PSP: Progressive Subdivision Paradigm for Large Scale Visualization

R. Borgo,[1] R. Scopigno,[1] P. Cignoni,[1] and V. Pascucci,[2]

[1] Visual Computing Group, Consiglio Nazionale delle Ricerche
[2] Center for Applied Scientific Computing Lawrence Livermore National Laboratory

Abstract

The increasing rate of growth in size of currently available datasets is a well known issue. The possibility of developing fast and easy to implement frameworks able to visualize at least part of a tera-sized volume is a challenging task. Subdivision methods in recent years have been one of the most successful techniques applied to the multiresolution representation and visualization of surface meshes. Extensions of these techniques to the volumetric case presents positive effects and major challenges mainly concerning the generalization of the combinatorial structure of the refinement procedure and the analysis of the smoothness of the limit mesh. In this paper we address mainly the first part of the problem, presenting a framework that exploits a subdivision scheme suitable for extension to 3D and higher dimensional meshes.

1. INTRODUCTION

Modern scanning devices, modelling systems and computer simulations give rise to surface and volume of ever increasing resolution. Real-time display and transmission of this sheer amount of data is a challenging task requiring to generate approximations of minimal size with respect to given error bounds. To address these issues new data-streaming techniques have been proposed mainly concerning progressive processing and visualization. Techniques relying on subdivision paradigms to generate approximations of minimal sizes, with respect to given error bounds, have made great progresses especially in the surface mesh visualization field. Generalization of such techniques to the volumetric case is not always straightforward. In this paper we present a new approach that combines the flexibility of a progressive multiresolution representation with the advantage of a recursive subdivision scheme. Our approach right now focuses on the case where the multi-resolution representation of the volumetric data is based on the edge bisection refinement rule widely used in mesh generation [10, 11].

2. PREVIOUS WORK

In the course of the paper we will refer mainly at isocontour extraction and visualization in very large dataset. For reason of space we need to abbreviate the section related to the state of the art of isosurface extraction techniques leaving the reader to refer to the Bibliography for a more detailed analysis. A very rich literature in isosurface extraction exists. Three main classes of algorithms can be identified. The first one groups all those methods that overworked and improved the well known Marching Cubes algorithm [6]; examples of accelerating techniques included the use of hierarchical data structures like octrees [14] and value decomposition methods [12, 5]. A second class of isosurface rendering algorithms refers to techniques that resembles the contour propagation algorithms [7] of Pascucci et al.; these type of algorithms identify a seed cell from which to begin the propagation, they end up with a sort of seed set covering the isovalue range. The third class groups those algorithms that mainly focus on the reduction of the number of triangles generated during the isosurface extraction; belongs to this class the algorithm proposed by Livnat and Hansen [4]. The approach we focus on belongs to the class of Subdivision Schemes introduced by Pascucci [9]. The approach adopted responds well to three big issues typical of multiresolution approaches: vertex proliferation (mainly dependent on the subdivision mask adopted), efficient extraction of the refined surface, rendering in time-critical environment. The following section describes in details the mathematical rules at the base of our refinement algorithm.

3. Subdivision Scheme Description

The refinement scheme at the base of our framework follows the edge-bisection refinement introduced by Rivara in [11] and Velho and Zorin in [13](4-8 or $\sqrt{2}$ subdivision). Figures 1 (a-e) show the subdivision scheme for a rectilinear grid, Figures 1(a'-e') show the subdivision strategy applied to quadrilateral elements. The base mesh is a squared mesh divided into triangles by bisecting each square at the middle of one of the two diagonals; the diagonal selected for the bisection is considered as "main diagonal". Each generated triangle is subdivided, at each refinement step, at the middle of its longest edge. Each refinement is performed inserting a point at the center of each square/rhombus and splitting the diamond into four triangles. Each pair of triangles adjacent along an old edge are merged into a new square/rhombus. In the next section we show how this procedures can be generalized to the volumetric case. For the extension of the scheme to 3D we organize the subdivision process into levels and tiers (see [9]). Each level l has four tiers, from 0 to 3, where tier 3 of level l is coincident with tier 0 of level $l+1$. This naming convention is used to maintain the comparison with classical tensor product subdivisions that would refine directly a mesh from tier 0 of level l to tier 0 of level $l+1$. In our scheme each refinement is a transition from tier i to $i+1$. At tier 3 the level is increased by one and the tier is reset to 0. We denote cells, facets, edges and vertices of the generated grid with the symbols c_i, f_i, v_i.

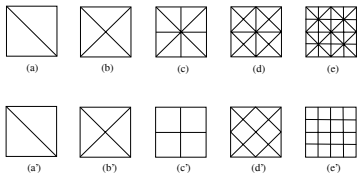

Figure 1: *4-8 recursive subdivision. (a-e) Classical longest edge bisection of a rectilinear grid. (a'-e') Equivalent $\sqrt{2}$ subdivision where pairs of adjacent triangles are merged into one square.*

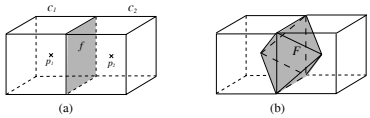

Figure 2: *3D cell refinement from tier 0 tier 1. (a)The two cells c_1 and c_2 in tier 0 . Their centers p_1 and p_2 are marked with two crosses. Their adjacency facet f is highlighted in gray. (b) The cell F of tier 1 (in gray) is the union of the pyramids $p_1 \lhd f$ and $p_2 \lhd f$.*

3.1. Subdivision Rules

In this section we analyze the geometrical aspect of our subdivision scheme. The subdivision scheme is similar to the 2D case described earlier in the paper. Extension of the scheme to the 3D case augments the subdivision process of one step indicated as tier 3. The following paragraphs analyze each refinement step in details.

3.1.0.1. From tier 0 to tier 1. For each cell c_i in the input mesh its center p_i is selected. The cell c_i having n facets is decomposed into n pyramidal cells by connecting the center p_i with all its facets. Let's denote by $p \lhd f$ the pyramid built by connecting p with a facet f. For each pair of cells c_i, c_j, adjacent along a facet f, a new cell F is created by merging the pyramid $p_i \lhd f$ with the pyramid $p_j \lhd f$:

$$F = (p_i \lhd f) \cup (p_j \lhd f), \qquad \text{with } f = c_i \cap c_j.$$

Figure 2 shows the construction of F from c_1 and c_2.

3.1.0.2. From tier 1 to tier 2. Consider a cell F of tier 1 and its center q. Let g_i be the facets of F that do not belong to tier 0 (for non-sharp F all the facets are of tier 1). We decompose F into a set of pyramids each given by $q \lhd g_i$. If F is a sharp cell, its center q_k is coincident with the center of its facet f of tier 0. In this way we handle directly boundary cases and 2-dimensional sharp features. Each pyramid $q \lhd g_i$ contains exactly one edge e_j of tier 0. After each tier 1 cell is split all the pyramids incident to the same edge e are merged into a cell E. All the cells built in this way form the mesh of tier 2. Figure 3 shows the construction of one cell of tier 2. The coarse mesh has four cells all incident to an edge e (Figure 3a). Four cells of tier 1 are built by merging pairs face pyramids (Figure 3b). Each tier 1 cell is then decomposed into four pyramids, of which we select only two incident to e (Figure 3c). The eight pyramids selected (two per cell) are finally merged into one cell E of tier 2, (Figure 3d).

3.1.0.3. From tier 2 to tier 3. As in the previous two steps one determines the center r of any cell E. Each cell E is then partitioned by joining r with each facet of E. As usual, for sharp cells the point r should be considered as the center of e and is shared among all the cells around e. The last merging step is among cells that are incident both to a vertex v and a cell center p. Figure 4 shows the construction of one cell of tier 3 from a cell of tier 2.

3.2. Refinement Characterization

Given a mesh representation model it can be organized hierarchically in terms of embedded entities that we call diamonds (with the term *diamond* we indicate a cell that can be combinatorially partitioned into a set of simplices all sharing an edge, called *axis* of the diamond, all the cells generated by our scheme are diamonds, for extended proof of the assumption made see[9]). By construction, the topology of such hierarchy is implicit to the diamonds themselves: from its center, characterized by three index (i, j, k), it is possible

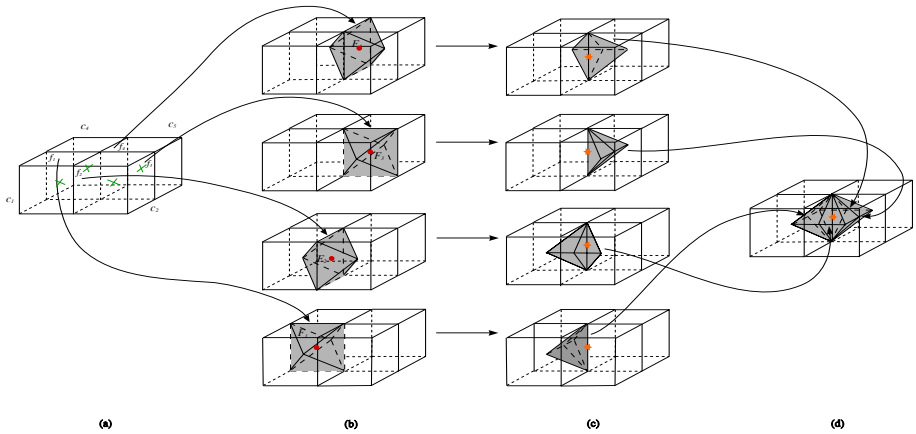

Figure 3: *Cell refinement from tier 1 to tier 2. (a) Four cells c_1, c_2, c_3 and c_4 of tier 0 share, in pairs, the facets f_1, f_2, f_3 and f_4. The edge e is shared by all facets f_1, f_2, f_3 and f_4. (b) Each facet f_i generates a cell F_i. (c) Each cell F_i is decomposed into four pyramids only two of which are selected. The selected pyramids are those containing the edge e. (c) All the pyramids containing e are merged together to form the cell E of tier 2.*

to derive tier, type, orientation and refinement level it represents, through simple mathematical rules it is possible to identify its sons. Every point of the mesh can be reached following our subdivision scheme. Traversals of the mesh by means of our diamond hierarchy allows the extraction of all the mesh related information: mesh data, range and approximation error. Diamonds as entities do not really exists, only their centers exists. The regularity of the diamond shape allows in fact to gather the diamond vertexes simply adding a δ constant to the center coordinates. Overworking these properties we have developed a Progressive Subdivision Paradigm (PSP) oriented to the visualization of large dataset. The following section describes the implementation details of our PSP algorithm and data-structures.

4. PSP Framework

The Progressive Subdivision Paradigm (PSP) framework corresponds to a level-of-detail approximation of a regular data volume. Each level consists of a set of uniformly represented diamond-entities generated through recursive subdivision of the volume and fusion of adjacent items following a merging "diamond-generation" scheme. Any kind of traversal of the multiresolution framework generates an approximation of the object volume corresponding to an error-based simplification of the volume itself. Our multiresolution framework can be seen as a two phases process: a *pre-processing* where auxiliary information (data, range, approximation error) are extracted, and a *rendering phase* where the mesh is traversed, at run-time, to extract the model under appropriate constraints (view-dependent, adaptiveness, error-based criteria). Input of the framework is a regular volumetric dataset extended when needed to even dimension i.e. $(2^N+1)\times(2^N+1)\times(2^N+1)$.

4.1. Pre-Processing Phase

Following the subdivision scheme the pre-processing phase correspond to a formalization of the dataset with our subdivision scheme. Initially the volume is subdivided through a per-vertex adding process. The initial step consists of the subdivision of the bounding volume introducing the vertex corresponding to the center of the bounding box itself, each successive step picks up new vertexes from the original volume and adds them continuing the subdivision process until all vertices are added. Vertexes are added at each step following a breadth first priority (BFP) policy. Through the subdivision process we extract the data embedded in the volume and calculate the *range* (min and max field values contained in a diamond) belonging to each diamond. In a successive step we traverse the volume in depth first order to compute the *approximation error* belonging to each diamond. Results are described in the following paragraph.

4.1.1. Data Organization

In the implementation of our framework we have decided to organize all of the information inferable from the mesh representation model in tables. We end up with three main tables: data, range, field. Each table has dimension $(2^N+1)\times(2^N+1)\times(2^N+1)$, equal to the dimension of the volume, and access key equal to a function of the (i,j,k) indexes of each diamond center. Filling of data and range tables can be done during the volume subdivision, a simple min/max routine assures the nesting of the min/max ranges. Because volume subdivision is performed following a BFP policy, the complexity of the filling step is equal to the complexity of a breadth first visit of a tree, linear in the number of cells/nodes. For computing the approximation error an explicit representation of the hierarchy is needed. The er-

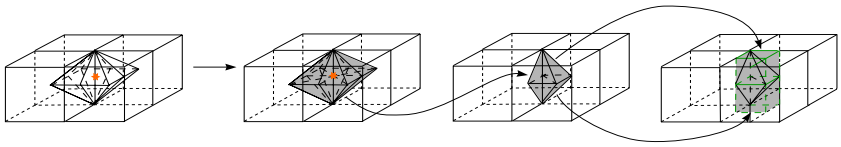

Figure 4: *Cell refinement from tier 2 to tier 3.*

ror metric we adopt assures an overestimation of the error introduced.

4.2. Rendering Phase

Extraction and refinement of the isosurface are executed at runtime performing traversals of the mesh representation model. Starting from the diamond cell of center coincident with the bounding box of the entire model we proceed generating, following our subdivision rules, the sons needed. The mesh can be traversed following a breadth first policy, to obtain a rough but homogeneous approximation of the original dataset, or a depth first policy allowing for selective refinement of the original dataset. Only those cells intersecting the isosurface are visited and eventually refined. Isosurface extraction is performed during the traversal. Refinement of a diamond is decided in function of error metrics (see Section 4.2.2). The isosurface is extracted even if further refinement is needed, this allows us to keep always a consistent version of the model available and to render at any given time partial results while the computation makes progress.

4.2.1. Isosurface Extraction

To perform the extraction we subdivide each diamond cell, belonging to the level of refinement required, into tetrahedra. In this way we have a piecewise linear representation of the scalar field $\mathcal{F}(x)$ necessary to compute an isocontour using the marching tetrahedra algorithm. Each isocontour is updated within a single tetrahedron and then composed to update the global isosurface within the set $\mathcal{T}$ of all tetrahedra around the bisection edge.

4.2.1.1. Isosurface Extraction: Inheritance. The recursive subdivision produces, by construction, a set of "partially" embedded diamonds, partially because only a portion of a diamond is embedded in each of its fathers and a diamond embeds only a portion of each of its children. This special embedding allows for each diamond to share with its fathers and sons part of the isocontour it intersects. To exploit this property and to avoid redundant calculation we have decided to try to support the inheritance of shared vertexes between diamonds: fathers pass to sons vertexes in common. Because there is no explicit representation of the hierarchy produced by the subdivision process (as mentioned in Sect 3.2 the topology of the refinement hierarchy is implicit to the cells) to support vertex inheritance we need to explicit

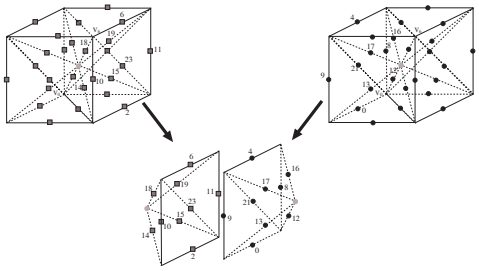

Figure 5: *Isovertex inheritance for a tier1 diamond from its two tier0 fathers. The diamond inherits exactly 12 vertexes, 8 from each father but 4 in common on the shared face.*

the hierarchy for at least two levels of refinement: the one of just refined diamond (i.e. diamonds belonging to refinement level l), and the one of diamonds generated by the refinement (i.e. diamonds belonging to refinement level $l+1$). The two hierarchy levels are organized in a data-structure, of only two levels, called Hierarchy Tree (HT). Each node in HT stores diamond center, computed isovertexes and sons position. The HT structure is used only at runtime. Supporting inheritance has gains and loss, advantages and disadvantages of this choice are analyzed in sections 4.2.3 and 5.

4.2.2. Error Metrics

To measure the error introduced by approximating the rendered model with low resolution level of details we adopt two different error metrics: field space error [1] (δ) and screen space error (ρ). Our field space error measure is an overestimation of the field space error computed between successive levels of refinement. The field space error is computed traversing the hierarchy DT from bottom to top in the preprocessing phase. The error of a diamond is the maximum between its internal error and the error of its sons, this guarantees a correct propagation of the object space errors during pre-processing. View-dependent algorithms projects object space errors onto the screen generating a screen space error $\rho(\delta)$. Screen space error is simply a factor that amplifies the object space error. It can be computed in function of the distance along the view direction of the objects from the point of view. The most simple metric of this form can be written as:

$$\rho_i = \lambda \frac{\delta_i}{||\mathbf{p}_i - \mathbf{e}||} \tag{1}$$

Dataset	Size	Pre-processing Time (sec.)
Hipip	$64 \times 64 \times 64$	3.9 secs
Hydrogen	$128 \times 128 \times 128$	10.5 secs

Table 1: *Computational time required for the pre-processing phase of the algorithm. Performances computed over the HIPIP* (64^3) *and the IDROGEN-ATOM datasets* (128^3).

4.2.3. Memory Occupancy and Overheads

In our strategy we have decided to store all the mesh related information in tables. We have four main table: data (IT), field value range(MT), error (ET) and flag (FT). Each of them has size equal to the size of the mesh grid: $(2^N + 1) \times (2^N + 1) \times (2^N + 1)$. Tables requires storage and computational time for filling. Let's us analyze both aspects in detail:

4.2.3.1. Storage Field values and errors are data that needs to be stored besides any type of implementation. Range is an information needed to be able to perform efficient isocontouring and, especially when dealing with of large meshes, the possibility to discard cells, not intersecting the isocontour, means lots of computational time saved during mesh traversal. The regularity of the structures in which those data are stored and the methods used for accessing the data makes them suitable for partitioning and distribution on the type of resources available.

4.2.3.2. Computational Time Table filling is one of the heaviest operations we perform, for this reason it is restricted to the phase of pre-processing. The only table "filled" (some of its value are updated during the hierarchy traversal) at run time is FT. Part of the memory is occupied by the Diamond Tree that we need to create for computing the error approximation (but just in the pre-processing phase). This structure is used only during the pre-processing phase. The diamond tree (DT) is actually needed because of the error metric adopted (Sect 4.2.2) that requires to easily move from bottom levels to top levels of the hierarchy. In the present context we can guarantee, keeping the hierarchy representation implicit to each diamond, an easy top-down traversal of the hierarchy but not an equally easy traversal bottom-up. In DT we store the least possible amount of information (only center coordinates, 3 short for each center, and pointer to the sons, 8 short in the worst case for each diamond), for hierarchy construction, due to the regularity of the subdivision, DT can be easily partitioned in blocks of smaller size. Each block can be distributed to different processors each of which can perform independently the error calculation. The regularity of the subdivision mask applied and the organization of the information in Tables allow us at run-time to keep everything implicit in the diamond cells that require a very low footprint to be represented (3 short). To access data in the tables we need only the center coordinates of the diamond we are interested in (constant time). The introduction of vertex-inheritance support causes as immediate drawback an overhead in memory requirements for what concerns the two level hierarchy structure HT. Nevertheless the introduced overhead is worth compared to the gain in terms of computational time saved. From the point of view of computational time saved introducing this overhead we avoid to recalculate for each diamond all the isosurface vertexes it contains limiting the computation to those vertexes not in common/inherited from the fathers reducing the operation of interpolation of a factor of 3. Results are shown in Section 5.

Dataset	Resolution	IsoSurf. Extr. w/ Inh.	IsoSurf. Extr. w/o Inh.
Hipip (64^3)	60%	0.1 secs	0.3 secs.
	85%	0.4 secs	1.0 secs.
	100%	0.9 secs	2.5 secs.
Hydrogen (128^3)	60%	0.4 secs	1.0 secs.
	85%	1.5 secs	4.6 secs.
	100%	3.9 secs	11.7 secs.

Table 2: *Computational time required for the run-time phase of the algorithm. Performances computed over the HIPIP dataset* (64^3) *and the IDROGEN-ATOM dataset* (128^3)

5. Results

Our algorithm has been implemented in C^{++} and developed on both SGI and Windows platforms. Results have been carried on a PC on a Windows 2000 Server platform, AMD Athlon processor, 528Kb RAM, NVIDIA GeForce2. We computed the performance of the algorithm on two datasets: Hipip and IdrogenAtom of sizes 64^3 and 128^3 respectively. Table 1 shows the time in seconds for the pre-processing phase (tables filling, DT hierarchy construction procedures and object space error measurement). Table 2 shows the time in seconds for the Isosurface extraction for different values of resolution required. Results obtained with introduction of inheritance support are compared to results obtained with a "plain" version of the algorithm. Fig. 6 and 7 show progressive refinement of Hipip and IdrogenAtom obtained applying our algorithm with the inheritance paradigm active.

6. DISCUSSION AND FUTURE WORK

In this paper we have introduced a progressive algorithm and data structures for time-critical and memory-critical isosurface extraction. Providing a set of local rules for continuous geometric transitions (geomorphs) of one level of resolution

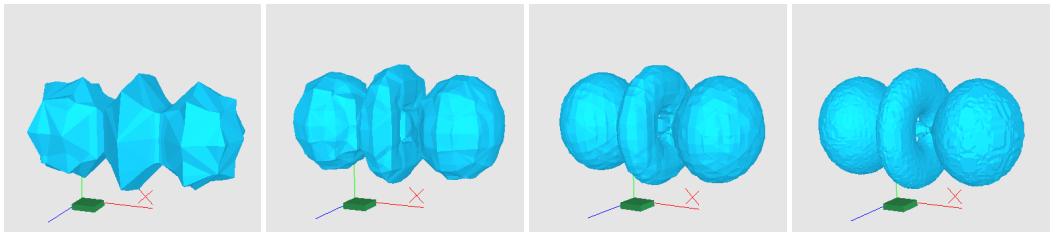

Figure 6: *Steps in a progressive isosurface computation from the volumetric IDROGEN-ATOM dataset, left to right.*

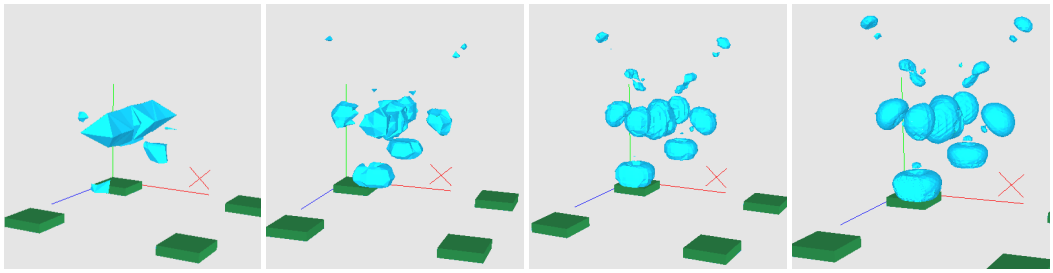

Figure 7: *Steps in a progressive isosurface computation from the volumetric HIPIP dataset, left to right.*

into the next we keep the same advantages of a hierarchical data structure without the overhead of keeping explicit the hierarchy structure. Our approach guarantees the generation of non-self intersecting surfaces while extracting adaptive levels of detail from the multi-resolution surface representation. Exploiting the subdivision scheme properties we can guarantee optimal time performance in isosurface extraction in spite of a minimal memory overhead (inheritance support). The regularity of the scheme makes our approach well suited for the design of an efficient run-time data partitioning and distribution algorithm to reduce the local memory requirement and overwork distributed environment potentiality currently only approached. Our present work regards performance testings of our paradigm to datasets of large size, our future work will regard the application of our technique in distributed environments and the development of data-partitioning schemes optimal for our framework.

References

1. P. Cignoni, D. Costanza, C. Montani, C. Rocchini, and R. Scopigno, *Simplification of tetrahedral meshes with accurate error evaluation*, IEEE Vis. '00, IEEE, pp. 85–92.
2. B. Gregorski, M. Duchaineau, P. Lindstrom, V. Pascucci, and K.I. Joy, *Interactive view-dependent rendering of large IsoSurfaces*, Proc. IEEE Vis. '02, IEEE, pp. 475–484.
3. Leif Kobbelt, $\sqrt{3}$ *subdivision*, Proc. of (SIGGRAPH-00),ACM, pp. 103–112.
4. Y. Livnat and C. Hansen, *View dependent isosurface extraction*, IEEE Vis. '98, IEEE, pp. 175–180.
5. Y. Livnat, H. W. Shen, and C. R. Johnson, *A near optimal isosurface extraction algorithm for structured and unstructured grids*, IEEE Transactions on Visual Computer Graphics **2** (1996), no. 1, 73–84.
6. W. E. Lorensen and H. E. Cline, *Marching cubes: a high resolution 3D surface construction algorithm*, SIGGRAPH '87 Conf. Proc., Computer Graphics, Volume 21, Number 4, July 1987, pp. 163–170.
7. V. Pascucci, C. L. Bajaj, and Daniel R. Schikore, *Fast isocontouring for improved interactivity*, 1996 Vol. Vis. Symp., IEEE, pp. 39–46.
8. V. Pascucci and P. Lindstrom, *Visualization of terrain made easy*, Proc. Vis. '01,IEEE, 2001.
9. Valerio Pascucci, *Slow growing subdivision (sgs) in any dimension: Towards removing the curse of dimensionality*, EUROGRAPHICS 02 Conf. Proc., pp. 451–460.
10. A. Plaza and G.F. Carey, *About local refinement of tetrahedral grids based on local bisection*, 5th International Meshing Roundtable (1996), 123–136.
11. M.-C. Rivara and C. Levin, *A 3-d refinement algorithm suitable for adaptive and multi-grid techniques*, Comm. in Appl. Numer. Meth. **8** (1992), 281–290.
12. H. W. Shen and C. R. Johnson, *Sweeping simplices: A fast isosurface extraction algorithm for unstructured grids*, Vis. '95 Proc., 143–150.
13. Luiz Velho and Denis Zorin, *4–8 subdivision*, Computer-Aided Geometric Design , no. 5, 397–427, Special Issue on Subd. Techniques.
14. Jane Wilhelms and Allen Van Gelder, *Octrees for faster isosurface generation*, ACM Trans. on Graphics (1992), no. 3, 201–227.