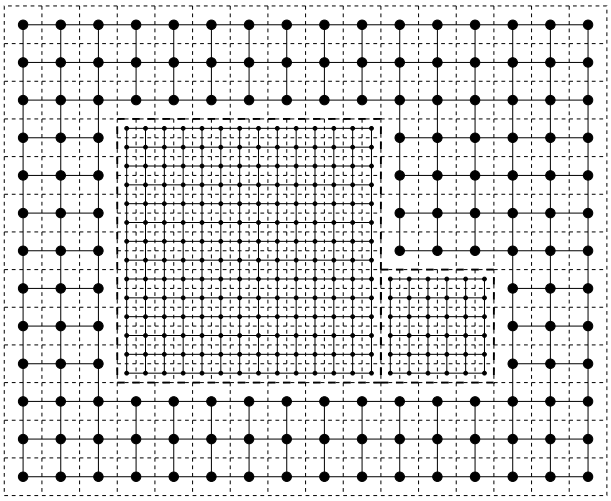


Fig. 4. Dual grids for the three AMR grids comprising the first two hierarchy levels shown in Fig. 1. The original AMR grids are drawn in dashed lines and the dual grids in solid lines

direction with respect to the original grid. The result is a gap between the coarse grid and the embedded fine grids. Due to the existence of this gap, there are no dangling nodes causing discontinuities in an isosurface. However, to avoid cracks in extracted isosurfaces as a result of gaps between grids, a tessellation scheme is needed that "stitches" grids of two different hierarchy levels.

4 Stitching 2D Grids

A *stitch mesh* used to fill gaps between different levels in the hierarchy is constrained by the boundaries of the coarse and the fine grids. In order to merge levels seamlessly, the stitch mesh must not subdivide any boundary elements of the existing grids. In the 2D case, this is achieved by requiring that only existing vertices are used and no new vertices generated. Since one of the reasons for using the dual grids is to avoid the insertion of new vertices, whenever possible, this causes no problems.

In the 2D case, a constrained Delaunay triangulation, see, for example, Chew [5], can be used to fill the gaps between grids. For two reasons, we do not do this. While in the 2D case only edges must be shared between the stitching grid and the dual grids, entire faces must be shared in the 3D case. The boundary faces of rectilinear grids are rectangles that cannot be shared by tetrahedra without being subdivided, thus causing cracks when used in an MC-based isosurface extraction scheme. Furthermore, an index-based approach is more efficient, since it takes advantage of the regular structure of the boundaries while avoiding problems that might be caused by this regular structure when using a Delaunay-based approach.

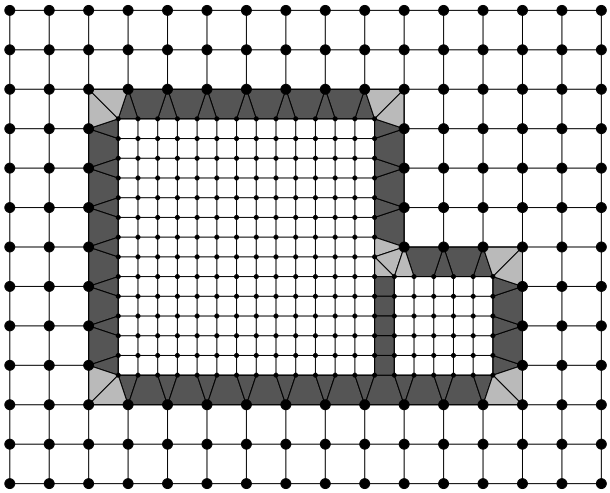


Fig. 5. Stitch cells for first two levels of AMR data set shown in Fig. 4

The stitching process for a refinement ratio of two is shown in Fig. 5. Stitch cells are generated for edges along the boundary and for the vertices of the fine grid. The stitch cells generated for the edges are shown in dark grey, while the stitch cells generated for the vertices are drawn in light grey. For the transition between one fine and one coarse grid, each edge of the fine grid is connected alternatingly to either a vertex or an edge of the coarse grid. This yields triangles and quadrilaterals as additional cells. The quadrilaterals are not subdivided, since subdivision is not unique. (This in turn would cause problems in the 3D case when quadrilaterals become boundary faces shared between cells.) The vertices are connected to the coarse grid via two triangles. A consistent partition of the deformed quadrilateral is possible. The obvious choice is to connect each edge to the two coarse edges that are “visible” from it.

In the case of multiple grids, a check must be performed: Are the grid points in the coarse grid refined or not? If a fine edge is connected to a coarse point, this check is simple. If the coarse point is refined, the fine edge must be connected to another fine edge; this yields a rectilinear instead of a triangular cell. The case of connecting to a coarse edge is more complicated and is illustrated in Figs. 6 (i)–(iv). If both points are refined, see Fig. 6(iv), the fine edge is connected to another fine edge. As a result, adjacent fine grids yield the same cells as a “continuous” fine grid. Problem cases occur when only one of the points is refined, see Figs. 6(ii) and 6(iii). Even though it is possible to skip these cases and handle them as vertex cases of the other grid, a more consistent approach is to include them in the possible edge cases. However, the same tessellations should be generated for both cases, as shown in Fig. 6.

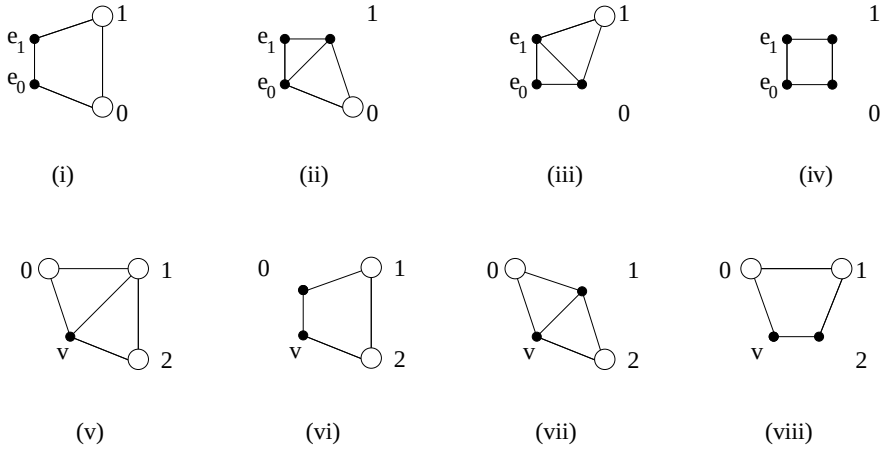


Fig. 6. Possible cases for connecting a boundary edge $\overline{e_0 e_1}$, cases (i)–(iv), or a boundary vertex v, cases (v)–(viii), to a coarse grid. If cells of the coarse grid are refined, the coarse grid points (circles) are replaced by the corresponding refining point (solid black discs)

The cases arising from connecting a vertex are illustrated in Figs. 6 (v)-(vii). In addition to replacing refined coarse grid points by the nearest fine-grid point, adjoining grids must be merged. If either of the coarse grid points 0, see Fig. 6(v), or 2, see Fig. 6(v), is refined, it is possible to change the border vertex to a border-edge segment by connecting it to the other refined grid point and treating it as an edge, and using the connection configurations from the previous paragraph, i.e., those shown in Figs. 6 (i)–(iv). (This case occurs along the bottom edge of the fine grids shown in Fig. 5.)

Even though arbitrary integer-refinement ratios r_l are possible for AMR grids, refinement ratios of two and four are the most common ones used. The stitching process can be generalized to more general refinement ratios. Instead of connecting edge segments of the refining grid alternatingly to a coarse-grid edge segment and point, $(r_l - 1)$ consecutive edge segments must be connected to one common coarse-grid point. Every r_l–th fine edge must be connected to a coarse edge. The same connection strategy results from connecting each fine grid edge to a parallel "phantom edge" that would exist if the grid continued in that direction. If both end points of the phantom edge are within the same grid cell in the parent level, the edge is connected to the coarse grid point within that cell. If the phantom edge crosses a boundary between two coarse grid cells, the fine edge is connected to the edge formed by the two grid points in those cells.

Even though the valence of the grid points of the coarse grid is increased, this is not a problem with the commonly used refinement ratios. Furthermore, it is important to note that arbitrary refinement ratios would not add more refinement config-

urations. The fundamental connection strategies remain the same. A fine-grid edge is connected to a coarse-grid vertex or a coarse-grid edge. A fine-grid vertex is connected to two coarse-grid edges. The cell subdivisions shown in Fig. 6 can be used for arbitrary refinement ratios.

5 Stitching 3D Grids

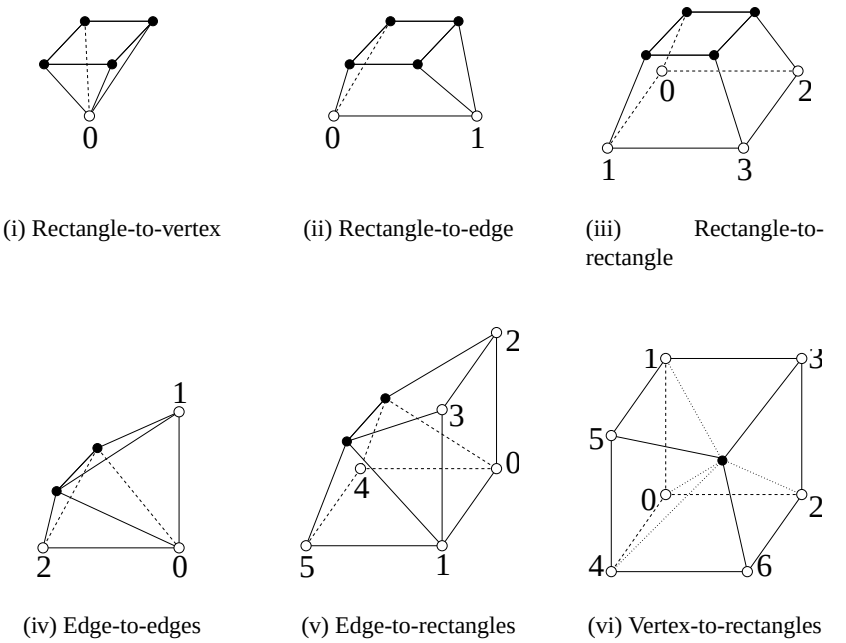
Our index-based approach can be generalized to 3D AMR grids. In the simple case of one fine grid embedded in a coarse grid, boundary faces, edges and vertices of the fine grid must be connected to the coarse grid.

Each of the six boundary faces of a grid consists of a number of rectangles defined by four adjacent grid points on the face. A boundary face is connected to the coarse mesh by connecting each of its comprising rectangles to the coarse grid. For each quadrilateral, the level indices of the four grid points that would extend the grid in normal direction are computed. These are transformed into level indices of the parent level by dividing them by the refinement ratio r_l of the fine level. In each of the two directions implied by a rectangle, these transformed points may have the same level index component. If they have the same index for a direction, the fine rectangle must be connected to one vertex in this direction; otherwise, it must be connected to an edge in that direction. The result is the same as the combination of two 2D edge cases. The various combinations result in rectangles being connected to either a vertex, a line segment (in the two possible directions) or another rectangle. The cell types resulting from these connections are pyramids, see Fig. 7(i), deformed triangle prisms, see Fig. 7(ii), and deformed hexahedral cells, see Fig. 7(iii).

An edge is connected to the coarse grid by connecting its comprising edge segments to the grids in the parent level. For each segment, the level indices of the six grid points that would extend the grid beyond the edge are computed. These indices are also transformed into level indices of the parent level. Depending on whether the edge segment crosses a boundary face of the original AMR grid or not, the edge must either be connected to three perpendicular edges or two rectangles of the coarse grid. This is equivalent to a combination of the vertex and edge connection types of the 2D case. If the viewing direction is parallel to the edge segment (such that it appears to the viewer as a point), it must always be connected to two perpendicular edges of the coarse grid. In the direction along the edge, one connects it to a point or a parallel edge. Connecting an edge segment to the coarse grid results in two tetrahedra, shown in Fig. 7(iv), or two deformed triangle prisms, shown in Fig. 7(v), as connecting cells.

A vertex is connected by calculating the level indices of the seven points that would extend the grid. These are transformed into level indices of the parent level. The result is the same as the combination of two 2D vertex cases. The vertex is connected to three rectangles of the coarse grid via pyramid cells, as shown in Fig. 7(vi).

When the coarse level is refined by more than one fine grid, one must check each coarse-grid point for refinement and adapt the generated tessellation accordingly.



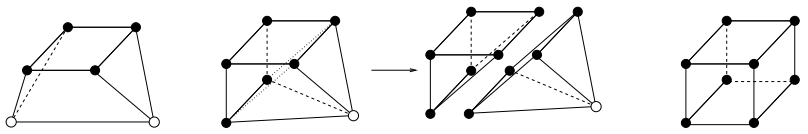
(i) Rectangle-to-vertex (ii) Rectangle-to-edge (iii) Rectangle-to-rectangle

(iv) Edge-to-edges (v) Edge-to-rectangles (vi) Vertex-to-rectangles

Fig. 7. Possible connection types for quadrilateral, edge and vertex in 3D case

The simplest case is given when a fine grid boundary rectangle is connected to a coarse point, see Fig. 7(i). If this coarse grid point is refined an adjacent fine grid exists and the fine grid boundary rectangle must be connected to the other fine grid's boundary rectangle. This case illustrates that it is helpful to retain the indices within the fine level in addition to converting them to the coarse level. In the unrefined case, the rectangle was connected to a coarse point, because transforming the four level indices from the fine level to the coarse level yielded the same coarse level index. If the grid point corresponding to that coarse level index is refined, the "correct" fine boundary rectangle can be determined using the original fine level indices. (For a refinement ration of $r_l = 2$ the correct choice of the fine-level rectangle is also implied by the connection type, but for general refinement ratios the fine-level index must be retained.)

Connecting a fine rectangle to a coarse edge, see Fig. 7(ii), is slightly more difficult. Each of the endpoints of a coarse edge can either be refined or unrefined. The resulting refinement configurations and their tessellations are shown in Fig. 8. Refinement configurations, the cases in Fig. 8 and subsequent figures are numbered as follows: For each connection type shown in Fig. 7, the coarse-grid grid points that are connected to a fine grid element are numbered according to the corresponding sub-figure. A case number is obtained by starting with case 0. For each refined



(i) Case 0 (ii) Cases 1 and 2 (iii) Case 3

Fig. 8. Refinement configurations for connecting a fine-grid rectangle to a coarse-grid edge

coarse-grid vertex k, 2^k is added to the number associated with this case. To determine to which refining grid points the fine rectangle should be connected we retain the fine-level indices in addition to converting them to coarse level indices.

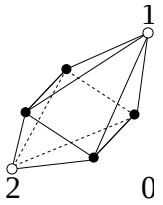
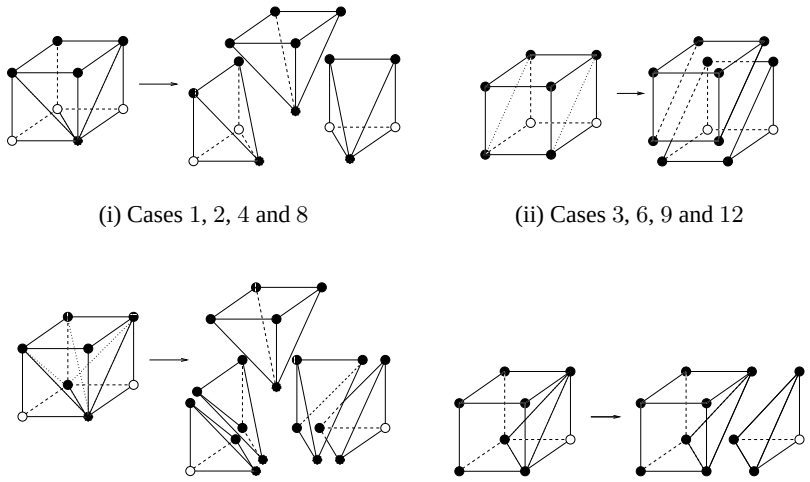


Fig. 9. If coarse-grid vertex 0 is refined, when a fine-grid edge is connected to two coarse-grid edges, see Fig. 7(iv), two pyramids are generated instead of two tetrahedra

When connecting a fine-grid edge to two perpendicular coarse-grid edges, eight refinement configurations arise. If all coarse-grid vertices are unrefined, two tetrahedra are generated, as illustrated in Fig. 7(iv). If either coarse-grid vertex 1 or 2 is refined, the fine-grid edge must be upgraded to a coarse-grid rectangle and the corresponding tessellation function called with appropriate vertex ordering. (This procedure ensures that adjacent fine grids produce the same stitch tessellation as a continuous fine grid.) In the remaining case, when only coarse-grid vertex 0 is refined, two pyramids are generated instead of tetrahedra, see Fig. 9.

For connection types where a fine-grid quadrilateral, see Fig. 7(iii), edge, see Fig. 7(v), or vertex, see Fig. 7(v), is connected to coarse-grid rectangles, eight points are considered. These points form a deformed hexahedral cell. One must consider 16 possible refinement configurations when a fine-grid rectangle is connected, since the four vertices belonging to the fine-grid rectangle are always refined, see Fig. 10. More cases arise when a fine-grid edge or a vertex is connected to the coarse level. It is important to devise an efficient scheme to determine the tessellation for a given refinement configuration. Each cell face corresponds to a possible 2D refinement configuration as shown in Fig. 6. It is important to note that the 2D refinement configurations that produce subdivided quadrilaterals are the same configurations



(i) Cases 1, 2, 4 and 8

(ii) Cases 3, 6, 9 and 12

(iii) Cases 5 and 10

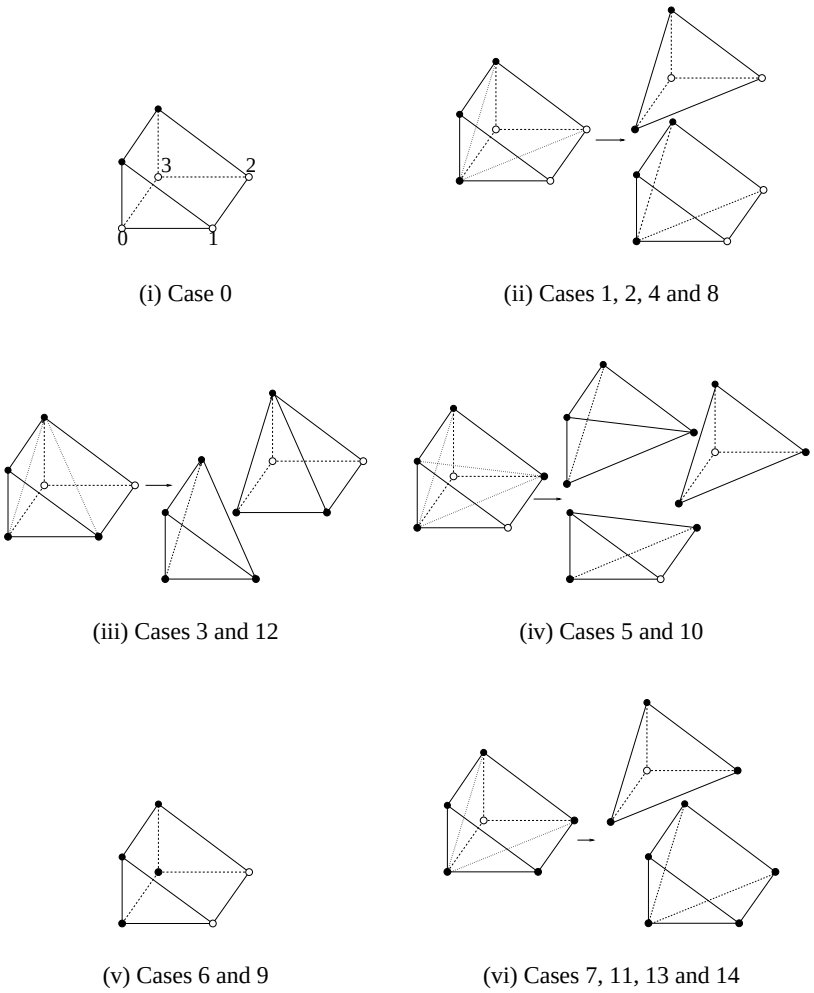
(iv) Cases 7, 11, 13 and 14

Fig. 10. Refinement configurations for connecting a fine-grid rectangle to a coarse-grid rectangle

that yield non-planar cell boundaries, i.e., boundaries that we must subdivide. The subdivision information alone is sufficient to determine a tessellation for any given refinement configuration. It is not necessary to consider the positions of the points. Each cell face Fig. 10 and subsequent figures is subdivided using the canonical tessellations depicted in Fig. 6, illustrated by the dotted lines in a figure illustrating tessellations of a connection type, e.g., Fig. 10.

The subdivision of cell faces implies subdivisions of hexahedral cells into pyramids, triangular prisms and tetrahedra. In certain cases, see, for example, Fig. 10(iv), a cell type arises that does not correspond to the standard cells (hexahedra, pyramids, triangle prisms and tetrahedra), and that cannot be subdivided further without introducing additional vertices. Even though it is possible to generate a case table to extend MC to this cell type, the asymmetric form of this cell makes this extension difficult. Symmetry considerations that are used to reduce the number of cases that need to be considered cannot be applied. Therefore, we handle cells of this type by generating an additional vertex at the centroid of the cell. By connecting the centroid to all cell vertices we obtain a tessellation consisting of pyramids and tetrahedra.

Edges are connected to the coarse grid by considering eight vertices forming a deformed hexahedral cell. When an edge is connected, two of these eight points belong to the edge. The other six vertices are coarse-grid vertices and can be either refined or unrefined. Thus, it is necessary to consider 64 possible refinement configurations. It is necessary to consider all six coarse grid points at once, since



(i) Case 0

(ii) Cases 1, 2, 4 and 8

(iii) Cases 3 and 12

(iv) Cases 5 and 10

(v) Cases 6 and 9

(vi) Cases 7, 11, 13 and 14

Fig. 11. Possible refinement configurations and corresponding tessellations for triangle prism cell

the boundary faces need not always be subdivided faces, as Fig. 7(v) implies. If, for example, all coarse grid points are refined, a single, not-tessellated cell must be produced. However, in certain refinement situations (Cases 0–3, 5, 7, 10, 11, 15, 17, 19, 27, 34, 35, 39, 51), the two cell faces perpendicular to the fine-grid edge segment must be divided as shown in Fig. 7(v). In these cases it is possible to connect the fine-grid edge segment to the coarse grid by handling each of the triangle prisms separately. Each triangle prism is tessellated according to the refinement con-

figurations shown in Fig. 11. When connecting a fine-grid edge to two coarse-grid rectangles, see Fig. 7(iii), it must be upgraded to to the rectangle case if either grid points 2 and 3 or grid points 4 and 5 are refined. In these cases (Cases 12–15, 28–31 and 44–63) the refinement configurations for connecting a fine-grid rectangle are used according to Fig. 10. The tessellations for the remaining cases are shown in Fig. 12. In Fig. 12, we only considered symmetry with respect to a plane perpendicular to the edge to reduce the number of cases. The cases in Figs. 12(ix) and 12(x) differ only by rotation but yield the same tessellation. We note that only the partition of the boundary faces matters in determining a valid tessellation. Even though the refinement configurations shown Figs. 12(iv) and 12(vi) differ, they yield the same partition of a cell's boundary faces and thus the same tessellation.

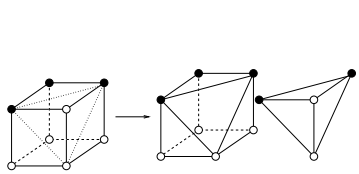
Vertices can be upgraded to edges, or even quadrilaterals, when more than two grids meet at a given location. The fine vertex shown in Fig. 7(vi) can be changed to an edge, if any of the coarse grid points 3, 5 or 6 is refined. In these cases, the procedure used to connect an edge segment is called with appropriate vertex ordering. As a result, the same tessellations are used as in the case of a continuing edge. Furthermore, this procedure ensures that an additional upgrade to the rectangle case is handled automatically when needed. In the remaining cases, each of the pyramids of the unrefined case can be handled independently. If the base face of a pyramid is not planer, it is subdivided using the corresponding configuration from Fig. 6, and the pyramid is split into two tetrahedra.

6 Isosurface Extraction

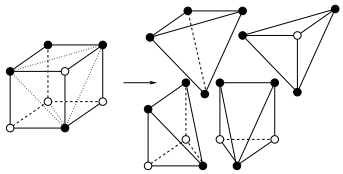
Within individual grids, we apply a slightly modified MC approach for isosurface extraction. Instead of considering all cells of a grid for isosurface generation, we consider only those cells that are not refined by a finer grid. We do this by precomputing a map with refinement information for each grid. For each grid cell, this map contains an index of a refining grid or an entry indicating that the cell is unrefined. This enables us to quickly skip refined portions of the grid. For the generation of an isosurface within stitch cells, the MC method must be extended to handle the cell types generated during the stitching process. This extension is achieved by generating case tables for each of the additional cell types. These new case tables must be compatible with the one used in the standard MC approach, i.e., ambiguous cases, see Section 2 must be handled in exactly the same way as handled for typical hexahedral cells.

7 Results

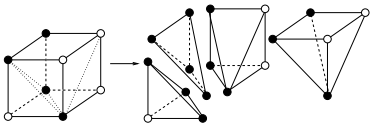
Fig. 13 shows isosurfaces extracted from an AMR data set. This data set is the result from an astrophyiscal simulation of star clusters performed by Bryan [4]. The isosurface in Fig. 13(i) shows an isosurface extracted from two levels of the hierarchy, and Fig. 13(ii) one extracted from three levels. To highlight the transitions between levels, the parts of the isosurface extracted from different levels of the



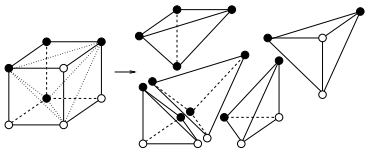
(i) Cases 4, 8, 16 and 32



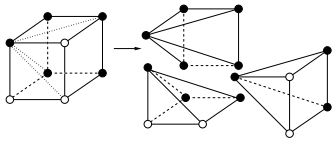
(ii) Cases 6 and 9



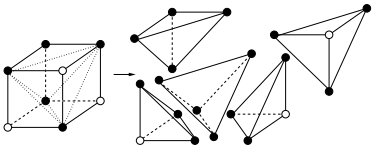
(iii) Cases 18 and 33



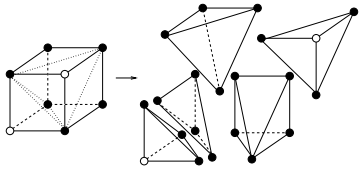
(iv) Cases 20 and 40



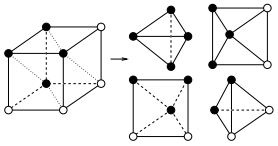
(v) Cases 21 and 42



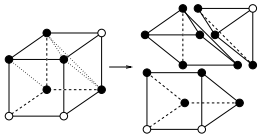
(vi) Cases 22 and 41



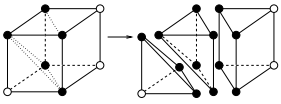
(vii) Cases 23 and 43



(viii) Cases 24 and 36



(ix) Cases 25 and 38



(x) Cases 26 and 37

Fig. 12. Remaining refinement configurations for connecting fine-grid edge to coarse-grid rectangles

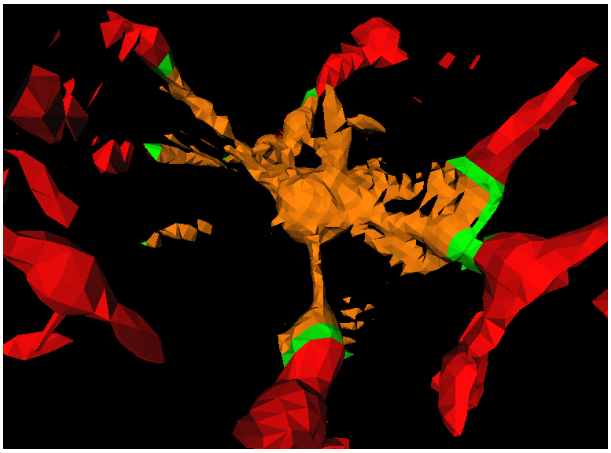
hierarchy are colored differently. Isosurface parts extracted from the root, the first and the second levels are colored red, orange and light blue, respectively. Portions extracted from the stitch meshes between the root and the first level are colored in green, and portions extracted from the stitch meshes between the first and second levels are colored in yellow. The root level and the first level of the AMR hierarchy each consist of one $32 \times 32 \times 32$ grid. The second level consists of 12 grids of resolutions $6 \times 12 \times 6$, $6 \times 4 \times 2$, $8 \times 12 \times 10$, $6 \times 4 \times 4$, $14 \times 4 \times 10$, $6 \times 6 \times 12$, $12 \times 10 \times 12$, $10 \times 4 \times 8$, $6 \times 6 \times 2$, $16 \times 26 \times 52$, $14 \times 16 \times 12$ and $36 \times 52 \times 36$. All measurements were performed on a standard PC with a $700\mathrm{MHz}$ Pentium III processor.

8 Conclusions and Future Work

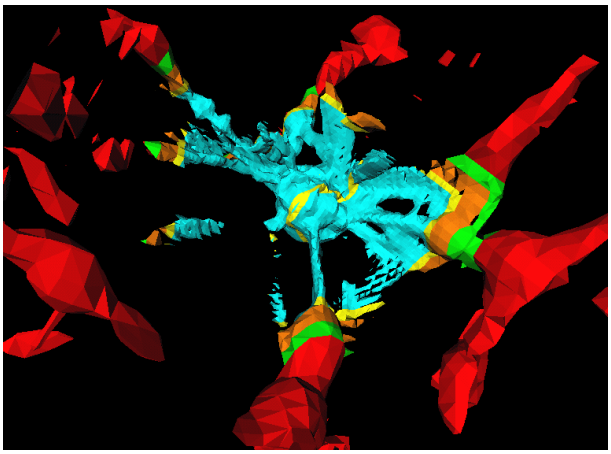
We have presented a method for the extraction of crack-free isosurfaces from AMR data. By using a dual-grid approach and filling gaps with stitch cells we avoid re-sampling of data and dangling nodes. By extending the standard MC method to the cell types resulting from grid stitching, we have developed an isosurfacing scheme that produces consistent and seamless isosurfaces. Theoretically any continuous (scattered data) interpolation scheme could also lead to a crack free isosurface. However, the use of dual grids and stitch cells ensures that the resolution of extracted isosurfaces automatically varies with the resolution of the data given in a region.

Several extensions to our method are possible. The use of a generic triangulation scheme would allow the use of our method for other, more general AMR data, where grids might not necessarily be axis-aligned, e.g., data sets produced by the AMR method of Berger and Oliger [3]. There are also possible improvements to our index-based scheme. The original AMR scheme by Berger and Colella [2] requires a layer of width of at least one grid cell between a refining grid and the boundary of a refined level. Even though Bryan [4] eliminates this requirement, we still require it. This is necessary, because this requirement ensures that only transitions between a coarse level and the next finer level occur in an AMR hierarchy. Allowing transitions between arbitrary levels would require us to consider too many cases during the stitching process. (The number of cases would be limited only by the number of levels in an AMR hierarchy, since within this hierarchy transitions between arbitrary levels are possible.) These requirements are equivalent to the requirements described by Gross et al.[7], where transitions between arbitrary levels are also prevented. Unfortunately, this requirement does not allow us to handle the full range of AMR data sets in use today, e.g., those produced by the methods of Bryan [4].

To handle the full range of AMR grid structures, our grid-stitching approach must be extended. Any LUT-based approach has inherent problems, since the number of possible level transitions is bounded only by the number of levels in a hierarchy. For the transformation of level coordinates to grid coordinates, we currently examine each grid in a level whether it contains a given grid point. This is efficient enough for moderately sized data sets; but for larger data sets, a space subdivision-



(i) Isosurface obtained when using two of seven levels of AMR hierarchy. Stitch cell generation required approximately $55\mathrm{ms}$, and isosurface generation required approximately $250\mathrm{ms}$



(ii) Isosurface obtained when using three of seven levels of AMR hierarchy. Stitch cell generation required approximately $340\mathrm{ms}$, and isosurface generation required approximately $600\mathrm{ms}$

Fig. 13. Isosurface extracted from AMR hierarchy simulating star clusters (data set courtesy of Greg Bryan, Massachusetts Institute of Technology, Theoretical Cosmology Group, Cambridge, Massachusetts)

based search scheme should be used. It should be possible to use a modification of the generalized k–D trees from Weber et al. [17] for this purpose.

9 Acknowledgments

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

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

References

1. AVS5. Product of Advanced Visual Systems, see `http://www.avs.com/products/AVS5/avs5.htm`.
2. Marsha Berger and Phillip Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82:64–84, May 1989. Lawrence Livermore National Laboratory, Technical Report No. UCRL-97196.
3. Marsha Berger and Joseph Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512, March 1984.
4. Greg L. Bryan. Fluids in the universe: Adaptive mesh refinement in cosmology. *Computing in Science and Engineering*, 1(2):46–53, March/April 1999.
5. L. Paul Chew. Constrained delaunay triangulations. *Algorithmica*, 4(1):97–108, 1989.
6. Allen Van Gelder and Jane Wilhelms. Topological considerations in isosurface generation. *ACM Transactions on Graphics*, 13(4):337–375, October 1994.
7. Markus H. Gross, Oliver G. Staadt, and Roger Gatti. Efficient triangular surface approximations using wavelets and quadtree data structures. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):130–143, June 1996.
8. Interactive Data Language (IDL). Product of Research Systems, Inc., see `http://www.rsinc.com/idl/index.cfm`.

9. William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21(4):163–169, July 1987.
10. Kwan-Liu Ma. Parallel rendering of 3D AMR data on the SGI/Cray T3E. In: *Proceedings of Frontiers '99 the Seventh Symposium on the Frontiers of Massively Parallel Computation*, pages 138–145, IEEE Computer Society Press, Los Alamitos, California, February 1999.
11. Nelson L. Max. Sorting for polyhedron compositing. In: Hans Hagen, Heinrich Müller, and Gregory M. Nielson, editors, *Focus on Scientific Visualization*, pages 259–268. Springer-Verlag, New York, New York, 1993.
12. Gregory M. Nielson and Bernd Hamann. The asymptotic decider: Removing the ambiguity in marching cubes. In: Gregory M. Nielson and Larry J. Rosenblum, editors, *IEEE Visualization '91*, pages 83–91, IEEE Computer Society Press, Los Alamitos, California, 1991.
13. Gregory M. Nielson, Dave Holiday, and Tom Roxborough. Cracking the cracking problem with coons patches. In: David Ebert, Markus Gross, and Bernd Hamann, editors, *IEEE Visualization '99*, pages 285–290,535, IEEE Computer Society Press, Los Alamitos, California, 1999.
14. Michael L. Norman, John M. Shalf, Stuart Levy, and Greg Daues. Diving deep: Data management and visualization strategies for adaptive mesh refinement simulations. *Computing in Science and Engineering*, 1(4):36–47, July/August 1999.
15. William J. Schroeder, Kenneth M. Martin, and William E. Lorensen. *The Visualization Toolkit*, second edition, 1998. Prentice-Hall, Upper Saddle River, New Jersey.
16. Raj Shekhar, Elias Fayyad, Roni Yagel, and J. Fredrick Cornhill. Octree-based decimation of marching cubes surface. In: Roni Yagel and Gregory M. Nielson, editors, *IEEE Visualization '96*, pages 335–342, 499, IEEE Computer Society Press, Los Alamitos, California, October 1998.
17. Gunther H. Weber, Hans Hagen, Bernd Hamann, Kenneth I. Joy, Terry J. Ligocki, Kwan-Liu Ma, and John M. Shalf. Visualization of adaptive mesh refinement data. In: Robert F. Erbacher, Philip C. Chen, Jonathan C. Roberts, Craig M. Wittenbrink, and Matti Groehn, editors, *Proceedings of the SPIE (Visual Data Exploration and Analysis VIII, San Jose, CA, USA, Jan 22–23)*, volume 4302, pages 121–132, SPIE – The International Society for Optical Engineering, Bellingham, WA, January 2001.
18. Gunther H. Weber, Oliver Kreylos, Terry J. Ligocki, John M. Shalf, Hans Hagen, Bernd Hamann, and Kenneth I. Joy. Extraction of crack-free isosurfaces from adaptive mesh refinement data. In: David Ebert, Jean M. Favre, and Ronny Peikert, editors, *Proceedings of the Joint EUROGRAPHICS and IEEE TCVG Symposium on Visualization, Ascona, Switzerland, May 28–31, 2001*, pages 25–34, 335, Springer Verlag, Wien, Austria, May 2001.
19. Rüdiger Westermann, Leif Kobbelt, and Thomas Ertl. Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. *The Visual Computer*, 15(2):100–111, 1999.