A Progressive Subdivision Paradigm (PSP)

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

[a] Visual Computing Group, Consiglio Nazionale delle Ricerche
[b] 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. Several techniques have been proposed in recent years ranging from simplification to wavelet analysis. Subdivision methods have been one of the most successful techniques applied to the multi-resolution 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. We introduce a technique that combines the flexibility of a progressive multi-resolution representation with the advantage of a recursive subdivision scheme. The main contributions of the paper are: (a) a progressive algorithm that builds a multi-resolution surface by successive refinements so that a consistent representation of the output is always available (b) a multi-resolution representation where any adaptively selected level of detail is guaranteed to be consistently embedded in 3D space (no self-intersections).

1. INTRODUCTION

Projects dealing with massive amounts of data need to carefully consider all aspects of data acquisition, storage, retrieval and navigation. The recent growth in size of large simulation datasets still surpasses the combined advances in hardware infrastructure and processing algorithms for scientific visualization. The cost of storing and visualizing such datasets is prohibitive, so that only one out of every hundred time-steps can be really stored and visualized. By 2004 ASCI will enable physics simulations on massively parallel computers (100 TeraFLOPS computers) generating upwards of 10 TeraBytes per hour with a potential total output of several petabytes per simulation. As a consequence interactive visualization of results is going to become increasingly difficult, especially as a daily routine from a desktop. High frequency of I/O operations start dominating the overall running time. The visualization stage of the modelling-simulation analysis activity, still the ideal effective way for scientists to gain qualitative understanding of simulations results, becomes then the bottleneck of the entire process. Such a problem poses fundamentally new challenges both to the development of visualization algorithms and to the design of visualization systems.

In this panorama starting from the point that it is no more possible to rely on fundamental assumptions on memory layout to speed up the access time, because of datasets too large to be kept in main memory or stored on a local disk, there is a need at system level to design the visualization process as a pipeline of modules able to process data in stages creating a flow of data that need themselves to be optimized globally with respect to magnitude and location of available resources. 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. Unfortunately generalization of such techniques to the volumetric case is not always straightforward. The more commonly used techniques currently refer to tensor product schemes whose high rate of refinement and lacks of good extensions to general cases, like unstructured meshes, limits their use in fundamental application areas like solid modelling, scientific visualization and mesh generation.

In this paper we present a new progressive visualization algorithm where an input grid is traversed and implicitly organized in a hierarchical structure (from coarse level to fine level) and subsequent level of detail are constructed and displayed to improve the output image. We uncouple the data extraction from its display allowing for a subdivision of the workload between different processes. The scheme allows us to render at any given time partial results while the computation of the complete hierarchy makes progress. The regularity of the hierarchy allows the creation of a good data-partitioning scheme

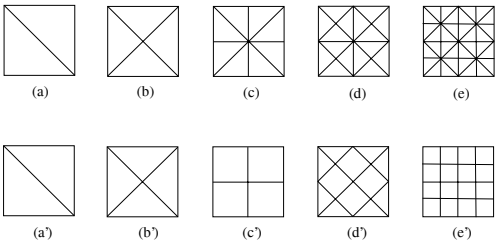

Figure 1. 4-8 recursive subdivision. (a-e) Classical longest edge bisection of a rectilinear grid. (a'-e') Equivalent $\sqrt{2}$ subdivision where pairs of adjacent triangles are merged into one square.

that allows us to balance processing time and data migration time. We restrict our attention to the case of meshes computed as isocontours of 3D scalar fields. Scientific 3D data (large physically based simulations, CT/MRI medical scans) are visualized using isosurfaces. Employment of multi-resolution data-structures for the output surface is a must for being able to achieve fast visualization performance and interactive response time. The approach introduced in this paper works as follows: to guarantee visual consistency we use a refinement primitive that updates a local portion of the output so that a consistent output mesh is maintained at any given time. This trick allows for asynchronous termination of the computation within a small constant delay, moreover the refinement operation is guaranteed to be performed within a small volumetric cell c and therefore the surface inside the cell is maintained persistent that guarantees a correct embedding of any adaptive mesh extracted at run time. 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. [25,26] For rectilinear volumetric input such a subdivision correspond to an adaptive hierarchical approximation. We select a set of cells intersecting the isocontour value and organize them in subsequent levels corresponding to different levels of refinement of the isocountour, each node corresponding to an atomic cell, cells can be grouped together in arbitrarily dimensioned bunches and sent to multiple processors to be elaborated. The final result becomes then a compositing, through graphics library that operates in distributed environments like WireGL, of the contribution of each bunch of cells coming from each one of the processors. In the next paragraph we analyze details and results of the developed paradigm.

2. PREVIOUS WORK

The topic of this paper concerns mainly multi-resolution surface generation, trend where several efforts have been made. The main contributions cover an area that goes from high quality surface simplification schemes to mesh refinement techniques.

Simplification Techniques. The simplification techniques existent in literature can be grouped into three basic categories: vertex removal,[28] edge contraction,[13] and triangle contraction.[10] Numerous methods have been proposed that coupling the efficiency of these primitives with the use of different error metrics [3,6] have allowed for the construction of high quality simplified objects. Multi-resolution representation of input data can be achieved through the application of simplification techniques maintaining a history DAG of the decimation process applied. The selective traversal of this representation allows for fast construction of adaptive levels of detail. [14]

Wavelet Analysis. Wavelet functions have been employed in the design of multi-resolution surface generation. Main advantage of the multi-resolution analysis at the basis of the wavelet approach is that it immediately gives a compact hierarchical multi-resolution data-structure with guaranteed error bounds. Unfortunately the set of meshes that can be processed is limited, requiring sometimes remeshing of the input. Hybrid approaches can be designed to take advantage from the quality of wavelet analysis maintaining the generality of a simplification scheme. [18] The general framework of wavelet analysis is formalized independently of the intrinsic/embedding dimension of the geometric object. This allows for example to achieve multi-resolution representation and analysis for volumetric data. [27]

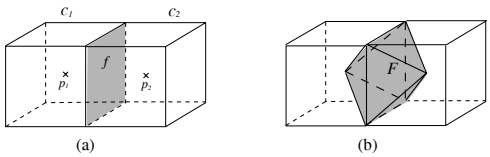

Figure 2. 3D cell refinement from tier 0 tier 1. (a)The two cells c_1 and c_2 in tier 0 . Their centers p_1 and p_2 are marked with two crosses. Their adjacency facet f is highlighted in gray. (b) The cell F of tier 1 (in gray) is the union of the pyramids $p_1 \lhd f$ and $p_2 \lhd f$.

Mesh Refinement. Other approaches in the meshing community have regarded subdivision methods for adaptive refinement of triangular meshes. Rivara's edge-bisection approach is one of the simplest and most flexible of these. A unique subdivision template[26] is used to recursively subdivide the cells of 2D meshes until a given adaptivity constraint is achieved. This implicitly yields a multi-resolution data-structure built starting form a coarse representation of the input. The approach generalizes immediately to 3D tetrahedralizations[25,33] and to higher dimension by performing the refinement process from the lower dimensional simplices of the mesh to the higher dimensional.

Subdivision Schemes. Using a recursive subdivision scheme one automatically achieves a hierarchical multi-resolution representation. This enables for example multi-resolution editing techniques. The quality of the generated meshes is dependent from the subdivision mask used. For triangular domains Loop[20] provides an approximating subdivision scheme converging to a surface that is C^2 almost everywhere. The exception is at extraordinary vertices, that do not have exactly six incident edges where the continuity decreases to C^1. The butterfly subdivision scheme[8] converges to an interpolating surface that is C^1 everywhere except for extraordinary points with exactly three or more than seven incident edges. A modified version[34] has been proposed that converges to a C^1 surface everywhere. For subdivision of quadrilateral domains two approaches have proposed: the Catmull-Clark scheme[2] and the interpolatory scheme by Kobbelt[16] to build smooth approximations of a coarse mesh. Bierman et al.[1] have improved the normal control mechanism for Catmull-Clark and Loop subdivisions. Subdivision schemes have been also used to build smooth vector fields.[31]

The approach we focus on belongs to the class of Subdivision Schemes. First introduced by Pascucci,[24] has been further developed by Gregorski et al..[11] Characteristics of the approach adopted is to respond well to three big issue typical of multiresolution approaches: vertex proliferation (mainly dependent on the subdivision mask adopted), efficient extraction of the refined surface, rendering in time-critical environment. Typical of tensor product refinement schemes, that normally increase the number of vertices by a factor of 8, vertex proliferation is an important issues especially when dealing with datasets of large size: as the dimension d of the mesh grows the complexity of the scheme augments leading to prohibitive refinement rates. The first to address this issue has been Kobbelt[17] with his $\sqrt{3}$-subdivision where the number of vertexes is increased at a slower rate than previous approaches. Velho et al.[30] and Duchaineau et al.[7] improved this approach with a different scheme definable as a $\sqrt{2}$ subdivision scheme (2D version of the algorithm at the base of our framework). For what concern efficient isocontouring the basic isocontour computation algorithm[21] is known to be inefficient since it wastes time exploring empty regions of the underlying volumetric data. Geometric space indexing[32] is sufficient to achieve a substantial speedup . Span space indexing techniques can provide further improvement with nearly optimal[19] or even optimal speedup.[5] The insufficient results achieved even with such optimal techniques require to use more flexible approaches[9,12] that allow to use the multi-resolution representation of the volumetric data to extract an adaptive level of detail for the output. The lack of an actual multi-resolution representation for the output limits the practical use of such schemes especially for large datasets since a substantial amount of computation may be required when adaption between different levels of resolution needs to be recomputed frequently. General solution to the problem of rendering in a time-critical environment have also been explored.[15] Such optimization techniques solve the "hard deadline" problem for high quality hierarchies if the time available is known in advance but do not consider the asynchronous termination problem or the case of dynamic change of the represented object.

The subdivision scheme at the base of our subdivision paradigm has taken into account several of the issues just presented: for what concern vertex proliferation our refinement scheme roughly doubles the number of vertices independently of the intrinsic dimension of the input mesh. For what concerns the capability of rendering in time-critical environment

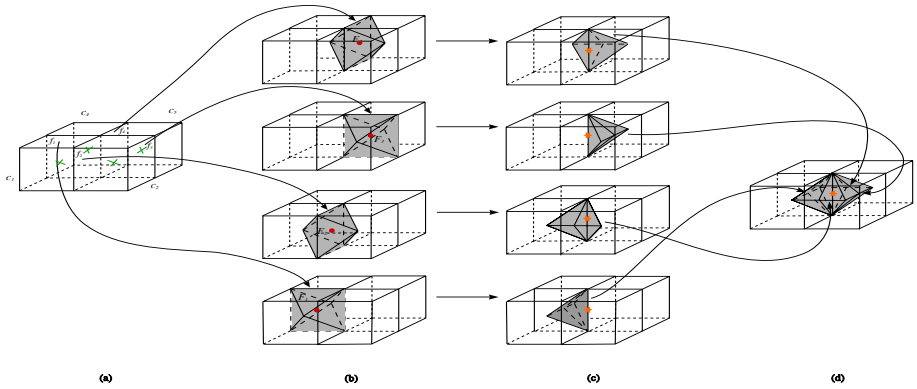

Figure 3. Cell refinement from tier 1 to tier 2. (a) Four cells c_1, c_2, c_3 and c_4 of tier 0 share, in pairs, the facets f_1, f_2, f_3 and f_4. The edge e is shared by all facets f_1, f_2, f_3 and f_4. (b) Each facet f_i generates a cell F_i. (c) Each cell F_i is decomposed into four pyramids only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

a consistent representation of the output is always available as long as a coarse representation of the input is given. For efficient isocontouring, the hierarchy is built only with cells intersecting the isocontour: no empty regions are visited, each level of the hierarchy correspond to a uniform level of resolution of the mesh, the hierarchy can be traversed to perform adaptive refinement of the input mesh. 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 has been introduced by Pascucci.[24] It follows the edge-bisection refinement introduced by Rivara[26] and proposed under a different approach by Velho and Zorin,[30] known as 4-8 subdivision or $\sqrt{2}$ subdivision. The techniques in these papers focus primarily on the case of surface subdivision and not the volumetric case. Figures 1 (a-e) show the subdivision scheme for a rectilinear grid. 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. The 4-8 subdivision rule follows these rules and adds an averaging step that repositions the vertices on the surface. To maintain the same combinatorial subdivision structure the scheme obliges to bisect, in each triangle, the oldest edge, i.e. the one that was not altered in the previous refinement step (that is always one and only one). Figures 1(a'-e') show the subdivision strategy applied to quadrilateral elements.[7] 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. We propose the edge-bisection refinement scheme from a new point of view based on a set of simple rules that characterize consistently the decomposition of a grid in simplices together with the recursive refinement of the derived simplicial mesh. The result is a new naming scheme that allows to represent an adaptive simplicial mesh with a very low memory foot print.

3.1. 3D Subdivision Scheme

As proposed in[24] we organize the subdivision process into levels and tiers. Each level l has four tiers, form 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.

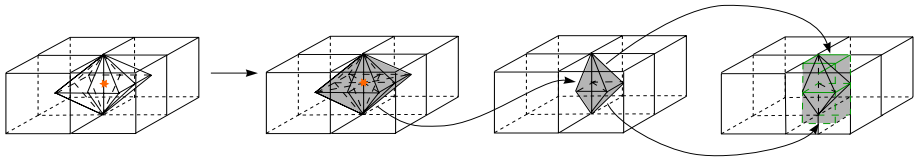

Figure 4. Cell refinement from tier 2 to tier 3.

3.2. Subdivision Rules

In this section we analyze the geometrical aspect of our subdivision scheme. The terms used (like "centers" and "diamonds") are to be considered in a combinatorial fashion, given more to provide an intuitive idea of the described structure than referring to their actual geometrical meaning (e.g. with the term "center" of a cell/face we indicate the relation between a point and the cell/face it belongs, it does not necessarily correspond to its actual geometric position). The subdivision schema is similar to the 2D case described above. The only inconvenient is that augmenting of one level augments, as direct consequence, the subdivision process of one step (in the 4-8 subdivision square-shaped cells, i.e. basic case or tier 0, are obtained after the second refinement step) indicated as tier 3. In the following paragraphs we will analyze each refinement step in details.

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.

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 on 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).

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. During this last merge step all the spurious edges introduced during the refinement procedure are removed. Figure 4 shows the construction of one cell of tier 3 from a cell of tier 2.

3.3. Refinement Characterization

Cells generated by our subdivision techniques can be easily characterized.

DEFINITION 3.1. *A* diamond *is a cell that can be combinatorially partitioned into a set of simplices all sharing an edge, called* axis *of the diamond.* We show that all the cells generated by our scheme are diamonds. To satisfy this property we only need each facet of the base mesh to be a polygon.

PROPOSITION 3.1. *Consider a complex C where all the 2D cells are simple loops. If we use C as the base mesh, the subdivision generates only diamonds cells.*

Proof. Since all the facets of the cells in C are simple loops, all the cells generated at the first tier of the subdivision procedure are either pairs of pyramids or single pyramids (for sharp features). In the first case the axis of the diamond is the edge connecting the apices of the two pyramids. In the second case the axis connects the apex of the pyramid with the center of its base. □

At the second tier the cells are diamonds by construction since they are just a set of tetrahedra merged along a common edge: the axis of the diamond. At the third tier each cell is the set of tetrahedra sharing the edge connecting the center of a coarse cell with one of its vertices. This edge is the axis of the diamond.

The first interesting aspect of this subdivision scheme, is that given a mesh representation model it can be organized hierarchically in terms of embedded entities that we call diamonds. By construction, the topology of such hierarchy is implicit to the diamonds themselves: each cell/diamond is a unique and independent nucleus that stores in itself all the information needed. From its center, characterized by three index (i, j, k), it is possible to derive tier, type, orientation and refinement level it represents. Through simple mathematical rules it is possible to identify its sons. The overall mesh is in fact seen as a collection of geometric primitives (the diamonds) that for the regularity of the subdivision criteria need a very low footprint to be represented. The mesh can be seen as a collection of embedded diamonds, 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. Another point worth noting is that diamonds as entities do not really exists, only their centers exists. Through the center it is possible in fact to derive diamond shape (that is type and orientation) and vertexes position with just a couple of unitary operations. The regularity of the diamond shape allows in fact to gather the diamond vertexes simply adding a δ constant to the center coordinates, diamond vertexes are needed only for sons generation. In case of regular grids the constant is fixed for each type of diamond and dependent in magnitude to the level of refinement reached (easily derivable from the coordinates of the center). 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. We have focused our initial efforts on the refinement of regular grid nevertheless the framework has been designed to be independent of the kind of input mesh.

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" schema. 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. The simplification may respond to view-dependent and adaptive constraints and allow for speeding up the rendering process of the volume data.

Our multiresolution framework can be seen as a two phases process:

- Pre-processing Phase: where some auxiliary information (data, range, approximation error) are extracted;
- Rendering Phase: where the mesh is traversed, at run-time, to extract the model under appropriate constraints (view-dependent, adaptiveness, error-based criteria).

In the present context the *pre-processing* phase comprehend: volume subdivision for extraction of all the data and their organization in tables. The *rendering phase* consists instead of traversal of the mesh and isocontour extraction following appropriate approximation 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 schema the pre-processing phase correspond to a formalization of the dataset with our subdivision schema. 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

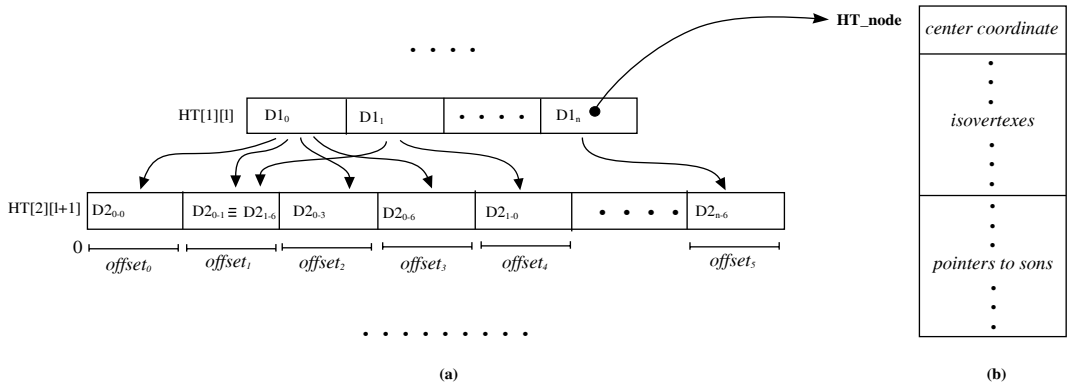

Figure 5. Example of the two level hierarchy tree HT for level *l* and *l+1*:(a)representation of two level (*l* and *l+1* respectively) of the hierarchy tree (b)internal structure of a node of the hierarchy tree HT.

vertices are added. Vertexes are added at each step following a breadth first priority (BFP) policy: the same subdivision step is implemented for all the diamonds in the same level before the next step is taken. Through the subdivision process we extract the data embedded in the volume and calculate the *range* belonging to each diamond (with range we identify the min and max field values contained in a diamond). In a successive step we traverse the volume in depth first order to compute the *approximation error* belonging to each diamond (i.e. maximal inaccuracy generated when we represent the field in the interior of the diamond by interpolation of field values on the diamond vertexes). These results are organized in tables as 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 table: data, range, field. Each table has dimension $(2^N+1)\times(2^N+1)\times(2^N+1)$ that is equal to the dimension of the volume, and with 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, that is linear in the number of cells/nodes. Therefore at subdivision step $l+1$ we need to have in memory only the diamonds belonging to level l, those diamonds are discarded as soon as level $l+1$ is completed.

Computing the approximation error is a bit more complex. An explicit representation of the hierarchy is needed to compute the error accuracy. The error metric we adopt assures an overestimation of the error introduced by the approximation but requires to be able to move easily from sons to fathers; because, by construction, diamonds share sons (i.e. a diamond of level $l+1$ is generated by the fusion of parts of diamonds of level l), computing father/son relation is not straightforward, though possible. We have decided to give easiness of implementation top priority, at least for now, for this reason only during error calculation the hierarchy organization is made explicit with a tree like data-structure. It consists of a Diamond Tree (DT) where each diamond of the same type and resolution shares level with its diamond siblings. Each node of the diamond tree stores only the indexes of the diamond center and pointers to its sons. A depth first traversal of DT allows for the calculation of the error. Results are stored in a $(2^N+1)\times(2^N+1)\times(2^N+1)$ table with the same characteristics as the data and range tables.

A fourth flag table is used to keep track of those diamonds that have been already visited. This information is needed at runtime and at runtime it is filled, its use will be explained in detail in Section 4.2.1.

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 $(2^{(N-1)},2^{(N-1)},2^{(N-1)})$ (coincident with the bounding box of the entire model)

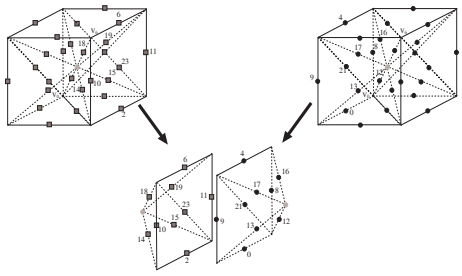

Figure 6. Isovertex inheritance for a tier1 diamond from its two tier0 diamond fathers. The diamond inherits exactly 12 vertexes, 8 from each father but 4 in common on the shared face.

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. The mesh traversal does not touch all the dataset cells (i.e. not all the cells are visited). Only those cells intersecting the isosurface (active diamonds) are visited and eventually refined (i.e. only their *active sons* are generated through subdivision). Isosurface extraction is performed during the traversal. Each active diamond is visited, the isosurface it intersects is extracted and sent to the renderer, if refinement is needed the cell is further subdivided to generate its active sons and then discarded, or simply discarded if no further refinement is needed.

Discrimination between active and non-active diamonds is possible simply referring to the range of isovalues corresponding to the diamond (stored in the min/max table). Refinement of a diamond is decided in function of error metrics. 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.

To avoid redundant visits of the same diamond (we recall that a diamond has more than one father, so we can reach a diamond from different journeys) we make use of a sort of "flag table" with the same characteristics (dimensions, access criteria) as the data, range and error tables mentioned in Section 4.1.1.

4.2.1. Isosurface Extraction

The 3D mesh partitions subdivides the region of space of interest in diamonds. 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. As the edge-bisection algorithm makes progress new function values are added and a more detailed definition of the function $\mathcal{F}(x)$ is obtained.

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. Essentially fathers and sons diamonds shares isovertexes between each other. 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 3.3 the topology of the refinement hierarchy is implicit to the cells) to support vertex inheritance we need to explicit 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 tree-like data-structure, of only two level, called Hierarchy Tree (HT). Each node in HT stores diamond center, computed isovertexes and sons position; fathers shares sons between each other but it is easy to avoid duplication of sons simply exploiting the Flag Table: if diamond d_j of indexes (i,j,k) is inserted in HT at position p_j (we do not to store the level because it is inferable from i,j,k) we insert the value d_j in position (i,j,k) of FT, this information allows us to determine if d_j is an active diamond, that is, has been already created and visited, and in case which position in HT it occupies. Figure5 shows an example

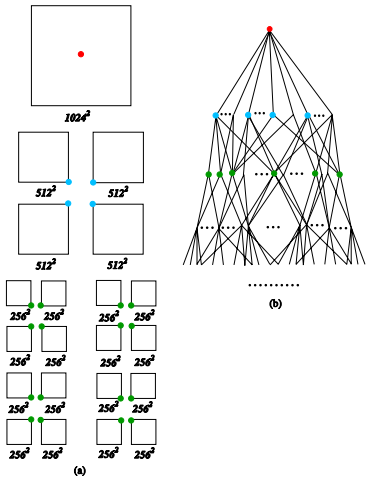

Figure 7. Data partitioning scheme of a 2D datasets of 1024^2:(a) Partitioning of the datasets in 16 blocks of 256^2, each block can be sent to a different processor, colored vertexes are the only vertexes for which the error is not computed, colored vertexes can be considered like *root* of the block they belong to; (b) corresponding position in DT of each *block-root* node, they all belong to the first three level of DT.

of such structure. The HT structure is used only at runtime. Supporting inheritance has gains and loss, loss in terms of memory overhead, gains in terms of speeding up of the rendering process avoiding useless calculations. Advantages and disadvantages 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[4] (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. By construction the field error can be considered as an upper bound of the error introduced by ending the refinement process at level l instead of level $l+1$. The field space error is computed traversing the hierarchy DT from bottom to top in the pre-processing 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 project 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:

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

The projected error decreases with the distance from the viewpoint. If we consider the perspective projection onto a plane: $\lambda = \frac{w}{2\tan\frac{\varphi}{2}}$, where w is the number of pixels along the field of view φ. Equation 1 correspond to a projection onto a sphere and not onto a plane, so a more appropriate choice for λ would be $\lambda = \frac{w}{\varphi}$. After this the error space ρ is compared against a user-specified screen space error tolerance. In computing our screen space error we follow the approach adopted by Lindstrom and Pascucci.[23] We compute the bounding sphere $\mathbf{B}_i$ of ray r_i of each diamond d_i and consider *active* all the cells inside B_i that satisfy:

$$(\frac{1}{k}\delta_i + r_i)^2 > \|\mathbf{p}_i - \mathbf{e}\|^2$$

where $k = \frac{\tau}{\lambda}$ constant during each refinement.

Dataset	Size	Pre-processing Time (sec.)
Hipip	$64 \times 64 \times 64$	3.9 secs
Hydrogen	$128 \times 128 \times 128$	10.5 secs
Bonsai	$256 \times 256 \times 256$	32.65 secs

Table 1. Computational time required for the pre-processing phase of the algorithm. Performances computed over the HIPIP dataset (64^3), the IDROGEN-ATOM dataset (128^3) and the Bonsai dataset (256^3) .

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:

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 large meshes, the possibility to discard cells, not intersecting the isocontour, means lots of computational time saved during mesh traversal. Moreover 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. 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, never at run-time, and after the error calculation is completed it can be discarded. Two main points are worth noting: necessity and complexity of introducing this data structure. The diamond tree 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. At least for the present results we have decided to give easiness of implementation top priority allowing for explicit hierarchy construction only during the pre-processing step. We made such decision considering the complexity introduced by DT in terms of memory occupancy and time complexity when working with large amount of data. 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. As shown in Fig 7 a good data-partitioning scheme can distribute evenly each block, leaving out from the error calculation only the block vertexes that by themselves correspond to the first levels of the hierarchy tree DT, levels that can be easily traversed.

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" at run time is FT, this table is not really filled at run time but only some of its value are updated during the hierarchy traversal. The regularity of the subdivision mask applied and the organization of the information in Tables allows 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 needs only the center coordinates of the diamond we are interested and is performed 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. To be more specific the memory requirements of HT is equal to: 3 short for the center, 24 short for the shared vertexes, 8 short for pointers to sons for a total of 70 bytes. This 70 bytes must be multiplied by the number m of diamonds belonging to the level under refinement. The average value of m depends on the level of subdivision and the percentage of cells containing the isosurface we are searching (around 10-20% of the total). 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 vertex not in common/inherited from the fathers reducing the operation of interpolation of a factor of 3. Each diamond needs to recalculate only the isovertexes laying on split or "new" edges introduced by the subdivision peculiar to that level. In this way a tier0 diamond needs to calculate only 8 new isovertexes (instead of 26), a tier1 diamond needs to calculate only 6 new isovertexes (instead of 18) and a tier2 diamond needs to calculate only 10 new isovertexes (instead of 34). Passing the inherited vertexes from father to son can be done with three operations each of unitary cost ($O(1)$). The first

Dataset	Resolution	IsoSurface Extr. (w/ Inh.)	IsoSurface Extr. (w/o Inh.)
Hipip ($64 \times 64 \times 64$)	60%	0.1 secs	0.3 secs.
	85%	0.4 secs	1.0 secs.
	100%	0.9 secs	2.5 secs.
Idrogen-Atom ($128 \times 128 \times 128$)	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)

operation consists, given a diamond of level l, in locating exactly the position of each diamond son in the $l+1$ level of the hierarchy and put the father's values in the isovertex record local to the son (operation of unitary cost because stored in FT). The order in which the vertexes should be inherited is known for construction, that is, because of the regularity of the subdivision we always know which son inherits which vertex (see figure 6). Results show (see Section 5) that, supporting inheritance, the computational time needed for the isosurface extraction is reduced by a factor of 3. Each operation of interpolation requires a constant number of arithmetic operations involving sum, subtraction, multiplication and division, for very large datasets the overall computational time saved it is shown to be worth the overhead.

Computational Behaviour Following the proofs provided in [22] our algorithm shows an optimal behaviour for the computation of large isocontour keeping to a reasonable factor the overhead induced by the computation of small ones. As far as the hierarchy construction proceeds we end up with a mesh organized into a tree-like structure with the coarse level having size $O(1)$ and the finest level having size $O(n)$. We know from construction that every cell in the hierarchy intersects the isocontour in a constant number of simplices (line segments in 2D, triangles in 3D), from this follows that no isocontour can have size larger than $O(n)$. For convention we define *large* an isocontour of dimension $\Theta(n)$ and *small* an isocontour that is $o(n^h)$,$\forall h > 0$, h constant, we also approximate the tree-like structure with a binary tree approximation that does not influence the purpose of the proof.

THEOREM 4.1. *Given an isocontour of output size $k = \Theta(n^h)$, for some constant $h > 0$, the corresponding hierarchy generated by the progressive isocontouring algorithm has size $\Theta(k)$ for $h = 1$ and has size $O(klogk)$ for $h < 1$.*

Proof. The hierarchy tree is a complete balanced tree where the finest resolution has size $\Theta(n)$. Hence the height of the input hierarchy is $\log_n + O(1)$. Since the progressive isocontouring algorithm (i.e. the process that deals with the isosurface extraction and progressive construction of the output) produces one level in the output hierarchy for each level in the input hierarchy the height of the output hierarchy is also $\log_n + O(1) = \frac{1}{h}\log_k + O(1)$. For $h = 1$ this implies that the overall output hierarchy has size $O(k \log k)$. $\diamondsuit$ □

Basically then for non-small contours the output is a balanced tree which is optimal for large contours.

THEOREM 4.2. *Given an isocontour of size $k = \Theta(n^h)$, for some constant $h > 0$, the time necessary to compute the isocontour is $\Theta(k)$ for $h = 1$ and has size $O(klogk)$ for $h < 1$.*

Proof. Follows immediately from the previous Theorem and from the fact that the progressive isocontouring algorithm generates an output tree of l nodes in l steps of constant time each. $\diamondsuit$ □

For large contours the computation time is linear in the size of the finest output resolution and hence optimal. Otherwise a logarithmic penalty factor is introduced by the traversal of the input hierarchy.

THEOREM 4.3. *Given an isocontour of size k, the corresponding hierarchy size and computation time is $O(k \log n)$.*

Proof. From the same observation of Theorem 4.1 it derives that the height of the output tree is $\log n + O(1)$. Hence the hierarchy's overall size is $O(k \log n)$. As shown in Theorem 4.2 the computation time is just proportional to the overall size of the output. $\diamondsuit$ □

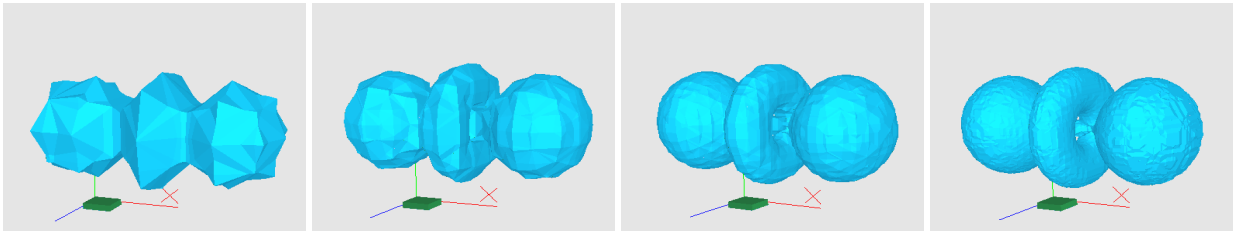

Figure 8. Steps in a progressive isosurface computation from the volumetric IDROGEN-ATOM dataset, left to right.

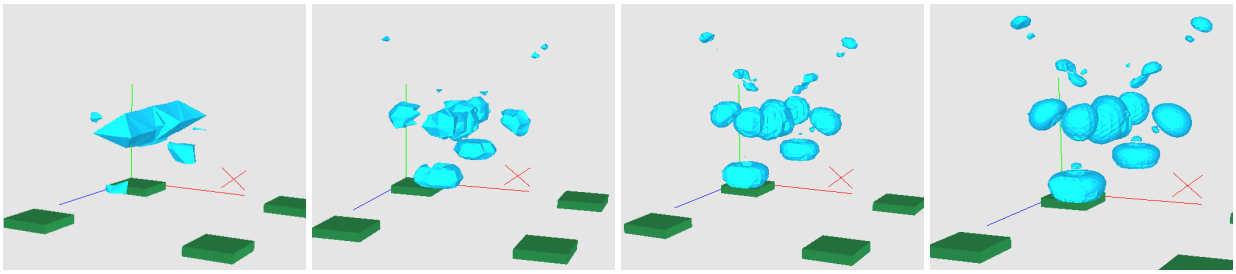

Figure 9. Steps in a progressive isosurface computation from the volumetric HIPIP dataset, left to right.

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 three datasets: Hipip, IdrogenAtom and Bonsai of sizes 64^3, 128^3 and 256^3 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. 8 and 9 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 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. Henning Biermann, Denis Zorin, and Adi Levin, *Piecewise smooth subdivision surfaces with normal control*, Proc. of the Computer Graphics Conf. 2000 (SIGGRAPH-00), ACM, July 23–28 2000, pp. 113–120.
2. E. Catmull and J. Clark, *Recursively generated B-spline surfaces on arbitrary topological meshes*, Computer-Aided Design (1978), 350–355.

3. A. Ciampalini, P. Cignoni, C. Montani, and R. Scopigno, *Multiresolution decimation based on global error*, The Visual Computer **13** (1997), no. 5, 228–246.
4. P. Cignoni, D. Costanza, C. Montani, C. Rocchini, and R. Scopigno, *Simplification of tetrahedral meshes with accurate error evaluation*, IEEE Vis. '00 (VIS '00), IEEE, October 2000, pp. 85–92.
5. P. Cignoni, C. Montani, E. Puppo, and R. Scopigno, *Optimal isosurface extraction from irregular volume data*, Proceedings of the Symposium on Volume Visualization (New York), ACM Press, October 28–29 1996, pp. 31–38.
6. J. Cohen, D. Manocha, and M. Olano, *Simplifying polygonal models using successive mappings*, IEEE Visualization '97 (VIS '97) (Washington - Brussels - Tokyo), IEEE, October 1997, pp. 395–402.
7. B. Gregorski Duchaineau and K.I. Joy, *Smooth centroid bintree subdivision surfaces with local wavelets*, Tech. report, 2001, UCDavis Technical Report.
8. Nira Dyn, David Levin, and John A. Gregory, *A butterfly subdivision scheme for surface interpolation with tension control*, ACM Transactions on Graphics **9** (1990), no. 2, 160–169.
9. Klaus D. Engel, Rüdiger Westermann, and Thomas Ertl, *Isosurface extraction techniques for web-based volume visualization*, Proc. of the 1999 IEEE Conference on Visualization (VIS-99), ACM, October 25–29 1999, pp. 139–146.
10. Tran S. Gieng, Bernd Hamann, Kenneth L. Joy, Gregory L. Schussman, and Isaac J. Trotts, *Constructing hierarchies for triangle meshes*, IEEE Transactions on Visualization and Computer Graphics **4** (1998), no. 2, ISSN 1077-2626.
11. B. Gregorski, M. Duchaineau, P. Lindstrom, V. Pascucci, and K.I. Joy, *Interactive view-dependent rendering of large IsoSurfaces*, Proc. IEEE Vis. 2002, IEEE, October 27– November 1 2002, pp. 475–484.
12. Roberto Grosso and Thomas Ertl, *Progressive iso-surface extraction from hierarchical 3D meshes*, Computer Graphics Forum (David Duke, Sabine Coquillart, and Toby Howard, eds.), vol. 17(3), Eurographics Association, 1998, pp. 125–135.
13. H. Hoppe, *Progressive meshes*, Proceedings of SIGGRAPH '96 (1996), 99–108.
14. Hugues Hoppe, *View-dependent refinement of progressive meshes*, SIGGRAPH 97 Conference Proceedings (Turner Whitted, ed.), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, August 1997, ISBN 0-89791-896-7, pp. 189–198.
15. James T. Klosowski and Claudio T. Silva, *Rendering on a budget: A framework for time-critical rendering*, Proc. of the 1999 IEEE Conference on Visualization (VIS-99) (N.Y.), ACM, October 25–29 1999, pp. 115–122.
16. L. Kobbelt, *Interpolatory subdivision on open quadrilateral nets with arbitrary topology*, Computer Graphics Forum (Proc. EUROGRAPHICS '96), 15(3), 1996, Eurographics '96 issue, pp. 409–420 (en).
17. Leif Kobbelt, $\sqrt{3}$ *subdivision*, Proceedings of the Computer Graphics Conference 2000 (SIGGRAPH-00) (New York) (Sheila Hoffmeyer, ed.), ACMPress, July 23–28 2000, pp. 103–112.
18. Leif Kobbelt, Swen Campagna, Jens Vorsatz, and Hans-Peter Seidel, *Interactive multi-resolution modeling on arbitrary meshes*, SIGGRAPH 98 Conf. Proc., ACM, Addison Wesley, July 1998, pp. 105–114.
19. 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.
20. Charles Loop, *Smooth spline surfaces over irregular meshes*, Computer Graphics **28** (1994), no. Annual Conf. Series, 303–310.
21. 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.
22. V. Pascucci and C. Bajaj, *Time critical isosurface refinement and smoothing*, Proceedings of the ACM/IEEE Volume Visualization and Graphics Symposium 2000 (New York), ACM Press, 2000, pp. 33–42.
23. V. Pascucci and P. Lindstrom, *Visualization of terrain made easy*, Proceedings Visualization 2001 (T. Ertl, B. Hamann, and A. Varshney, eds.), IEEE Computer Society Technical Committe on Computer Graphics, 2001.
24. Valerio Pascucci, *Slow growing subdivision (sgs) in any dimension: Towards removing the curse of dimensionality*, EUROGRAPHICS 02 Conference Proceedings, Annual Conference Series, EUROGRAPHICS, sept 2002, pp. 451–460.
25. A. Plaza and G.F. Carey, *About local refinement of tetrahedral grids based on local bisection*, 5th International Meshing Roundtable (1996), 123–136.
26. 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.
27. Ricardo Sánchez and Marcelo Carvajal, *Wavelet based adaptive interpolation for volume rendering*, IEEE Symposium on Volume Visualization, IEEE, ACM SIGGRAPH, 1998, pp. 127–134 (en).
28. William J. Schroeder, *Decimation of triangle meshes*, Proceedings of the Spring Cray Users Group Conference (33rd Spring CUG'94) (San Diego, CA), March 1994, G-E, pp. 87–91.
29. O. G. Staadt, M. Gross, and R. Weber, *Multiresolution compression and reconstruction*, Proceedings of IEEE Visualization 1997, IEEE, 1997 (en).
30. Luiz Velho and Denis Zorin, *4–8 subdivision*, Computer-Aided Geometric Design **18** (2001), no. 5, 397–427, Special Issue on Subdivision Techniques.
31. Henrik Weimer and Joe Warren, *Subdivision schemes for fluid flow*, Siggraph 1999, Computer Graphics Proceedings (Los Angeles) (Alyn Rockwood, ed.), Annual Conference Series, ACM Siggraph, Addison Wesley Longman, 1999, pp. 111–120.
32. Jane Wilhelms and Allen Van Gelder, *Octrees for faster isosurface generation*, ACM Transactions on Graphics **11** (1992), no. 3, 201–227.
33. Yong Zhou, Baoquan Chen, and A. Kaufman, *Multiresolution tetrahedral framework for visualizing regular volume data*, IEEE Visualization '97 (VIS '97) (Washington - Brussels - Tokyo), IEEE, October 1997, pp. 135–142.
34. Denis Zorin, Peter Schroeder, and Wim Sweldens, *Interpolating subdivision for meshes with arbitrary topology*, SIGGRAPH 96 Conf. Proc., ACM, August 1996, pp. 189–192.