A Progressive Subdivision Paradigm (PSP)

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 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.

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. In this paper we present a new approach 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), (c) a regular hierarchy that allows the creation of a good data-partitioning scheme to balance processing time and data migration time. We restrict our attention to the case of meshes computed as isocontours of 3D scalar fields. 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]. 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 a sort of tree like structure where each level correspond to a different level of refinement of the isocountour, each node corresponds to an atomic cell, cells that 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

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 as isosurfaces are an effective technique to analyze 3D scalar fields generated from large-scale numerical simulations. Three main classes of algorithms used to perform such a task can be identified. The first one groups all those methods that overworked and improved the well known March-

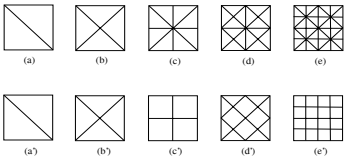

Figure 1: 4-8 recursive subdivision. (a-e) Classical longest edge bisection of a rectilinear grid. (a'-e') Equivalent $\sqrt{2}$ subdivision where pairs of adjacent triangles are merged into one square.

ing 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]; 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 Surfaces. First introduced by Pascucci [9], has been further developed by Gregorski et al.[2]. While borrowing the basic refinement schema from Pascucci [9] our approach differs from Gregorski et al. [2] in one main aspect: to support refinement and derefinement of the mesh Gregorski [2] need to keep a data structure of two queues (merge and split) whose accesses must be guaranteed to be atomic (for consistency of the data). Our algorithm instead is based on an implicit coarse to fine refinement scheme that does not need to keep any extra data structure (except the one used to support vertex inheritance, sec. 5.2, used to improve isosurface extraction but not essential to the refinement algorithm itself) we can also just make use of the double buffer to perform refinement and rendering. Moreover characteristic of the approach adopted is to respond 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. 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 (see [3], [13]). For what concerns this issue 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, 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 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 in [9]. It follows the edge-bisection refinement introduced in [11], and proposed under a different approach in [13], 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, Figures 1(a'-e') show the subdivision strategy applied to quadrilateral elements. 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 footprint.

4 3D Subdivision Scheme

As proposed by Pascucci in [9] 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 $l+1$. In our scheme

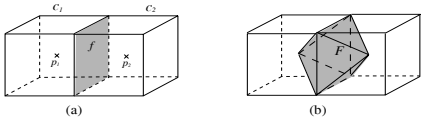

Figure 2: 3D cell refinement from tier 0 tier 1. (a)The two cells c_1 and c_2 in tier 0 . Their centers p_1 and p_2 are marked with two crosses. Their adjacency facet f is highlighted in gray. (b) The cell F of tier 1 (in gray) is the union of the pyramids $p_1 \lhd f$ and $p_2 \lhd f$.

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.

4.1 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.

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 $p_j \lhd f$ (see Fig. 2):

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

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 onto 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.

4.2 Refinement Characterization

Cells generated by our subdivision technique can be easily characterized. A *diamond* is a cell that can be combinatorially partitioned into a set of simplices all sharing an edge. All the cells generated by our scheme are diamonds (for reason of space we refer to [9] for the proof of what just stated). The first interesting aspect of this subdivision scheme, is that given a mesh representation model it can be organized hierarchically in terms of embedded diamond-entities. 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 a diamond 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 diamond 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. 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. A center can be used as *key* to derive diamond shape (that is type and ori-

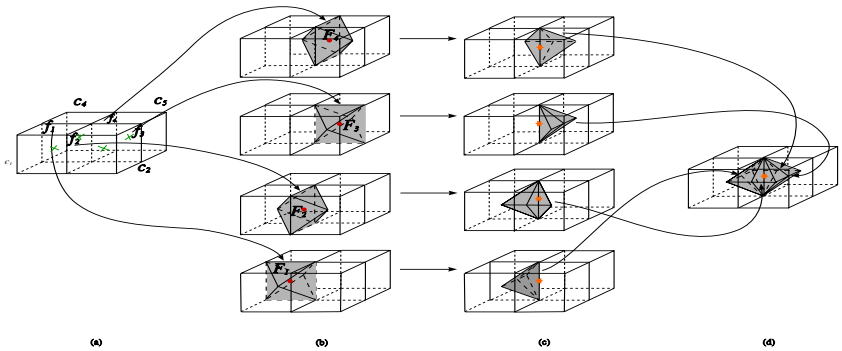

Figure 3: Cell refinement from tier 1 to tier 2. (a) Four cells c_1, c_2, c_3 and c_4 of tier 0 share, in pairs, the facets f_1, f_2, f_3 and f_4. The edge e is shared by all facets f_1, f_2, f_3 and f_4. (b) Each facet f_i generates a cell F_i. (c) Each cell F_i is decomposed into four pyramids only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

entation) 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. Overworking these properties we have developed a Progressive Subdivision Paradigm (PSP) oriented to the visualization of large datasets. 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 grids nevertheless the framework has been designed to be independent of the kind of input mesh.

5 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 constraint and allow for speeding up the rendering process of the volume data. qOur multiresolution framework can be seen as a two phases process: a *pre-processing phase* where some 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). 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. $(2^N+1)\times(2^N+1)\times(2^N+1)$.

5.1 Pre-Processing Phase

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 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

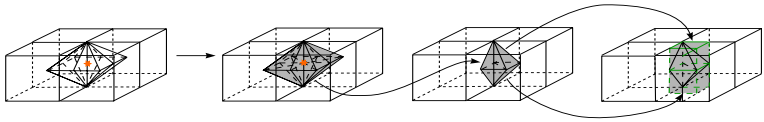

Figure 4: Cell refinement from tier 2 to tier 3.

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 . These results are organized in tables as described in the following paragraph.

5.1.1 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 and field. Each table has dimension 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 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 in the hierarchy; 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 table with the same characteristics as the data and range tables.

5.2 Rendering Phase

Extraction and refinement of the isosurface are executed at runtime performing traversals of the mesh representation model. Starting from the center of the bounding box of the model we proceed following our subdivision rules. 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 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. The isosurface is extracted even if further refinement is needed, this allows us to maintain always a consistent version of the model available and to render at any given time partial results while the computation makes progress.

5.2.1 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. As the edge-bisection algorithm makes progress new

function values are added and a more detailed definition of the function 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. To exploit this property we have decided to try to support the inheritance of shared vertexes between diamonds. Because there is no explicit representation of the hierarchy produced by the subdivision process (as mentioned in Sect 4.2 the topology of the refinement hierarchy is implicit to the cells) to support vertex inheritance we need to locally explicit the hierarchy for one level 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 two-level data-structure (HT). Each node in HT stores diamond center and computed isovertexes at level l . 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 of this choice are analyzed in sections 5.2.3 and 6.

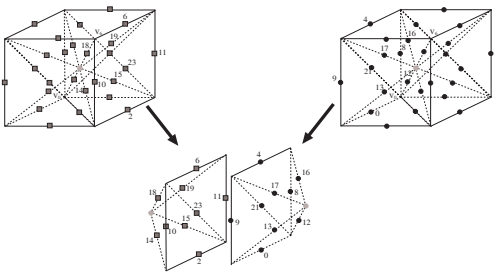

Figure 5: Isovertex inheritance for a tier1 diamond from its two tier0 diamond fathers. The diamond inherits exactly 12 vertexes, 8 from each father but 4 in common on the shared face.

5.2.2 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 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 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:

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

The projected error decreases with the distance from the viewpoint. In computing our screen space error we follow the approach adopted by Lindstrom and Pascucci in [8]. 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 pre-defined constraints.

5.2.3 Memory Occupancy and Overheads

In our strategy we have decided to store all the mesh related information in tables. We have three main table: data (IT), field value range(MT) and error (ET). Each of them has size equal to the size of the mesh grid. Tables requires storage and computational time for filling. Because table filling is one of the heaviest operations we perform, for this reason it is restricted to the phase of pre-processing. 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 of discarding cells, not intersecting the isocontour, means 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 also by the Diamond Tree that we

need to create for computing the error approximation. This structure is used only during the preprocessing 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 5.2.2) 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 pointers to 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. 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. 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 needs only the center coordinates of the diamond we are interested in and it is performed in constant time. The introduction of vertex-inheritance support causes an overhead in memory requirements (HT data-structure). 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 vertexes 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. Passing the inherited vertexes from father to son can be done with three operations each of unitary cost ($O(1)$). The first 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 HT). The order in which the vertexes should be inherited is known by construction (see figure 5). Results show (see Section 6) 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 large datasets the overall computational time saved it is shown to be worth the overhead.

6 Results

Our algorithm has been implemented in C^{++} and developed on both SGI and Windows platforms. Results have been carried on 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.

Dataset	Size	Pre-proc. Time(sec.)
Hipip	64^3	3.9 secs
Hydrogen	128^3	10.5 secs

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

Dataset	Res.	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)

7 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 keeping at reasonable rate the proliferation of vertexes. Exploiting the subdivision scheme properties we can guarantee optimal time performance in isosurface extraction in spite of a minimal memory overhead (inheritance support). Adopting a marching tetrahedra algorithm to perform isosurface extraction we end up producing more triangles than well known techniques as the one based on Marching Cubes or Dual Contouring, our future effort will be the development of a more efficient approach for extracting the contour. Nevertheless with respect to current techniques the regularity of our scheme allows us to easily handle the presence of sharp features and to produce adaptive refinements without needing special rules to handle cells connecting regions at different resolution. Our approach is also 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, Oct. 2000, pp. 85–92.

[2] B. Gregorski, M. Duchaineau, P. Lindstrom, V. Pascucci, and K.I. Joy, *Interactive view-dependent rendering of large Iso-Surfaces*, Proc. IEEE Vis. 2002 , IEEE, Oct. 27– Nov. 1 2002, pp. 475–484.

[3] Leif Kobbelt, $\sqrt{3}$ *subdivision*, Proc. of the Computer Graphics Conf. 2000 (SIGGRAPH-00), ACMPress, July 23–28 2000, pp. 103–112.

[4] Y. Livnat and C. Hansen, *View dependent isosurface extraction*, IEEE Visu. '98, IEEE, Oct. 1998, 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 Trans. on Visual Computer Graphics (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. Symposium, IEEE, Oct. 1996, ISBN 0-89791-741-3, pp. 39–46.

[8] V. Pascucci and P. Lindstrom, *Visualization of terrain made easy*, Proceedings Visualization 2001, IEEE, 2001.

[9] Valerio Pascucci, *Slow growing subdivision (sgs) in any dimension: Towards removing the curse of dimensionality*, EUROGRAPHICS 02 Conf. Proc., , EUROGRAPHICS, Sept 2002, 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. (1992), 281–290.

[12] H. W. Shen and C. R. Johnson, *Sweeping simplices: A fast iso-surface extraction algorithm for unstructured grids*, Vis. '95 Proc. (1995), 143–150.

[13] Luiz Velho and Denis Zorin, *4–8 subdivision*, Computer-Aided Geometric Design (2001), no. 5, 397–427, Special Issue on Subdivision Techniques.

[14] Jane Wilhelms and Allen Van Gelder, *Octrees for faster isosurface generation*, ACM TOG (1992), no. 3, 201–227.

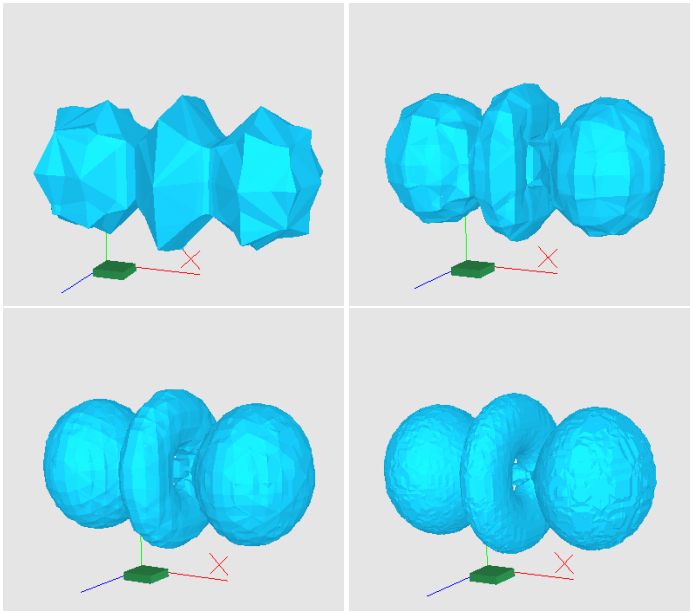

Figure 6: Steps in a progressive isosurface computation from the volumetric IDROGEN-ATOM dataset, left to right.

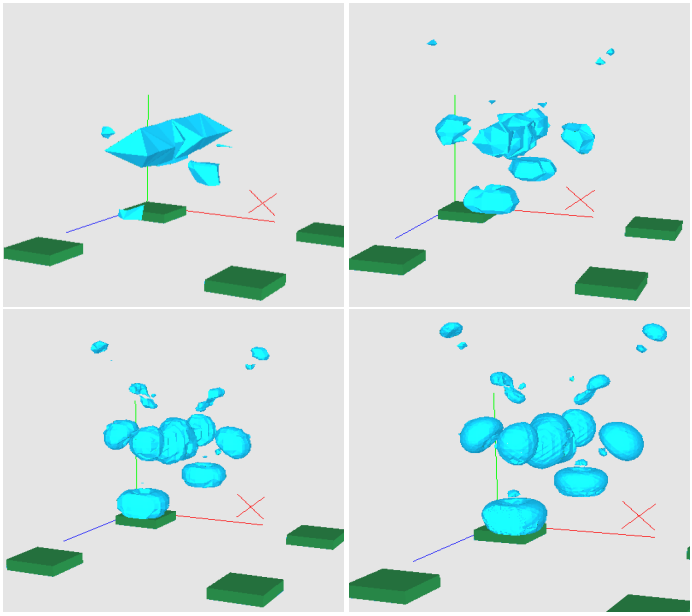

Figure 7: Steps in a progressive isosurface computation from the volumetric HIPIP dataset, left to right.