Eurographics Symposium on Parallel Graphics and Visualization (2004), pp. 1–7
Dirk Bartz, Bruno Raffin and Han-Wei Shen (Editors)

Massive Data Pre-Processing with a Cluster Based Approach

R. Borgo[1†] P. Cignoni[1‡] V. Pascucci[2§] and R. Scopigno[1¶]

[1] Visual Computing Lab., CNR, Pisa [2] Lawrence Livermore National Lab.

Abstract

Data coming from complex simulation models reach easily dimensions much greater than available computational resources. Visualization of such data still represents the most intuitive and effective tool for scientific inspection of simulated phenomena. To ease this process several techniques have been adopted mainly concerning the use of hierarchical multi-resolution representations. In this paper we present the implementation of a hierarchical indexing schema for multiresolution data tailored to overwork the computational power of distributed environments.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Parallel Volume Visualization

1. Introduction

Data coming from complex simulation models reach easily dimensions much grater than available computational resources. Visualization of such data still represents the most intuitive and effective tool for scientific inspection of simulated phenomena. To ease this process several techniques have been adopted mainly concerning the use of hierarchical multi-resolution representations. A key component of these schemes has become the adaptive traversal of hierarchical data-structures to build, in real time, approximate representations of the input geometry to speed up the rendering stage. To guarantee consistent outputs a pre-processing of at least part of the input data is required. Due to size boundaries smart distributed processing, able to overwork inner properties of a hierarchical organization of the data, represents a powerful source of computation capability. In this paper we present the implementation of a powerful subdivision schema tailored to overwork the computational power of distributed environments. Three features make the scheme particularly attractive: (i) the data layout requires as low memory as the data themselves, (ii) the computation of the index for rectilinear grids is implemented with simple bit address manipulations and (iii) there is no replication of data neither during hierarchy construction nor during data preprocessing, avoiding performance penalties for dynamically modified data. This paper introduces a new global indexing scheme that accelerates adaptive traversal of geometric data represented with regular grids by improving the locality of hierarchical/spatial data access. The effectiveness of the approach was tested with the fundamental visualization technique of rendering regular 3D volume datasets through isosurface extraction.

† borgo@isti.cnr.it
‡ cignoni@isti.cnr.it
§ pascucci@llnl.gov
¶ scopigno@isti.cnr.it

2. Previous Work

The panorama of distributed graphics presents a wide range of interesting approaches developed to speed up processing and rendering of large volume data. A common requirement of such schemes has been the construction of hierarchical data structures before performing any visualization [5, 11, 15]. New algorithmic techniques and analysis tools have been developed to address the problem of memory layout in the case of geometric algorithms and scientific visualization [3]. Closely related issues emerge in the area of parallel and distributed computing where remote data transfer can become a primary bottleneck during computation. In this context space filling curves are often used as a tool to determine, very quickly, data distribution layouts that guarantee good geometric locality [6, 7, 8]. In the approach proposed here a new data layout is used to allow efficient processing and access to volumetric information. This is achieved by combining a low memory refinement strategy while maintaining geometric proximity of the generated

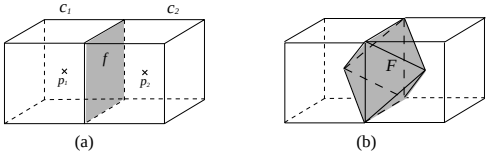

Figure 1: *3D cell refinement from tier 0 to tier 1. (a) The two cells c_1 and c_2 in tier 0 . Their centers p_1 and p_2 are marked with two crosses. Their adjacency facet f is highlighted in gray. (b) The cell F of tier 1 (in gray) is the union of the pyramids $p_1 \lhd f$ and $p_2 \lhd f$.*

level of resolution. One main advantage is that the resulting data layout can be used with different blocking factors making it beneficial throughout the entire memory hierarchy.

The approach we implemented comprehend two main phases:

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

In the present context the *pre-processing* phase comprehends volume subdivision for extraction of all the data and their organization in tables. The *rendering phase* consists instead of mesh traversal and isocontour extraction following appropriate approximation criteria. Input of the framework is a regular volumetric dataset extended when needed to even dimension ($(2^N+1)\times(2^N+1)\times(2^N+1)$). In this paper we face the parallelization of only the first phase (pre-processing) that currently represents the most expensive step, in terms of computational time and memory occupation, of the framework we built [2]. For clarity in section 3 we present the subdivision rule, at the base of our framework, while the rest of the paper will concentrate mainly on issues and properties characteristics of the application we are interested in.

3. Subdivision Technique

The purpose of this section is mainly to introduce the subdivision scheme adopted to build the refinement hierarchy. The theory at the base of the refinement scheme adopted has been introduced by Pascucci [11] for arbitrarily dimensional volume meshes. We propose the refinement scheme from a 3D point of view based on a set of simple rules that characterize consistently the decomposition of a regular 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 [2].

3.1. 3D Subdivision Scheme

As proposed in [11] 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 $l+1$. Each refinement consists of 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.

3.2. Subdivision Rules

In the following paragraphs we will analyze each refinement step in details.

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

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

Figure 1 shows the construction of F from c_1 and c_2.

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

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

After each merge step all the spurious edges introduced during the refinement procedure are removed.

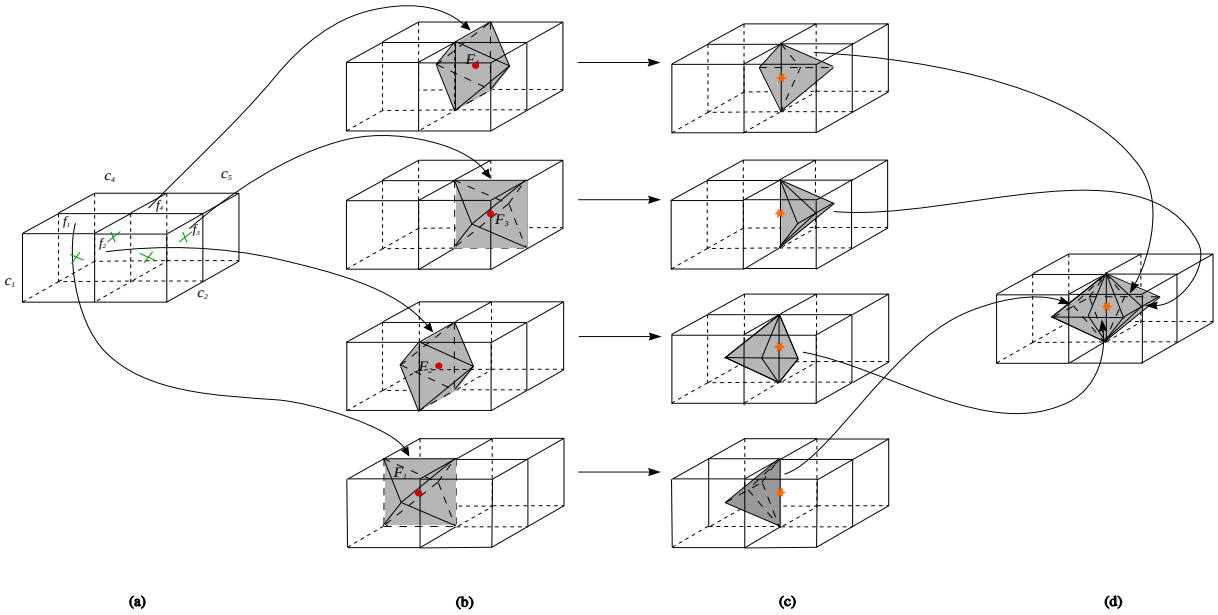

Figure 2: *Cell refinement from tier 1 to tier 2. (a) Four cells c_1, c_2, c_3 and c_4 of tier 0 share, in pairs, the facets f_1, f_2, f_3 and f_4. The edge e is shared by all facets f_1, f_2, f_3 and f_4. (b) Each facet f_i generates a cell F_i. (c) Each cell F_i is decomposed into four pyramids only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

4. Indexing The Schema

The first interesting aspect of this subdivision schema is that, given a mesh representation model, it can be organized hierarchically in terms of embedded entities that for shape reason we freely define as *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. The overall mesh is 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: diamonds as entities do not really exists, only their centers exists and only 3 shorts are needed to represent satisfactorily a center. From a center, characterized by three index (i, j, k), it is possible, with just a couple of unitary operations, to derive tier, type, orientation and refinement level the diamond belongs to. The regularity of the diamond shape allows to gather the diamond vertexes position simply adding a δ constant to the center coordinates. In case of regular grids the constant is fixed for each type of diamond and dependent in magnitude to the level of refinement reached (as said before easily derivable from the coordinates of the center of the diamond cell). Through simple mathematical rules it is possible to identify its sons (diamond vertexes are needed only for sons generation, that is to "proceed" in the refinement). Every point of the mesh can be reached following the 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. Our initial efforts have been focused on the refinement of regular grid nevertheless the framework has been designed to be independent of the kind of input mesh.

4.1. Pre-Processing Data Management

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, error. Each table has dimension equal to the dimension of the volume $((2^N+1)\times(2^N+1)\times(2^N+1))$, and access key equal to a function of the (i, j, k) indexes of each diamond center. Filling of data and range tables can be done during 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. For large dataset the tables can reach prohibitive dimensions for this reason in our cluster implementation we have substituted the actual tables with local files (see 5.3). 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 (see Sec. 4.1.1) we adopt assures an overestimation of the error introduced by the approximation but requires to be able to move eas-

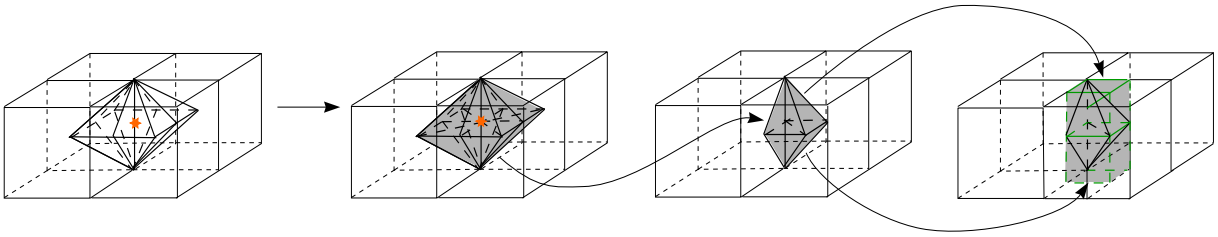

Figure 3: *Cell refinement from tier 2 to tier 3.*

ily 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.

4.1.1. 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 from bottom to top in the pre-processing phase. For each traversed diamond D we compute its field F_D and, for each grid vertex v contained in D, we compute the error introduced by approximating the field value of v with $F_D(v)$. Defined with V_G the total number of vertexes of a $2^N \times 2^\times 2^N$ grid G, an upper bound to the computational complexity of the field space error for the entire grid G is given by the formula: $V_G * \log_4(V_G)$.

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 therefore it is easier to compute. 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 :

$$\lambda = \frac{w}{2\tan\frac{\varphi}{2}} \tag{2}$$

where w is the number of pixels along the field of view φ.

Equation 1 corresponds 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 in [10]. 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.

To guarantee error nesting the error of a diamond is always computed as 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.

The heaviest part in terms of computational time is represented by the computation of field error that requires the traversal of the entire dataset (while the screen space error is equivalent to multiply by a "fixed" constant the field error). On this aspect we have focused part of our efforts during cluster oriented implementation of the pre-processing phase.

5. Cluster Implementation

In inspecting our volumetric dataset the heaviest computational part is represented by the pre-processing phase in which per cell min/max and approximation error values are computed. This three factors are necessary to be able to perform adaptive traversal of the hierarchy generated by the refinement schema. Moreover the min/max and error computation imply the traversal of the entire dataset (while at runtime only part of the data need to be inspected) that, for large size dataset, implies a significant cost in terms of computational time. Table filling is one of the heaviest operations we

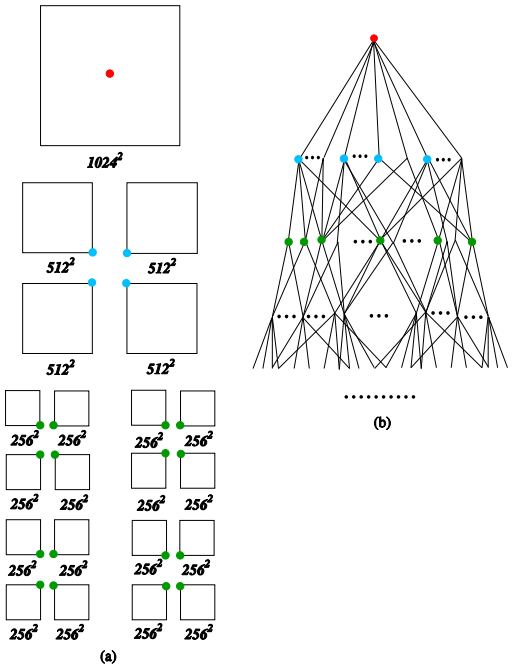

Figure 4: *Data partitioning scheme of a 2D datasets of* 1024^2*:(a) Partitioning of the datasets in 16 blocks of* 256^2*, each block can be sent to a different processor, colored vertexes are the only vertexes for which the error is not computed, colored vertexes can be considered like* root *of the block they belong to; (b) corresponding position in DT of each* block-root *node, they all belong to the first three level of DT.*

perform, part of the memory is occupied 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. The diamond tree is actually needed because of the error metric adopted (Sec. 4.1.1), that requires to easily move from bottom levels to top levels of the hierarchy. Keeping the hierarchy representation implicit to each diamond we can guarantee: an easy top-down traversal of the hierarchy, not an equally easy traversal bottom-up. Subdivision of the data allows us to constraint this operations on small parts speeding up at almost real-time rates their processing (see Sec. 6).

5.1. Building the Data

The subdivision schema adopted allows us to partition the entire dataset G into equal consecutive bricks $B_1 \dots B_m$ of arbitrary size, each one corresponding to a grid of $2^b \times 2^b \times 2^b$ vertices. Each grid by itself corresponds to part of the dataset in terms of an embedded subtree of implicit topology, and correspond directly to part of the tables in which we collect computed values (like error and ranges). It corresponds also to a subtree, independent from the others, opened to be sent to a different node to be evaluated. The data-partitioning scheme distributes evenly each block, leaving out from the error calculation only the block roots that by themselves correspond to the nodes that make up the first levels of the hierarchy, levels that correspond to very coarse data for which an arbitrary big error can be assumed. An example for the partition of a 2D dataset of dimensions 1024×1024 is given in Fig. 4.

5.2. Sending the Data

For regularity the absolute position of each block, with respect to the global grid G, can be easily derived in function of the block index b_i and its size. The order in which we subdivide a 3D grid is given in Fig. 5. Given a grid of dimension $2^N \times 2^N \times 2^N$ subdivided into K blocks each of dimension $2^{N/K} \times 2^{N/K} \times 2^{N/K}$ the block of index b_i will correspond to the grid element containing the grid vertexes V_g of indexes (i,j,k) with i, j and k included in the ranges:

$$low\_x\_dim(b_i) \leq i \leq high\_x\_dim(b_i)$$

$$low\_y\_dim(b_i) \leq j \leq high\_y\_dim(b_i)$$

$$low\_z\_dim(b_i) \leq k \leq high\_z\_dim(b_i)$$

Indicating with *rem* and *quot* respectively the remainder and quotient of a division, the boundaries of each range can be determined as:

$$low\_x\_dim[b_i] = (rem(b_i/K*K)/K)*2^{N/K}$$
$$high\_x\_dim[b_i] = low\_x\_dim[b_i] + 2^{N/K};$$

and

$$low\_y\_dim[b_i] = quot(b_i/K*K)*K$$
$$high\_y\_dim[b_i] = low\_y\_dim[b_i] + 2^{N/K};$$

$$low\_z\_dim[b_i] = (quot(rem(b_i/K*K)/K)*K)$$
$$high\_z\_dim[b_i] = low\_z\_dim[b_i] + 2^{N/K};$$

As a consequence the messages that we need to send to each node have dimension equal to:

$$size\_of(B_i) + tot\_vertexes\_in(b_i)$$

that in numerical terms corresponds to 1 char plus $2^{N/K} \times 2^{N/K} \times 2^{N/K}$ short for a total of $(1 + 2^{N/K} * 2^{N/K} * 2^{N/K} * 3)$ bytes (i.e the data themselves plus just one extra byte).

Dataset	Size	Data Partition	Range Computation
Visible Human	512^3	10.6 secs	46.08 secs
Drip	1024^3	78.16 secs	6.15 min.

Table 1: *Computational time required for the partition of the data (i.e. blocks construction) and the computation of min/max ranges. Performances computed over the Visible Human Male Chest dataset* (512^3) *and a synthetically generated dataset Drip* (1024^3).

5.3. Gathering the Results

In our implementation we have designated, in a range of eight node (d01 to d08), node d01 as the node in charge of receiving all the results coming from each of the siblings nodes. As said before given a block index b_i it is straightforward to determine its grid positions in terms of absolute values. Incoming results have the same shape of the sent data, and size three times greater because comprehending computed min, max and error values. Results are stored on local files, of dimension equal to the dimension of one block, and no more in Tables (as stated in section 4.1) and picked up as needed at rendering time.

5.4. Load Balancing

We take a load balancing approach based on static data distribution [9, 13, 14] where there is no data replication and no contention on a centralized agent that distributes tasks. Data are partitioned evenly and each block has dimension corresponding to a power of two.

Data to be sent to each node are limited to the index of the block sent and of the data themselves in terms of field values.

6. Implementation Results

We have been doing our experiments on a Linux Cluster of 8 nodes (d01 to d08)[†]. Node d01 has been designed as collector of resulting data coming from siblings node. Table1 shows the timing for the computation of the partition of the data, that is data loading and blocks construction, and of the min/max range for a $(512 \times 512 \times 512)$ and $(1024 \times 1024 \times 1024)$ datasets subdivided in blocks of dimensions $(32 \times 32 \times 32)$. Data are organized as consecutive memory mapped blocks constructed during data loading. The 1024^3 dataset has been synthetically generated through the $F(x,y,z) = x^2 + y^2 - 0.5(0.995z^2 + 0.005 - z^3)$ (courtesy of Terry J. Ligocki, Lawrence Berkeley National Laboratory). The function has been evaluated for $x,y,z \in [-1.5, 1.5]$ and normalized to $[0, 2^N]$. The adoption of function F is due to the difficulty in obtaining 1024^3 volume datasets.

[†] Courtesy of dott. Raffaele Perego, HLAB, Italian National Research Council, and Giancarlo Bartoli, CNUCE, Italian National Research Council

6.1. Conclusions

In this paper we have introduced a subdivision schema for level of refinement construction well suited for the design of an efficient run-time data partitioning and distribution algorithm. We have presented a primary implementation of such algorithm able to reduce the local memory requirement and overwork distributed environment potentiality. Because of the regularity of the grid and because the distributed implementation regards only the pre-processing phase, that is a phase where the all dataset needs to be traversed, load balancing does not really constitute an issue, each node computes the same amount of data. In terms of memory requirements and computational time the scheme seems to be pretty promising for this reason our next step is to try to optimize also the rendering phase of our approach overworking the computational resources of distributed environments.

References

[1] Dirk Bartz, Wolfgang Strasser, Roberto Grosso, and Thomas Ertl, *Parallel construction and isosurface extraction of recursive tree structures*, WSCG'98 Conference Proceedings of Recursive Tree Structures (V. Skala, ed.), 1998.

[2] R. Borgo, V. Pacsucci, P.Cignoni, and R. Scopigno, *A progressive subdivision paradigm (psp)*, Proceedings of the 2004 Visualization and Data Analysis Conference, IST/SPIE's 16th Annula Symposium (Stephan N. Spencer, ed.), IST/SPIE, January 12004.

[3] Yi-Jen Chiang and Cláudio T. Silva, *I/O optimal isosurface extraction*, IEEE Visualization '97 (Roni Yagel and Hans Hagen, eds.), IEEE, 1997, pp. 293–300.

[4] P. Cignoni, D. Costanza, C. Montani, C. Rocchini, and R. Scopigno, *Simplification of tetrahedral meshes with accurate error evaluation*, IEEE Visualization '00 (VIS '00) (Washington - Brussels - Tokyo), IEEE, October 2000, pp. 85–92.

[5] Benjamin Gregorski, Mark Duchaineau, Peter Lindstrom, Valerio Pascucci, and Kenneth I. Joy, *Interactive view-dependent rendering of large IsoSurfaces*, Proceedings of the 13th IEEE Visualization 2002 Conference (VIS-02) (Piscataway, NJ) (Robert Moorhead, Markus Gross, and Kenneth I. Joy, eds.), IEEE Computer Society, October 27– November 1 2002, pp. 475–484.

[6] M. Griebel and G. W. Zumbusch, *Parallel multigrid in an adaptive PDE solver based on hashing*, Proceedings of ParCo '97 (E. D'Hollander, G. R. Joubert, F. J. Peters, and U. Trottenberg, eds.), Elsevier, 1997, pp. 589–599.

[7] R. Niedermeier, K. Reinhardt, and P. Sanders, *Towards optimal locality in mesh-indexings*, Lecture Notes in Computer Science **1279** (1997).

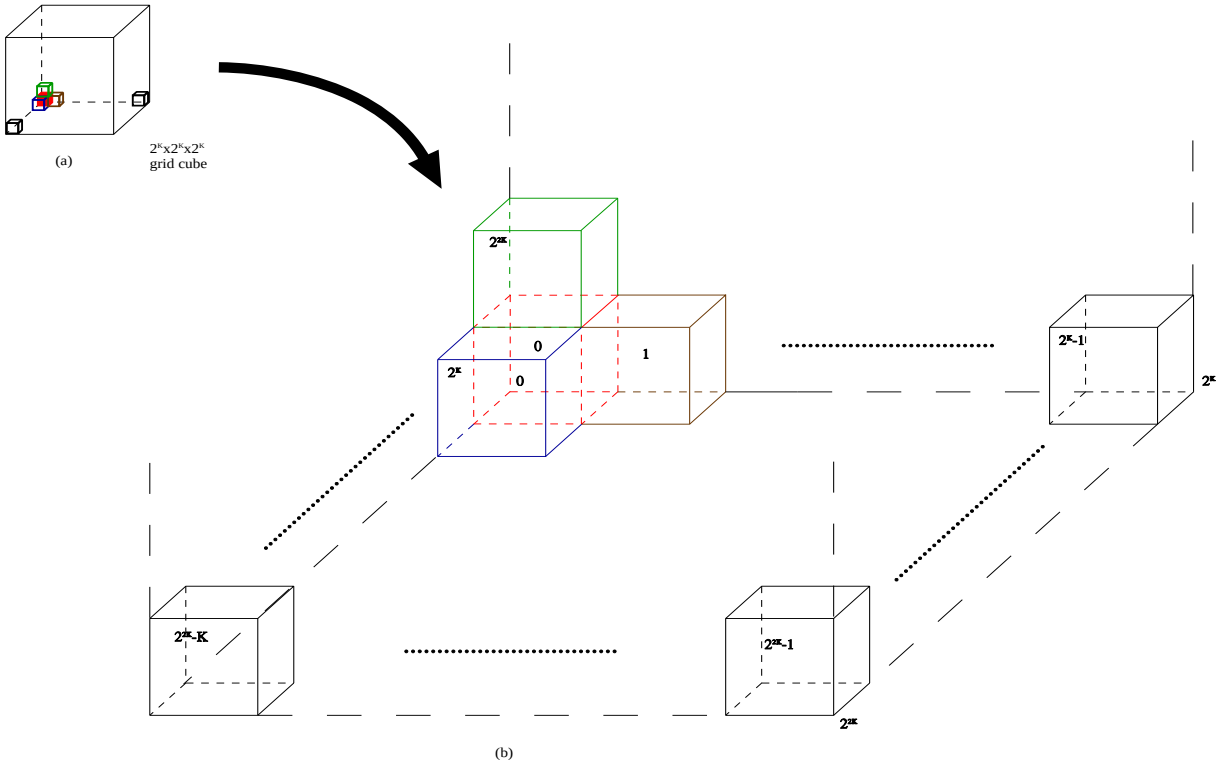

Figure 5: *Block enumeration in a* $2^K \times 2^K \times 2^K$ *regular grid.*

[8] M. Parashar, J. C. Browne, C. Edwards, and K. Klimkowski, *A common data management infrastructure for adaptive algorithms for PDE solutions*, Proceedings of Supercomputing'97 (CD-ROM) (San Jose, CA), ACM SIGARCH and IEEE, November 1997, University of Texas, Austin.

[9] V. Pascucci, C. L. Bajaj, D. Thompson, and X. Y. Zhang, *Parallel accelerated isocontouring for out-of-core visualization*, Proceedings of the 1999 IEEE Parallel Visualization and Graphics Symposium (PVGS'99) (N.Y.) (Stephan N. Spencer, ed.), ACM Siggraph, October 25–26 1999, pp. 97–104.

[10] 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.

[11] 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.

[12] F. Vivodtzev, L. Linsen, G.-P. Bonneau, B. Hamann, K. I. Joy, and B. A. Olshausen, *Hierarchical isosurface segmentation based on discrete curvature*, Proceedings of the symposium on Data visualisation 2003 (C. D. Hansen G.-P. Bonneau, S. Hahmann, ed.), Eurographics Association, 2003, pp. 249–258.

[13] X. Zhang, C. Bajaj, and W. Balnke, *Scalable isosurface visualization of massive datasets on cots clusters*, IEEE Visualization '01 (VIS '01), IEEE, October 2001, pp. 51–58.

[14] Xiaoyu Zhang, Chandrajit Bajaj, and Vijaya Ramachandran, *Parallel and out-of-core view-dependent isocontour visualization using random data distribution*, 2002.

[15] 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.