Multiresolutional Parallel Isosurface Extraction based on Tetrahedral Bisection

Thomas Gerstner and Martin Rumpf
Department for Applied Mathematics,
University of Bonn, Germany

Abstract

Nowadays, multiresolution visualization methods become an indispensable ingredient of real time interactive post processing. We will here present an efficient approach for tetrahedral grids recursively generated by bisection, which is based on a more general method for arbitrary nested grids. It especially applies to regular grids, the hexahedra of which are procedurally subdivided into tetrahedra. Besides different types of error indicators, we especially focus on improving the algorithm's performance and reducing the memory requirements. Furthermore, parallelization combined with an appropriate load balancing on multiprocessor workstations is discussed.

1 Introduction

A variety of multiresolution visualization methods has been designed to serve as tools for interactive visualization of large data sets. The local resolution of the generated visual objects, such as isosurfaces, is thereby steered by error indicators which measure the error due to a locally coarser approximation of the data. On one hand, post processing methods are based on already extracted surfaces and turn them into multiresolutional objects, which can then be interactively inspected [2, 5, 7, 14]. On the other hand, we can already adaptively extract the considered isosurfaces from the 3D data set. Thereby, starting at a coarse approximation of the data, we recursively add details in areas where some error indicator points out a large local error with respect to the exact data values. If the error is below a user prescribed threshold, the algorithm locally stops the successive refinement and extracts the surface on the current level. Different approaches have been presented to solve the outstanding continuity problem, i.e. to avoid cracks in the adaptive isosurfaces. In the Delaunay approach by Cignoni et al. [3] and in the nested mesh method by Grosso et al. [6] the successive remeshing during the refinement guarantees the continuity. On the other hand, Shekhar et al. [15] rule out hanging nodes by inserting additional points on faces with a transition from finer to coarser elements due to an adaptive stopping criterion.

We will here consider a method, which is based on tetrahedral grids generated by bisection. These meshes are widespread and well known from adaptive numerical methods [1]. We will especially focus on the recursive bisection [8] of originally hexahedral grids. Compared to an octree approach the tetrahedral strategy has several advantages:

- Adaptive octree strategies require some elaborated matching of local isosurfaces at transition faces [15] between cubes of different resolution. Even if a continuous adaptive projection [12] is generated, the bilinearity on faces requires special care [11]. In the tetrahedral frame such difficulties are ruled out.
- A mesh generated by tetrahedral bisection consists of typically three times more grid levels than a standard octree with the same data resolution. Therefore, the granularity of adaptive grids is more flexible.
- Furthermore, the multilevel algorithm attains a compact form. The operations to be performed in each refinement step are very simple.
- Tetrahedral bisection is not restricted to structured grid data. Although we here focus to this case, an essential advantage of tetrahedral grids is their potential in complex domain approximation. By pushing refinement nodes onto the actual curved boundary such grids may also be generated by recursive bisection, starting with a coarse initial grid.

Our approach presented here, can be seen as a special case and a very efficient implementation of a universal concept of multiresolutional visualization on arbitrary nested grids [10, 11, 12, 13]. The core of our approach is identical to the method presented by Zhou et al. [17]. It can be regarded as a 3D generalization of the techniques presented by Livnat et al. [9] and in [4]. Besides a discussion of different types of error measurement, we will focus on improving the algorithm's performance and its memory requirements. This invokes hash table techniques for smooth shading and the application of error indicators to evaluate data bounds on coarse grid tetrahedra which prevents us from storing min/max values. Furthermore, gauss map estimation enables us to implement a multilevel backface culling.

Finally, we will explain a load balancing concept on multiprocessor graphic workstations, which leads to near optimal speed up concerning the triangle generation rates. High resolution interactive visualization of very large data sets thus becomes a feasible option. To give an example, the presented algorithm is able to extract adaptive isosurfaces on large grids with about $800k$ triangles per second on a 4 processor SGI R10000 with infinite reality graphics.

2 Outline of the adaptive algorithm

Let us consider a family of nested, conforming, tetrahedral meshes $\{\mathcal{T}^l\}_{0\leq l\leq l_{\max}}$. The tetrahedra are assumed to be refined by recursive bisection. For a tetrahedron T the midpoint of a predestined edge $e_{\text{ref}}(T)$ is thereby picked up as a new node $x_{\text{ref}}(T)$, and the tetrahedron is cut at the face $F_{\text{ref}}(T)$ spanned by

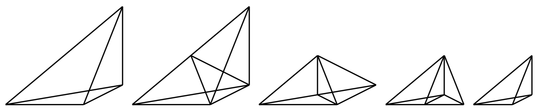

Figure 1: Recursive decomposition of a tetrahedron.

$x_{\mathrm{ref}}(T)$ and the two nodes of T opposite to $e_{\mathrm{ref}}(T)$ into two child tetrahedra $\mathcal{C}(T) = \{T_{\mathcal{C}}^1, T_{\mathcal{C}}^2\}$ (cf. Figure 1). In the considered applications we restrict ourselves to a specific case of such tetrahedral meshes, i.e. to originally regular hexahedral grids with n^3 nodes for $n = 2^k + 1$. These regular grids are only procedurally converted to a hierarchy of tetrahedral bisection grids, the nodes of which coincide with the regular grid nodes [8, 17]. A simple alternating scheme for the refinement edge e_{ref} guarantees the conformity of the resulting grids. Straightforward index arithmetic enables us to identify data values $U(x)$ at the nodes x of the tetrahedra in a recursive grid traversal. Let U^l denote the piecewise linear function on $\mathcal{T}^l$ uniquely described by the data values on the corresponding nodes.

Multiresolutional visualization methods can now be implemented on the hierarchy of tetrahedral grids. Here we restrict ourselves to isosurface extraction. Other visualization methods such as the generation of slices, or a projective volume rendering can be implemented analogously. In the latter case, the in general required back to front processing of tetrahedra is straightforward. The ordering of children in the recursive grid traversal solely depends on the viewing direction and the type of bisection. It can be determined in advance.

The adaptive isosurface algorithm is based on a depth first traversal of the grid hierarchy. On every tetrahedron we check for a stopping criterion. If it is true we stop and extract the local isosurface. Otherwise, we recursively proceed on the child set $\mathcal{C}(T)$. If we stop on a specific tetrahedron T and refine another tetrahedron $\tilde{T}$ which shares the refinement edge with T, i.e. $e_{\mathrm{ref}}(T) = e_{\mathrm{ref}}(\tilde{T})$, an inconsistency occurs at the hanging node x_{ref}. This leads to cracks in the isosurface. In the case of general nested grids we can apply adaptive projection operators to guarantee consistency. For details we refer to [12]. Here we simple have to ensure that, whenever a tetrahedron is refined, all tetrahedra sharing its refinement edge are refined as well. This can be achieved defining error indicators $\eta(x)$ on the grid nodes and choosing as a stopping criterion $\eta(x_{\mathrm{ref}}(T)) < \epsilon$ for some user prescribed threshold value ϵ. For an arbitrary error it still might happen that, although $\eta(x_{\mathrm{ref}}(T)) < \epsilon$, $\eta(x_{\mathrm{ref}}(\hat{T})) \geq \epsilon$ on some descendant $\hat{T}$, whose refinement point $x_{\mathrm{ref}}(\hat{T})$ is located on the boundary of T. Therefore, the adjacent tetrahedron will possibly be refined, and an inconsistency occurs again. To avoid this we assume the following saturation condition on the error indicators (cf. [12]):

(Saturation Condition)
$\eta(x_{\mathrm{ref}}(T)) > \eta(x_{\mathrm{ref}}(T_{\mathcal{C}}))$ *for all* $T \in \mathcal{T}^l$ *with* $l < l_{\max}$ *and* $T_{\mathcal{C}} \in \mathcal{C}(T)$.

An error indicator η is called admissible if it fulfills the saturation condition. Otherwise it can easily be adjusted in a preroll step. By using a bottom up traversal of the hierarchy we construct the saturated indicator as the smallest indicator larger than the original indicator respecting the saturation condition. Let us emphasize that a depth first traversal of the hierarchy in the adjustment procedure would not be sufficient.

Now we are able to formulate the adaptive algorithm. The depth first traversal of the grid hierarchy can be sketched in pseudo code as follows:

Inspect(T) {
    if TetrahedronIsOfInterest(T)
        if $\mathcal{C}(T) \neq \emptyset \wedge \eta(x_{\text{ref}}) \geq \varepsilon$
            { Inspect($T^1_{\mathcal{C}}$); Inspect($T^2_{\mathcal{C}}$); }
        else Extract(T);
}

The function *TetrahedronIsOfInterest()* checks whether the tetrahedron is a candidate for the intersection with an isosurface or not. In our case, it is checked if the current isovalue is contained in a certain data interval. For an implementation of such a routine considering the already available error indicator values see Sect. 3.

3 Error Measurement

The visual impression and a sufficient resolution of the numerical data in the visualization process is closely related to the specific type of error measurement applied in the adaptive traversal of the tree structure. In our case, error indicators will be defined on grid nodes. All nodes, except those on the coarsest level, are refinement nodes $x_{\text{ref}}(T)$ on a refinement edge $e_{\text{ref}}(T)$ with respect to a tetrahedron T. Therefore, an indicator value $\eta(x)$ measures the error on all the tetrahedra, which share the corresponding edge. In what follows we will present different types of error indicators and explain some of their benefits.

Instead of considering the true data values $U(x)$ at the grid nodes we can consider the offset values $U_\delta(x)$ corresponding to the approximation on the next coarser level. They are related to the original data values by the recursive formula

$$U(x_{\text{ref}}(T)) = \frac{U(x_1) + U(x_2)}{2} + U_\delta(x_{\text{ref}}(T))$$

where x_1 and x_2 are the end points of the edge $e_{\text{ref}}(T)$. For smooth data, e.g. $U(x) = u(x)$ for all nodes x with $u \in C^2$, $|U_\delta(x_{\text{ref}}(T))| = O(\text{diam}(T)^2)$ which implies the saturation condition holds asymptotically on grids $\mathcal{T}^l$ for l sufficiently large. Let us emphasize that the handling of the U_δ–values would therefore allow an economical δ–compression of the data and the original values can easily be retrieved during the recursive tree traversal. Now, we define the hierarchical error indicator $\eta_H(x) := |U_\delta(x)|$. As before it is admissible if the saturation condition is fulfilled. The resulting isosurfaces are shown in Figure 2 and 4. Polygon and frame rates are listed in Table 5. Instead of isosurfaces we can analogously extract arbitrary slices and visualize data adaptively on these slices (cf. Figure 3).

As an alternative to the above saturation procedure we can compute a robust upper bound for the offset values on elements by the recursive formula

$$\eta_H^+(x) := \eta_H(x) + \max_{T_\mathcal{C} \in \mathcal{C}(T)} \eta_H^+(x_{\mathrm{ref}}(T_\mathcal{C})) \tag{1}$$

where for nodes appearing on the second finest grid level $\eta_H^+(x) := \eta_H(x)$. These values can also be used to perform the necessary intersection test during the hierarchical extraction of an isosurface. We thereby avoid the expensive storing of min/max–values as discussed in [16].

With a focus on an isosurface's geometric shape, we will now consider a curvature estimation. We ask for a discrete curvature quantity which locally measures the quality of the data approximation from the viewpoint of the visual appearance [11, 12]. In isosurface images consisting of linear patches we can easily recognize folds on the surface. In each tetrahedron the data gradient ∇U^l is always perpendicular to an isosurface. Therefore, at any face F the normal component of the jump of the normalized gradient, denoted by $[\frac{\nabla U^l}{|\nabla U^l|}]_F$, locally measures the fold in the data function (cf. Figure 5). Here, the jump operator $[\cdot]_F$ is defined as the difference of the argument on both sides of the face. This jump obviously serves as a well–founded graphical error criterion and motivates the following definition of an error indicator for a refinement node $x_{\mathrm{ref}}(T)$ with $T \in \mathcal{T}^l$:

$$\eta_N(x_{\mathrm{ref}}(T)) \quad := \quad \left[\frac{\nabla U^l}{|\nabla U^l|}\right]_{F_{\mathrm{ref}}(T)}$$

After a possible saturation it serves as an admissible indicator related to the visual appearance. Alternatively, we can saturate this error indicator recursively by adding finer level indicator values as in the hierarchical case (cf. Eq. 1). Thereby, we obtain a new indicator η_N^+. Besides the representation of a visual error this indicator allows an adaptive backface culling. Let N denote the normal of some triangle of the final isosurface triangulation on the tetrahedron $T \in \mathcal{T}^l$ and V the viewing vector from the object to the eye (we confine ourselves here to parallel projection). Then, the triangle is faced towards the viewer, if $N \cdot V \geq 0$, it will not be drawn otherwise. Now, we obtain a significant acceleration of our isosurface algorithm, if on a much coarser grid level we recognize tetrahedra containing only isosurface triangles which are faced away from the viewer. Then we are already able to stop the local traversal on this level. Thereby,

$$N \cdot V + \eta_N^+(x_{\mathrm{ref}}(T)) \leq 0$$

serves as an appropriate, additional stopping criterion. It can easily be seen that, on average, while arbitrarily rotating the object, we save up to one half of the computing time for an isosurface (cf. Table 1).

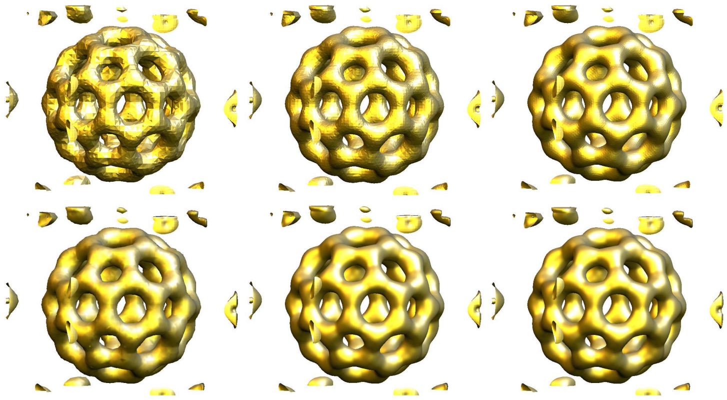

Figure 2: Flat (top row) and smooth (bottom row) shaded adaptive isosurfaces are extracted from a 129^3 sized Bucky Ball data set. We consider the hierarchical error indicator for threshold values $\epsilon = 0.02, 0.005, 0.0$, where 80 k, 211 k, respectively 590 k triangles are generated.

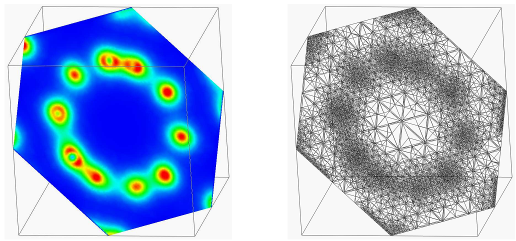

Figure 3: A color shading on slices (left) and the drawing of intersection lines on faces of the corresponding adaptively extracted tetrahedra (right) is shown for the Bucky Ball data set. For a threshold value of $\epsilon = 0.005$ about 16 k triangles are shaded.

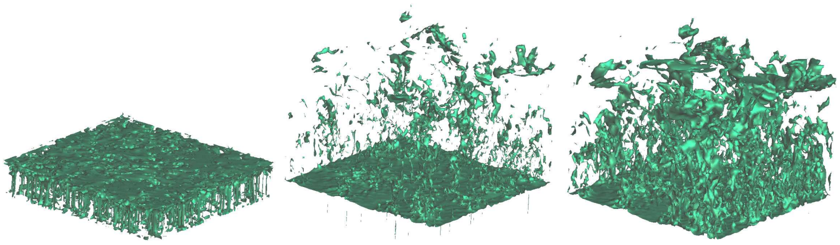

Figure 4: Three isosurfaces are extracted from a $220 \times 220 \times 100$ (resampled to 129^3) regular CFD data set, the velocity of a turbulent flow field above a white dwarf star (courtesy A.Kercek, MPA Garching) for $\varepsilon = 0.01$ and isovalues 0.5, 1.5 and 2.5. The horizontal structure corresponds to the surface of the star.

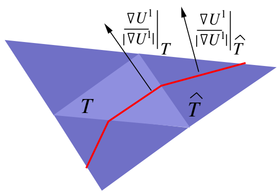

Figure 5: The jump of the normalized gradient is a suitable error criterion. Here the analogous 2D case is depicted.

Unfortunately, the evaluation of normals on every tetrahedron traversed in the adaptive algorithm is computationally expensive. Therefore we ask for a modification, which still ensures an adaptive backface culling. We consider an error indicator which measures non–normalized gradient jumps, i.e.

$$\eta_{1,\infty}(x_{\text{ref}}(T)) \quad := \quad [\nabla U^l]_{F_{\text{ref}}(T)} \,,$$

and analogously construct a saturated indicator $\eta^+_{1,\infty}$. Then the backface rejection criterion can be modified to

$$\nabla U^l \cdot V + \eta_{1,\infty}(x_{\text{ref}}(T)) \leq 0 \,.$$

The gradients $\nabla\hat{\lambda}_i$ of the barycentric coordinates $\{\hat{\lambda}_i\}_{0\leq i\leq 3}$ with respect to a reference tetrahedron $\hat{T}$ for each class of up to translation and scaling identical tetrahedra can be precomputed. If $\alpha(T)$ is the corresponding scaling factor for a specific tetrahedron T the backface test reduces to

$$\sum_{i=1}^{3} U_i W_i \leq -\alpha(T)\,\eta_{1,\infty}(x_{\text{ref}}(T))$$

where $W_i = V \cdot \nabla\hat{\lambda}_i$ is also precomputable. Table 1 shows the effect of this strategy in reducing the number of visited cells. Asymptotically, we obtain a saving of nearly $\frac{1}{2}$ for fine grid resolutions.

For details on further types of error indicators we refer to [12]. Here we have focused on above two types in order to emphasize that error indicators serve other purposes as well, such as the evaluation of data bounds or adaptive backface culling.

4 Reducing the storage requirements

Independently of the concrete type, error indicators have to be precomputed and stored on the grid nodes. A naive approach requires the same memory as the numerical data set itself. Let us now discuss an improvement which significantly reduces storage requirements. First, we recognize that slightly increasing the indicator values does not affect the overall performance of the algorithm. Typically, we do not need the true values but the ordering of nodes

	without backface culling		with backface culling	
ε	triangles drawn	visited tetrahedra	triangles drawn	visited tetrahedra
1.6	72076	181616	56918	169270
0.8	92955	227874	65485	205010
0.4	126463	301332	81081	252936
0.2	224522	521472	130262	380627
0.1	342173	781218	188072	519739
0.05	463443	1026587	256757	650741
0.0	590018	1259669	317024	772752

Table 1: Number of generated triangles and visited tetrahedra with and without multilevel backface culling for different threshold values ε using the normalized gradient jump error indicator η_N^+.

corresponding to their indicator value. We are therefore able to classify the indicator values according to the intervals

$$(\alpha^{m+1}, \alpha^m]$$

in which they are contained for a fixed $\alpha \in (0,1)$. Then, we only have to store m which is a small integer and only needs some bits in storage instead of several bytes for a floating point number. A lookup table enables us to retrieve the α^m values efficiently.

The appearance of the extracted isosurfaces is significantly improved when smooth shading is used. Therefore, normals on vertices of the isosurface triangulation are required. For data sets of moderate size normals can be calculated on the grid nodes of the original mesh in advance and be stored. But on finer grids, it is undesirable to provide three times the storage of the actual data set to store normals derived from the data. Thus, they have to be calculated when needed at runtime. The normals are required at vertices on edges of the original grid. They can be obtained by linear interpolation of discrete gradients on the edge end points. Discrete gradients at these nodes are typically calculated by central differences in case of regular hexahedral grids, or by local averaging of cell normals. Such a normal is requested several times, once for each tetrahedron sharing the edge. We enable a retrieval of already calculated normals using a hash table for them. On a n^3 sized data set the number of vertices on the isosurface triangulation, which equals the number of intersected edges is at most $O(n^2)$ depending on the smoothness module of the discrete function U. A hash table of size $O(n^2)$ is therefore sufficient for our purpose. The pair of index vectors for the end points of the intersected edge serves as an appropriate hash key.

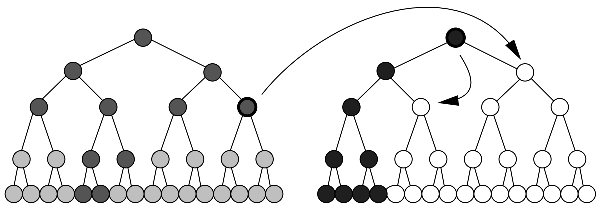

Figure 6: Two process subtrees are sketched. The dark grey process has already finished his job passing over all its tetrahedra of interest. The light grey nodes indicate tetrahedra which are not reached due to the adaptive stopping and the hierarchical testing. Now the dark grey process resets its mark (indicated by a thickened circle), taking over part of the originally black process subtree. The black mark is correspondingly shifted downwards.

5 Parallel Implementation

In what follows we will address the question of how to run the presented algorithm efficiently in parallel on a workstation with several processors. The very first idea would be to apply a domain decomposition method with respect to the tetrahedral grid. But this approach is not able to manage the complicated behaviour of our adaptive and hierarchical isosurface extraction algorithm. Small and large portions of the isosurface will both be located in regions of the same size. Therefore, the number of traversed tetrahedra by the algorithm in each region – which is proportional to the required computing time – will significantly differ. The problem is therefore how to prevent processors from getting idle.

Here, we present an efficient solution to this problem, independent of the concrete isosurface shape and straightforward to implement. Interactive visualization in mind, we confine ourselves here to the case of a shared memory workstation where processes (threads) are assumed to have comparable access times to arbitrary local information in the complete data set. The exchange of domain data turns out to be critical in distributed environments, although unavoidable in certain cases of very fine grids. Then, load balancing decisions have to be taken more carefully.

Initially, we set up several processes, each of which is given a subtree of the complete tetrahedral hierarchy which can be handled independently of the other processes subtrees. We denote the entry node of the current subtree of a process by its mark. In particular, we assume that the right subtree of a process has not yet been processed. If the process is going to enter his right subtree, the corresponding mark has to be shifted downwards into the right subtree.

Now, if a process has already finished its whole subtree, a new task has to be found. Therefore, the entry node of a right subtree of another process subtree is chosen as the currently idle process's new mark. We always select the process whose mark is on the coarsest level of the grid. The mark of the process

ε	triangles drawn	visited tetrahedra	f/sec (1 proc)	f/sec (4 proc)
0.02	81184	201757	3.45	8.33
0.01	128709	307384	2.27	5.26
0.005	211219	487107	1.43	3.44
0.0025	315440	727419	0.98	2.43
0.00125	439230	984029	0.74	1.78
0.0	590018	1259669	0.58	1.36

Table 2: Number of generated triangles, visited tetrahedra, frames per second in the scalar, and in the parallel case for different threshold values ε using the hierarchical error estimator η_H^+.

dealing with the complete subtree before, is shifted recursively downwards to the right, starting at the left child node of the former mark until it reaches a node in the right subtree which is yet unvisited (cf. Figure 6). This strategy ensures that a process which becomes idle is supplied with the largest unvisited subtree.

Let us finally remark on the actual implementation in case of 6 tetrahedra on the coarsest level of the grid hierarchy, e.g. for a cube subdivided into tetrahedra. We index them with integers $6, \ldots, 11$. The other tetrahedra are numbered recursively. Left and right child are indexed $2i$, respectively $2i+1$ for a tetrahedron with index i. We solely have to store an array of such indices, one for each process. If a process has finished a left subtree, his mark index is set to the right subtree index until the complete tree has been processed. An idle process looks for the smallest mark index $i_{\min}$ in a list of indices for all processes, sets its mark index to $2i_{\min}+1$, and modifies the other index.

If the graphic workstation is equipped with m processors, we may start m processes. One of them has to be reserved to push graphic patches from some buffer into the graphics hardware. The other $m-1$ processes can be occupied with actual isosurface extraction. Thereby, they collect patches to be drawn in buffers which they hand over to the first process. On a 4 processor SGI R10000 workstation with infinite reality graphics a speedup of 2.4 and a rate of $800k$ triangles can thus be achieved. Detailed results are listed in the Table 2. There we considered the hierarchical error estimator.

References

[1] E. Bänsch. Local mesh refinement in 2 and 3 dimensions. *IMPACT of Computing in Science and Engineering*, 3:181–191, 1991.

[2] A. Certain, J. Popović, T. DeRose, T. Duchamp, D. Salesin, and W. Stuetzle. Interactive multiresolution surface viewing. In *SIGGRAPH 96 Conference Proceedings*, pages 91–98, 1996.

[3] P. Cignoni, L. De Floriani, C. Montoni, E. Puppo, and R. Scopigno. Multiresolution modeling and visualization of volume data based on simplicial complexes. In *Symposium on Volume Visualization*, pages 19–26, 1994.

[4] T. Gerstner. Adaptive hierarchical methods for landscape representation and analysis. In S. Hergarten and H.-J. Neugebauer, editors, *Lecture Notes in Earth Sciences 78*. Springer, 1998.

[5] M. H. Gross and R. G. Staadt. Fast multiresolution surface meshing. In *Proceedings Visualization*, pages 135–142, 1995.

[6] R. Grosso, C. Luerig, and T. Ertl. The multilevel finite element method for adaptive mesh optimization and visualization of volume data. In *Proceedings Visualization*, 1997.

[7] H. Hoppe. Progressive meshes. In *SIGGRAPH 96 Conference Proc.*, pages 99–108, 1996.

[8] J. Maubach. Local bisection refinement for n-simplicial grids generated by reflection. *SIAM J. Sci. Comput.*, 16:210–227, 1995.

[9] Y. Livnat, H. W. Shen, and C. R. Johnson. A near optimal isosurface extraction algorithm using the span space. *Transaction on Visualization and Computer Graphics*, 2(1):73–83, 1996.

[10] R. Neubauer, M. Ohlberger, M. Rumpf, and R. Schwörer. Efficient visualization of large scale data on hierarchical meshes. In W. Lefer and M. Grave, editors, *Visualization in Scientific Computing*. Springer, 1997.

[11] M. Ohlberger and M. Rumpf. Hierarchical and Adaptive Visualization on Nested Grids. *Computing*, 59 (4):269–285, 1997.

[12] M. Ohlberger and M. Rumpf. Adaptive projection methods in multiresolutional scientific visualization. *IEEE Transactions on Visualization and Computer Graphics, Vol 4 (4)*, 1998.

[13] M. Rumpf. Recent numerical methods – a challenge for visualization. *Future General Computer Systems*, to appear, 1998.

[14] W. J. Schroeder, J. A. Zarge, and W. A. Lorensen. Decimation of triangle meshes. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 65–70, 1992.

[15] R. Shekhar, E. Fayyad, R. Yagel, and J. F. Cornhill. Octree-based decimation of marching cubes surfaces. In *Proc. Visualization*. IEEE, 1996.

[16] J. P. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. In *Computer Graphics*, volume 24, 5, pages 57–62, 1990.

[17] Y. Zhou, B. Chen, and A. Kaufman. Multiresolution tetrahedral framework for visualizing volume data. In *IEEE Visualization '97 Proceedings*. IEEE Press, 1997.