Dealing with Massive Volumetric Visualization: Progressive Algorithms and Data Structures

Rita Borgo

Visual Computing Group - Consiglio Nazionale delle Ricerche, via G. Moruzzi 1, I-56100 Pisa, Italy, phone: +39 050 315 3471, e_mail: borgoiei.pi.cnr.it

Valerio Pascucci

Center for Applied Scientific Computing Lawrence Livermore National Laboratory

Abstract

Massive data sets coming from complex simulations are "unwilling" to fit completely inside computer's internal memory. Consequently it is increasingly more difficult to visualize simulations results interactively. The use of smart External Memory (EM) algorithm becomes a must. To address these issues we propose an elegant and simple to implement framework for performing out-of-core visualization and view dependent refinement of large volume datasets. We adopt a method for view-dependent refinement that relies on a well known strategy (longest edge-bisection) yet introducing a new method for extending the technique to the field of Volume Visualization. We perform a top-down traversal of the mesh hierarchy to produce a continuous surface guaranteeing connectivity of the resulting subdivision and achieving good memory coherency in data accesses. We show how this framework extends the framework for terrain visualization of Pascucci and Lindstrom [**?**], to Volume Visualization still maintaining simplicity and memory/computing efficiency of the original technique.

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. As a consequence interactive visualization of result is going to become increasingly difficult, especially as a daily routine from a desktop. Such a problem poses fundamentally new challenges both to the development of visualization algorithms and to the design of visualization systems.

To address these issues new data-streaming techniques have been proposed mainly concerning progressive processing and visualization, and new high-technology is under development focusing more than before on commodity type of resources. Out-of-core computing [**?**] deals directly with algorithm adaptation and data layout to reduce performance degradation due to memory access in out-of-core processing. Results in this field are applicable in parallel and distributed computing ranging from cluster of PC's to more complex and expensive architectures.

In this paper we present a new progressive visualization algorithm where the input grid is traversed and 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. The hierarchy is built by one process that traverses the input 3D mesh. A second process performs the traversal and display. 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 that allows us to balance processing and data migration time. We use this approach with a new algorithm for storage layout that minimizes the amount of memory accesses necessary during a progressive traversal speeding up, as a consequence, the visualization process itself.

2 PREVIOUS WORK

In the course of the paper we will refer mainly at isocontour extraction and visualization in very large dataset. Unfortunately 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 Marching Cubes algorithm [**?**]; examples of accelerating techniques included the use of hierarchical data structures like octrees [**?**] and value decomposition methods [**?**; **?**]. A second class of isosurface rendering algorithms refers to techniques that resembles the contour propagation algorithms [**?**] of Pascucci et al.; these type of algorithms identify a seed cell from which to begin the propagation, they end up with a sort of seed set covering the isovalue range. The third class groups those algorithms that mainly focus on the reduction of the number of triangles generated during the isosurface extraction; belong to this class the algorithms proposed by Livnat and Hansen [**?**]. The approach we focus on is the one presented in [**?**] were Pascucci and Lindstrom presented a strong and performing algorithm for out-of-core and view-dependent refinement for large terrain surfaces. Their idea resulted to be suitable and performing for visualization even on small commodity devices like a normal laptop. In the following section we will show how to extend the same approach to 3D scalar field extraction and visualization.

3 SUBDIVISION SCHEME DESCRIPTION

This section describes in detail our progressive refinement algorithm and data-structures. The design of the algorithm is dependent on the kind of hierarchical data-structure assumed for the input spatial hierarchy. Our approach focus on the case of edge-bisection refinement, technique that is currently widely used in the meshing community [**?**; **?**; **?**]. 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

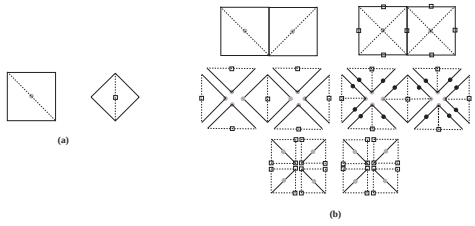

Figure 1: Sequence of 2D edge-bisection refinements and diamond construction for a regular grid. (a) Example of 2D diamond types; (b) 2D grid subdivision and diamonds generation. Diamond centers are labelled differently. Bisection edge are drawn with dashed patterns.

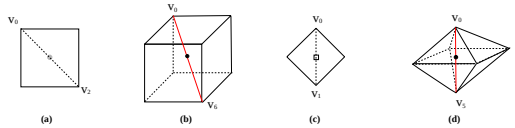

Figure 2: 2D type 0 and type 1 cells vs. 3D type 0 and type 1 cells. (a) shows a 2D cell of type 0; (b) shows the corresponding 3D type 0 cell ; (c) shows a 2D cell of type 1; (d) shows the corresponding 3D type 1 cell.

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.

For rectilinear volumetric input such a subdivision corresponds to an adaptive tree like sampling. We select a set of cells intersecting the isocontour value and organize them in a tree-like structure where each level corresponds to a different level of refinement of the isocountour and each node corresponds to an atomic cell.

In the next paragraph we analyze the 2D and 3D case of our subdivision technique.

3.1 2D Subdivision Scheme

We generate basically a partition, sub-sampling, of the input data, in figure 1 we show the 2D edge-bisection refinement sequence that we adopt to sub-sample a 2D regular grid. The coarse level basically is triangular mesh. Each refinement step inserts a new vertex on an edge and splits the triangles adjacent along such edge into two halves. Instead of "triangles" we reason in terms of "2D diamonds" considering each cell as the composition of the two triangular halves, that will be split by the bisection. For the 2D case we can subdivide the diamonds in two main classes (see figure 1a), first class diamond (or type 0 diamonds) square shaped, and second class diamond (o type 1 diamonds) rhombus shaped. Each diamond is characterized by a center (the center of the bisected edge).

The refinement scheme we adopt rely on properties that characterize the diamond center. First it is easy to see that each center is unique to the cell it belongs to (no two cell share the same center). Second, the center is defined by a tuple of indexes (i, j) $(i, j, k$ for the 3D case) from which it is possible to derive the class the cell belongs to (i.e. the diamond type), the subdivision level it has been generated from and, more evident for 3D cells, the orientation of the diamond (i.e. x, y oriented for the 2D case). In 2D such information is significant for the irregular grid case.

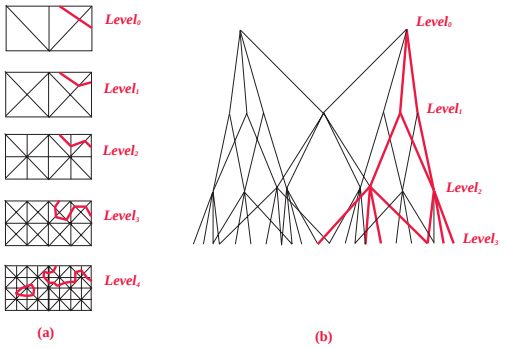

Figure 3: Grid bisection and hierarchy production. (a)Sequence of levels of resolution of an isocontour I (red bold line) of a 2D scalar field; (b) hierarchy representation of the levels sequence and of the cells containing the isovalue (red bold line).

The refinement can be applied *locally* to perform adaptive refinement or *globally* to increase uniformly the resolution of the mesh. The 2D mesh partitions subdivides the region of space of interest in triangles. Each vertex is associated with an input function value. Inside each triangular cell a linear function is used to interpolate the function values at the vertices. In this way we have a piecewise linear representation of the scalar field $F(x)$ necessary to compute an isocontour. As the edge-bisection algorithm makes progress new function values are added and a more detailed definition of the function $F(x)$ is obtained. In this way it is possible to map the refinement procedure directly into a refinement of the output isocontour, that is, instead of recomputing portions of the contour we augment its representation generating directly a progressive data-structure.

This progressive representation of the isocontour can be traversed adaptively independently from the underlying mesh. This refinement scheme has been successfully adopted by Pascucci and Lindstrom in [**?**] for terrain visualization.

3.2 2D Hierarchy and Adaptive Refinement

The subdivision scheme holds implicitly a hierarchical organization of the dataset itself. For the 2D case the hierarchy starts with a first class diamond cell and proceeds through each level of refinement with an alternation of second and first class diamond cell. The last possible level of refinement is always third class diamond cells. It is easy to organize such a hierarchy in a tree-like data structure (see figure 3). The traversal of the hierarchy allows to extract a sort of "seed set" [**?**] made up of cells whose internal range of isovalues includes the isovalue target. The seed set generated correspond to an adaptive traversal of the mesh at different levels. In our prototype the hierarchy is really built only for the cell belonging to the seed set, that is produced in a preprocessing stage, and traversed in depth to extract the isosurface itself.

3.3 3D Subdivision Scheme

Translating the refinement scheme in 3D introduces some challenges. Augmenting of 1 dimension implies as first consequence a growth in the number of diamond classes and possible diamond orientations. Orientations and classes augment of one, diamond cell can be of first class (i.e. cube shaped diamonds or type 0 diamonds), of second class (i.e. octahedral shaped diamonds or

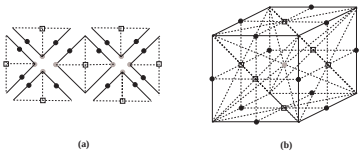

Figure 4: 2D Regular grid bisection point and corresponding scheme projected in 3D. (a)Location of bisection point for a 2D regular grid; (b) location of bisection point for a 3D regular grid.

type 1 diamonds) and of third class (i.e. esahedral diamonds or type 2 diamonds). It is possible to find a directed relation between first and second class 2D diamonds and first and second class 3D diamonds respectively (see figure 2).

In 3D the edge bisection refinement becomes a scheme for tetrahedral meshes subdivision. It applies to tetrahedral meshes in the same way it applies to 2D triangular meshes, with respect to the 2D case it maintains all the properties, especially the ones related to the center. Each cell is still subdivided along "the longest edge" that for a 3D cell correspond to the main diagonal of the cell itself. Each bisection gives birth to an arbitrary number of diamond-like new cells. For regular grids the starting cell is a first class diamond cell that results to be cube shaped (see figure 5), the longest edge correspond with one of the main diagonal. For convenience we number the diamond vertexes with fixed criteria and for first class diamond we always bisect along the diagonal limited by the vertex tuple (v_0, v_6) (i.e. vertexes v_0 and v_6 identify the "main" diagonal). The bisection gives birth to 6 halves of a first class diamond cell (figure 5b). Each of this six halves cell merged with the adjacent along its squared shaped face gives birth to 6 new second class diamond cell octahedral shaped (figure 5c). Each new second class diamond cell is bisected along its main diagonal and for a regular grid 4 new quarters of a third class diamond cell (for each of the previous 6) are created (see figure 6b). Each quarter merged with the 4 neighbors adjacent along its longest edge gives birth to a new third class diamond cell esahedral shaped (see figure 6c). Each new esahedral cell is bisected along its longest edge and a new bisection generates 16 tetrahedral shaped cell corresponding to one sixth of a first class diamond cell. In 3D we need to repeat the bisection procedure 3 times (instead of two as for the 2D case) to reach the final stage in which a new bisection will give birth to 16 new tetrahedral cells, see figure 7, (for each subdivided cell) that merged respectively with the 6 neighbors cells, adjacent within each other along their main edge, will generate a first class cube shaped cell (we need 6 tetrahedra to compose a cube shaped cell), returning to the subdivision step for the starting cell (in the 2D case we needed only two steps and two triangles to come out with a first class diamond cell square shaped).

Isocontour Extraction In 3D we have 3 basic "level of refinement" that amplify as the refinement proceeds. Each generated diamond internally holds a "piece" of the total isocontour that needs to be extracted. To perform the extraction we subdivide each diamond cell, belonging to the level of refinement required, into tetrahedra and apply a marching tetrahedra algorithm. Each isocontour is updated within a single tetrahedron and then composed to update the global isosurface within the set T of all tetrahedra around the bisection edge.

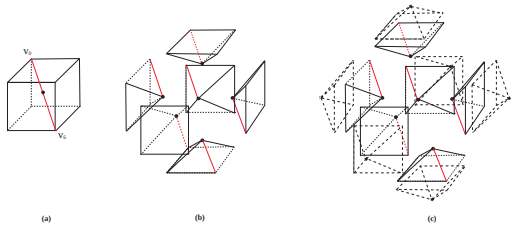

Figure 5: Sequence of 3D edge-bisection refinements for a regular grid. (a) Example of a first class diamond cell; (b) subdivision of the cell, it generates 6 new pieces of diamonds cell that (c) merged with neighbours will participate in the construction of 6 new first level diamond cells;

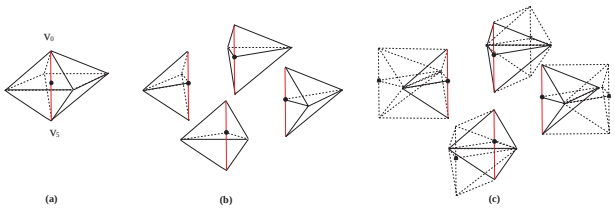

Figure 6: Sequence of 3D edge-bisection refinements for a regular grid. (a) Example of a second class diamond cell y oriented; (b) subdivision of the cell, it generates 4 new pieces of third class diamonds cell that (c) merged with neighbours will participate in the construction of 4 new second level diamond cells (2 x oriented and 2 z oriented).

3.4 3D Hierarchy and Adaptive Refinement

For the 3D case the hierarchy construction is similar to the one for the 2D case. The hierarchy still starts with a first class diamond cell and proceeds through each level of refinement now with an alternation of second, third and first class diamond cells. The last possible level of refinement is always third class diamond cells. The hierarchy is organized in a tree-like data structure. As for the 2D case we perform a traversal of the hierarchy to extract in a preprocessing step a "seed set". The seed set generated correspond to an adaptive traversal of the mesh at different levels. In our prototype the hierarchy is really built only for the cell belonging to the seed set, that is produced in a preprocessing stage, and traversed in depth to extract the isosurface itself.

4 Implementation Details

The first aim of our implementation has been to reduce as much as possible the amount of information needed to perform subdivision and isosurface extraction. The first interesting challenge has been represented by the mesh representation model in terms of diamonds. The overall mesh is in fact seen as a collection of geometric primitives (the diamonds) that for the regularity of the subdivision criteria allows for a very low memory footprint to be represented. Each cell is a unique and independent nucleus that stores in itself all the information needed. Cardinal point of a cell (i.e. of a diamond) is its center characterized by three index (i, j, k) from which it is possible to derive type, orientation and refinement level of the diamond cell. As a consequence of the subdivision scheme regularity to identify a diamond we need only to store its center coordinate (12 bytes). The regularity of the diamond shape allows in fact to gather also the diamond vertexes simply adding

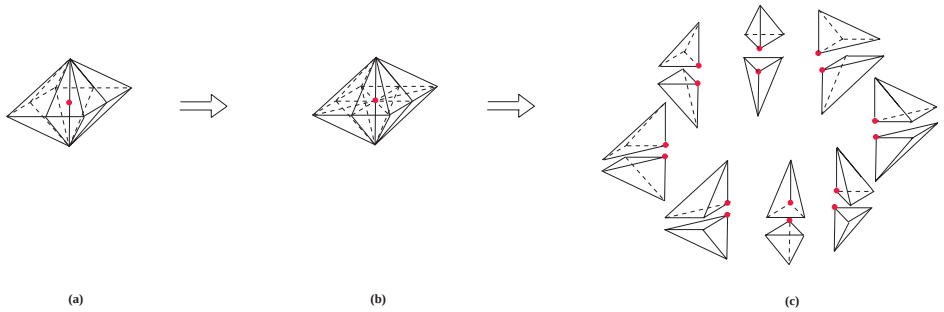

Figure 7: Sequence of 3D edge-bisection refinements for a regular grid. (a) Example of a third class y oriented diamond cell; (b) subdivision of the diamond cell along the longest edge (c) partition of the diamond cell, it generates 16 new pieces of diamonds cell that merged with neighbours will participate in the construction of 8 new first level diamond cell.

a δ constant to the center coordinates. The constant is fixed for each type of diamond and dependent in magnitude to the level of refinement (as afore head mentioned easily derivable from the coordinates of the center) reached.

For the fundamental role played by the center in our subdivision scheme we have decided to organize most of the data related information in tables filled in preprocessing steps and to use the center itself as key to address the tables. These tables are mainly useful for the seed set construction, they introduce an overhead, in terms of computational time and memory occupancy, counter balanced by the number of computation steps that it allows us to save during the isourface extraction.

Another choice worth to be mentioned regards the features of the "nodes" that compose the hierarchy tree. In the hierarchy we memorize the center of each diamond and a list that allows us to keep track of the vertexes of the piece of isosurface contained in the diamond. This extra information is needed to pass shared isosurface vertexes from father to son. With a small memory overhead of at most 34 int for a third class diamond (18 int for a second class diamond, 26 for a first class diamond) we avoid to recalculate for each diamond all the isosurface vertexes it contains limiting the computation to those vertex not in common or inherited from the fathers reducing the operation of interpolation of a factor of 3. A first class diamond needs to calculate only 8 new isovertexes (instead of 26), a second class diamond needs to calculate only 6 new isovertexes (instead of 18) and a third class diamond needs to calculate only 10 new isovertexes (instead of 34). The passing of inherited vertexes from father to son can be done with three operations each of unitary cost ($O(1)$). The order in which the vertexes should be inherited is known for construction, we always know which son inherits which vertex (see figure 8).

In our implementation we uncouple the isosurface extraction from its display. The isocontour hierarchy is built by one process that traverses the input 3D mesh; a second process performs the isocontour display. Each level of refinement (corresponding 1 to 1 to a level of the hierarchy tree) can be sent independently to the second process that will perform the rendering phase.

For a more detailed description of implementation issues and decisions we suggest to refer to [**?**].

Computational Behaviour 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

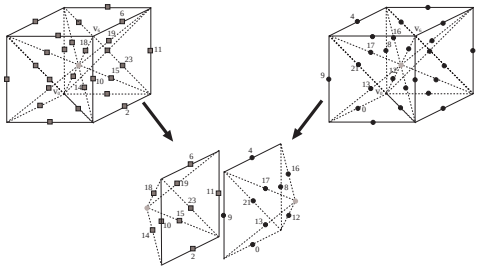

Figure 8: Isovertex inheritance for a second class diamond from its two first class diamond fathers. The diamond inherits exactly 12 vertexes, 8 from each father but 4 in common on the shared face.

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 $O(n)$.

5 DISCUSSION AND FUTURE WORK

In this paper we propose a new type of trade-off between speed and accuracy in view dependent isosurface extraction. First by traversing an optimized tree-like structure in a coarse to fine order can collect active cells in the same order. Second with the help of a progressive visualization technique active cells can be rendered at interactive frame rate. This allows us to render at any given time partial results while the computation of the complete hierarchy makes progress. The peculiarity of our technique is to be adaptive and at the same time to maintain a consistent hierarchy for the output isocontour. In spite of a minimal memory overhead we can guarantee optimal time performance in isosurface construction and extraction. The key novelty of the present scheme is that providing a set of local rules for continuous geometric transitions (geomorphs) of one level of resolution into the next we bridge the gap between adaptive techniques and multi-resolution decimation-based techniques. Moreover the regularity of the scheme allows 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.