Preprint
UCRL-JC-140581

Multi-Resolution Indexing for Hierarchical Out-of-Core Traversal of Rectilinear Grids

V. Pascucci

This article was submitted to
National Science Foundation/DOE Lake Tahoe Workshop on Hierarchical Approximation and Geometrical Methods for Scientific Visualization
Tahoe City, CA
October 15-17, 2000

U.S. Department of Energy

Lawrence
Livermore
National
Laboratory

July 10, 2000

Center for Applied Scientific Computing - Lawrence Livermore National Laboratory

Multi-resolution Indexing for Hierarchical Out-of-core Traversal of Rectilinear Grids

Valerio Pascucci

1 Introduction

The real time processing of very large volumetric meshes introduces specific algorithmic challenges due to the impossibility of fitting the input data in the main memory of a computer. The basic assumption (RAM computational model) of uniform-constant-time access to each memory location is not valid because part of the data is stored out-of-core or in external memory. The performance of most algorithms does not scale well in the transition from the in-core to the out-of-core processing conditions. The performance degradation is due to the high frequency of I/O operations that may start dominating the overall running time.

Out-of-core computing [28] addresses specifically the issues of algorithm redesign and data layout restructuring to enable data access patterns with minimal performance degradation in out-of-core processing. Results in this area are also valuable in parallel and distributed computing where one has to deal with the similar issue of balancing processing time with data migration time.

The solution of the out-of-core processing problem is typically divided into two parts:

(i) analysis of a specific algorithm to understand its data access patterns and, when possible, redesign the algorithm to maximize their locality;

(ii) storage of the data in secondary memory with a layout consistent with the access patterns of the algorithm to amortize the cost of each I/O operation over several memory access operations.

In the case of a hierarchical visualization algorithms for volumetric data the 3D input hierarchy is traversed to build derived geometric models with adaptive levels of detail. The shape of the output models is then modified dynamically with incremental updates of their level of detail. The parameters that govern this continuous modification of the output geometry are dependent on the runtime user interaction making it impossible to determine a priori what levels of detail are going to be constructed. For example they can be dependent from external parameters like the viewpoint of the current display window or from internal parameters like the isovalue of an isocontour or the position of an orthogonal slice. The structure of the access pattern can be summarized into two main points: (i) the input hierarchy is traversed level by level so that the data in the same level of resolution or in adjacent levels is traversed at the same time and (ii) within each level of resolution the data is mostly traversed at the same time in regions that are geometrically close.

In this paper I introduce a new static indexing scheme that induces a data layout satisfying both requirements (i) and (ii) for the hierarchical traversal of n-dimensional regular grids. In one particular implementation the scheme exploits in a new way the recursive construction of the Z-order space filling curve. The standard indexing that maps the input nD data onto a 1D sequence for the Z-order curve is based on a simple bit interleaving operation that merges the n input indices into one index n times longer. This helps in grouping the data for geometric proximity but only for a specific level of detail. In this paper I show how this indexing can be transformed into an alternative index that allows to group the data per level of resolution first and then the data within each level per geometric proximity. This yields a data layout that is appropriate for hierarchical out-of-core processing of large grids.

The scheme has three key features that make it particularly attractive. First the order of the data is independent of the out-of-core blocking factor so that its use in different settings (e.g. local disk access or transmission through a network) does not require large data reorganization. Secondly the conversion from the standard Z-order indexing to the new index can be implemented with a simple sequence of shift operations making it appealing for a possible hardware implementation. Third there is no data replication which is especially desirable when the data is accessed through slow connections and avoids eventual problems of data consistency.

Beyond the theoretical interest in developing hierarchical indexing schemes for n-dimensional space filling curves the approach is currently targeted for its practical use in out-of-core visualization algorithms. Experimental results and theoretical analysis are reported in this paper for the simple case of orthogonal slicing of volumetric data. The results show how the practical performance enhancement corresponds to the theoretical expectations. The scheme is also targeted to perform out-of-core progressive computation of general slices, for its use in combination with the 3D progressive isocontouring algorithm [24] and for out-of-core visualization of large terrains using edge bisection hierarchies [11, 19].

The remainder of this paper is organized as follows. Section 2 discusses briefly previous work in related areas. Section 3 introduces the general framework for the computation of the new indexing scheme. Section 4 discusses the basic implementation of the approach for binary tree hierarchy. Section 5 presents the extension to higher order hierarchies. Sections 7 and 8 discuss possible extensions of the scheme and concluding remarks.

2 Related Previous Work

External memory algorithms [28], also known as out-of-core algorithms, have been rising in recent years to the attention of the computer science community since they address systematically the problem of non uniform memory structure of modern computers (fast cache, main memory, hard disk, ...). This issue is particularly important when dealing with large data-structures that do not fit in the main memory of a single computer since the access time to each memory unit is dependent on its location. New algorithmic techniques and analysis tools have been developed to address this problem for example in the case of geometric algorithms [20, 2, 13, 1] or scientific visualization [10, 4]. Closely related issues emerge in the area of parallel and distributed computing where remote data transfer can become a primary bottleneck in the 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 [23, 14, 21]. Space filling curves [26] have been also used in the past in a wide variety of applications [3] both because of their hierarchical fractal structure as well as for their well known spatial locality properties. The most popular is the Hilbert curve [15] which guarantees the best geometric locality properties [22]. The pseudo-Hilbert scanning order [8, 7, 16] generalizes the scheme to rectilinear grids that have different number of samples along each coordinate axis.

Recently Lawder [17, 18] explored the use of different kinds of

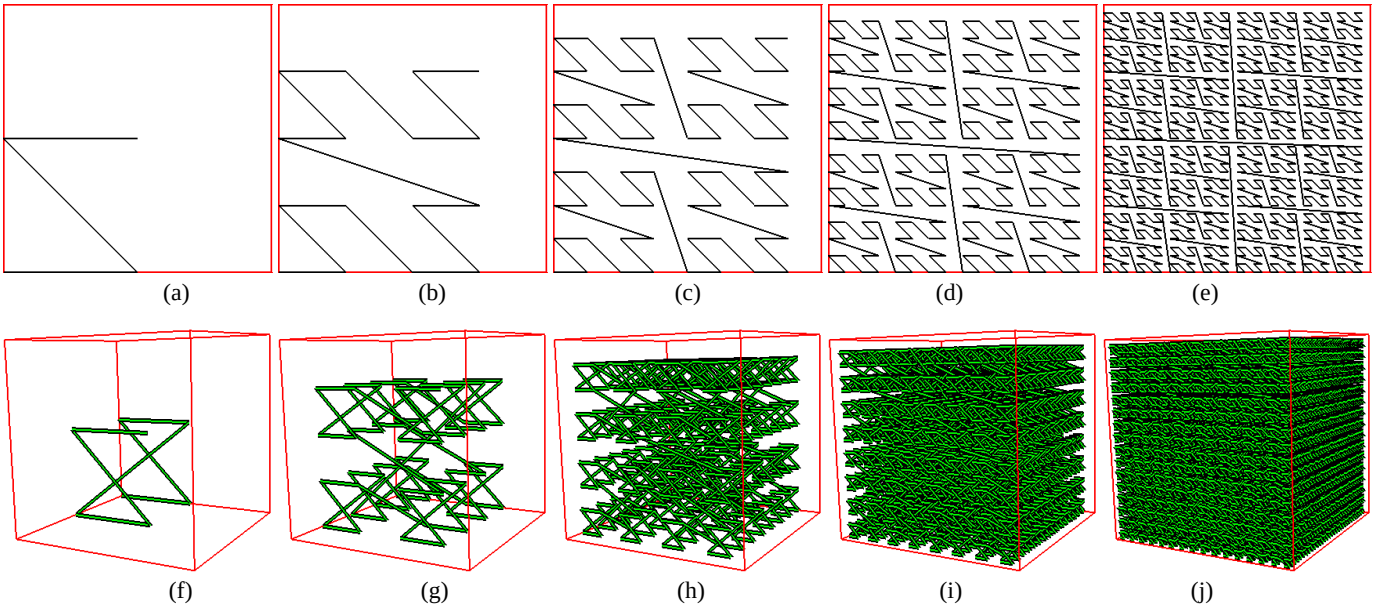

Figure 1: (a-e) The first five levels of resolution of the 2D Lebesgue's space filling curve. (f-j) The first five levels of resolution of the 3D Lebesgue's space filling curve.

space filling curves to develop indexing schemes for data storage layout and fast retrieval in multi-dimensional databases.

Balmelli at al. [5, 6] use the Z-order space filling curve to navigate efficiently a quad-tree data-structure without using pointers. They use simple closed formulas for computing neighboring relations and nearest common ancestors between nodes to allow fast generation of adaptive edge-bisection triangulations. They improve on the basic data-structure already used for terrain visualization [11, 19] or adaptive mesh refinement [25]. The use of the Z-order space filling curve for traversal of quadtrees [27] (also called Morton-order) and has been also proven useful in the speedup of matrix operations allowing to make better use of the memory cache hierarchies [9, 29, 12].

In the approach proposed here a new data layout is used to allow efficient progressive access to volumetric information stored in external memory. This is achieved by combining interleaved storage of the levels in the data hierarchy while maintaining geometric proximity within each level of resolution. One main advantage is that the resulting data layout is independent of the particular adaptive traversal of the data. This improves fundamentally from the previous schemes since they used the space filling curves only for computation and dynamic relocation of data layouts for single resolution or fixed adaptive resolution meshes.

3 The General Framework

Consider a set S of n elements decomposed into a hierarchy $\mathcal{H}$ of k levels of resolution $\mathcal{H} = \{S_0, S_1, \ldots, S_{k-1}\}$ such that:

$$S_0 \subset S_1 \subset \cdots \subset S_{k-1} = S$$

where S_i is said to be coarser than S_j iff $i < j$. The order of the elements in S is defined by the cardinality function $I : S \rightarrow \{0 \ldots n-1\}$. This means that the following identity always holds:

$$S[I(s)] \equiv s$$

where the square brackets are used to index an element in a set.

Let's define a derived sequence $\mathcal{H}'$ of sets S'_i as follow:

$$S'_i = S_i \backslash S_{i-1} \qquad i = 0, \ldots, k-1$$

where formally $S_{-1} = \emptyset$. The sequence $\mathcal{H}' = \{S'_0, S'_1, \ldots, S'_{k-1}\}$ is a partitioning of S. A derived cardinality function $I' : S \rightarrow \{0 \ldots n-1\}$ can be defined on the basis of the following two properties:

- $\forall s, t \in S'_i : I'(s) < I'(t) \Leftrightarrow I(s) < I(t)$;
- $\forall s \in S'_i, \forall t \in S'_j : i < j \Rightarrow I'(s) < I'(t)$.

If the original function I has strong locality properties when restricted to any level of resolution S_i then the cardinality function I' generates the desired global index for hierarchical and out-of-core traversal.

The construction of the function can be achieved in the following way: (i) determine the number of elements in each derived set S'_i and (ii) determine a cardinality function $I''_i = I'|_{S'_i}$ restriction of I' to each set S'_i. In particular if c_i is the number of elements of S'_i one can predetermine the starting index of the elements in a given level of resolution by building the sequence of constants $C_0, \ldots, C_{k-1}$ with

$$C_i = \sum_{j=0}^{i-1} c_j. \tag{1}$$

Secondly one needs to determine a set of local cardinality functions $I''_i : S'_i \rightarrow \{0 \ldots c_i - 1\}$ so that:

$$\forall s \in S'_i : I'(s) = C_i + I''_i(s). \tag{2}$$

The computation of the constants C_i can be performed in a preprocessing stage so that the computation of I' is reduced to the following two steps:

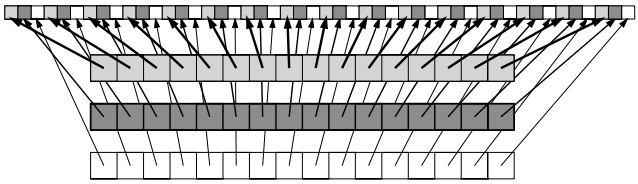

Figure 2: Construction of the 1D index from the Lebesgue's Z-order space filling curve. In the 3D case the original index is a set of three bit-strings. The 1D index is formed by interleaving the bit of the three sequences into a single bit-string.

- given s determine its level of resolution i (that is the i such that $s \in S'_i$);
- compute $I''_i(s)$ and add it to C_i.

These two steps need to be performed very efficiently because they are going to be executed repeatedly at run time. The following sections report practical realizations of this scheme for rectilinear cube grids in any dimension.

4 Binary Tree And the Lebesgue Space Filling Curve

This section reports the details on how to derive from the Z-order space filling curve the local cardinality functions I''_i for a binary tree hierarchy in any dimension.

4.1 Indexing the Lebesgue Space Filling Curve

The Lebesgue space filling curve, also called Z-order space filling curve for its shape in the 2D case, is depicted in figure 1. In the 2D case the curve can be defined inductively by a base Z shape of size 1 (figure 1a) whose vertices are replaced each by a Z shape of size $\frac{1}{2}$. The vertices obtained are then replaced by Z shapes of size $\frac{1}{4}$ (figure 1c) and so on. In general the i^{th} level of resolution is defined as the curve obtained by replacing the vertices of the $(i-1)^{th}$ level of resolution with Z shapes of size $\frac{1}{2^i}$. The 3D version of this space filling curve has the same hierarchical structure with the only difference that the basic Z shape is replaced by a connected pair of Z shapes lying on the opposite faces of a cube as shown in Figure 1f. Figure 1f-j shows five successive refinements of the 3D Lebesgue space filling curve. The d-dimensional version of the space filling curve has also the same hierarchical structure where the basic shape (the Z of the 2D case) is defined as a connected pair of $(d-1)$-dimensional basic shapes lying on the opposite faces of a d-dimensional cube.

The property that makes the Lebesgue's space filling curve particularly attractive is the easy conversion from the d indices of a d-dimensional matrix to the 1D index along the curve. If one element e has d-dimensional reference $(i_1, \ldots, i_d)$ its 1D reference is built by interleaving the bits of the binary representations of the indices $i_1, \ldots, i_n$. In particular if i_j is represented by the string of h bits "$b_j^1 b_j^2 \cdots b_j^h$" (with $j = 1, \ldots, d$) then the 1D reference of e is represented the string of hd bits $I =$ "$b_1^1 b_2^1 \cdots b_d^1 b_1^2 b_2^2 \cdots b_d^2 \cdots b_1^h b_2^h \cdots b_d^h$". Figure 2 shows this interleaving scheme in the 3D case.

The 1D order can be structured in a binary tree by considering elements of level i those that have the last i bits all equal to 0.

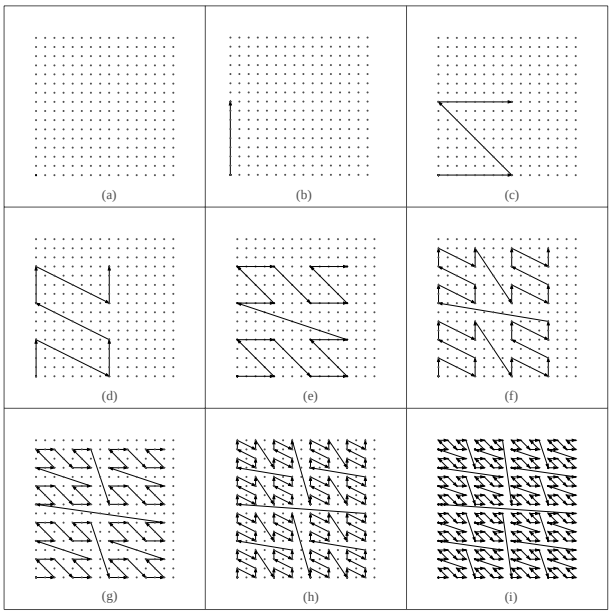

Figure 3: The nine levels of resolution of the binary tree hierarchy defined by the 2D space filling curve applied on 16×16 rectilinear grid. The coarsest level of resolution (a) is a single point. The number of points that belong to the curve at any level of resolution (b-i) is double the number of points of the previous level.

This yields a hierarchy where each level of resolution has twice as many points as the previous level. From a geometric point of view this means that the density of the points in the d-dimensional grid is doubled alternatively along each coordinate axis. Figure 3 shows the binary hierarchy in the 2D case where the resolution of the space filing curve is doubled alternatively along the x and y axis. The coarsest level (a) is a single point, the second level (b) has two points , the third level (c) has four points (forming the Z shape) and so on.

4.2 Index Remapping

The cardinality function discussed in section 3 for a binary tree case has the structure shown in table 1. Note that this is a general structure suitable for out-of-core storage of static binary trees. It is independent of the dimension d of the grid of points or of the Z-order space filling curve.

The structure of the binary tree defined on the Z-order space filling curve allows to determine easily the three elements that are necessary for the computation of the cardinality which are: (i) the level i of an element, (ii) the constants C_i of equation (1) and (iii) the local indices I''_i.

i - if the binary hierarchy has k levels then the element of Z-order index j belongs to the level $k-h$ where h is the number of trailing zeros in the binary representation of j;

C_i - the number of elements in the levels coarser than i, with $i > 0$, is $C_i = 2^{i-1}$ with $C_0 = 0$;

I''_i - if an element has index j and belongs to the set S'_i then $\frac{j}{2^i}$ is

level	0	1	2		3				4							
Z-order index (2 levels)	0	1														
Z-order index (3 levels)	0	2	1	3												
Z-order index (4 levels)	0	4	2	6	1	3	5	7								
Z-order index (5 levels)	0	8	4	12	2	6	10	14	1	3	5	7	9	11	13	15
hierarchical index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Table 1: Structure of the hierarchical indexing scheme for binary tree combined with the order defined by the Lebesgue space filling curve.

an odd number. Its local index is then:

$$I_i''(j) = \left\lfloor \frac{j}{2^{i+1}} \right\rfloor .$$

These three elements can be put together to build an efficient algorithm that computes the hierarchical index $I'(s) = C_i + I_i''(s)$ in the two steps shown in the diagram of Figure 4:

1. set to 1 the bit in position $k+1$;
2. shift to the right until a 1 comes out of the bit-string.

Clearly this diagram could have a very simple and efficient hardware implementation. The software C++ version can be implemented as follows:

```
inline adhocidex remap(register adhocindex i){
  i |= last_bit_mask;  // set leftmost one
  i /= i&-i;           // remove rightmost zeros
  return (i>>1);       // remove rightmost+ one
}
```

This code would work only on machines with two's complement representation of numbers. In a more portable version one needs to replace `i /= i&-i` with `i /= i&((~i)+1)`.

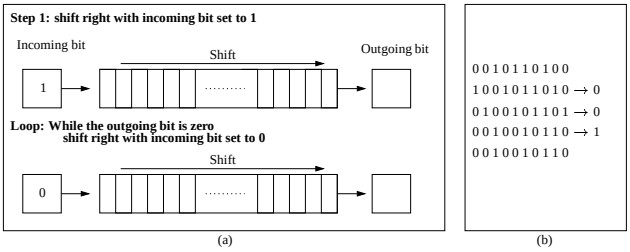

Figure 4: (a) Diagram of the algorithm for index remapping from Z-order to the hierarchical out-of-core binary tree order. (b) Example of the sequence of shift operations necessary to remap an index. The top element is the original index the bottom is the output remapped index.

5 2^n tree and the Lebesgue space filling curve

In some occurrences it might be appropriate to use a blocking scheme that follows better the underlying quad-tree/octtree structure of the hierarchy. For example when large disk blocks are being used it is better to use a hierarchy with less levels of resolution than the basic binary tree scheme in the previous section. At this end one can apply the following generalized version of the index remapping scheme.

The scheme of Figure 4 correspond to a binary tree because the sequence of shifts in the loop are performed one bit at a time. To align the data layout to a 2^l tree one needs to shifts the bit-string by l bits at a time. For example $l = 2$ aligns the data to a quad-tree and $l = 3$ aligns the data to an octtree.

The Z-order index of the data is shifted to the right in blocks of l bits to determine the level in the hierarchy. Following the notation of section 3 we have:

i - is the number of trailing sequences of l zeros at the end of the bit-string;

C_i - the number of elements in the levels coarser than i, with $i > 0$, is $C_i = \left(2^l\right)^{i-1}$ with $C_0 = 0$;

I_i'' - if an element has index j and belongs to the set S_i' then $\left\lfloor \frac{j}{2^l} \right\rfloor$ has one of its last l bits different from 0. The local index is then:

$$I_i''(j) = \left\lfloor \frac{j}{2^{li}} \right\rfloor - \left\lfloor \frac{j}{2^{l(i+1)}} \right\rfloor - 1.$$

The computation of the final index $I'(s) = C_i + I_i''(s)$ is then performed with the scheme of Figure 5. Note how the term sum $C_i + I_i''(s)$ is computed directly by shifting $I(s)$ to the right and adding the complement of its high bits. This can be done because $C_i = 2^{l(i-1)}$ and hence $C_i - \left\lfloor \frac{j}{2^{l(i+1)}} \right\rfloor - 1$ can be obtained directly by complementing the rightmost $l(k-i-1)$ bits of $\left\lfloor \frac{j}{2^{l(i+1)}} \right\rfloor$.

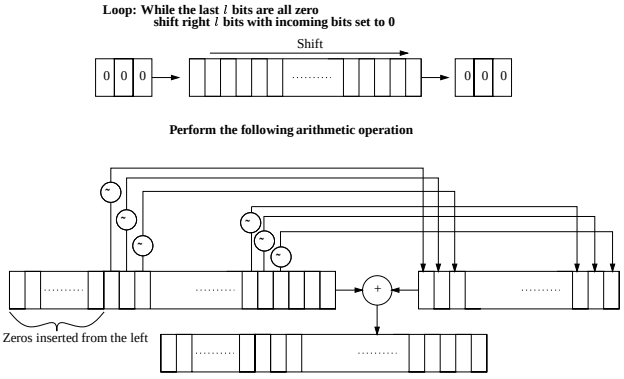

Figure 5: Diagram of the algorithm for index remapping for blocks that scale with a factor of 8 (three bits).

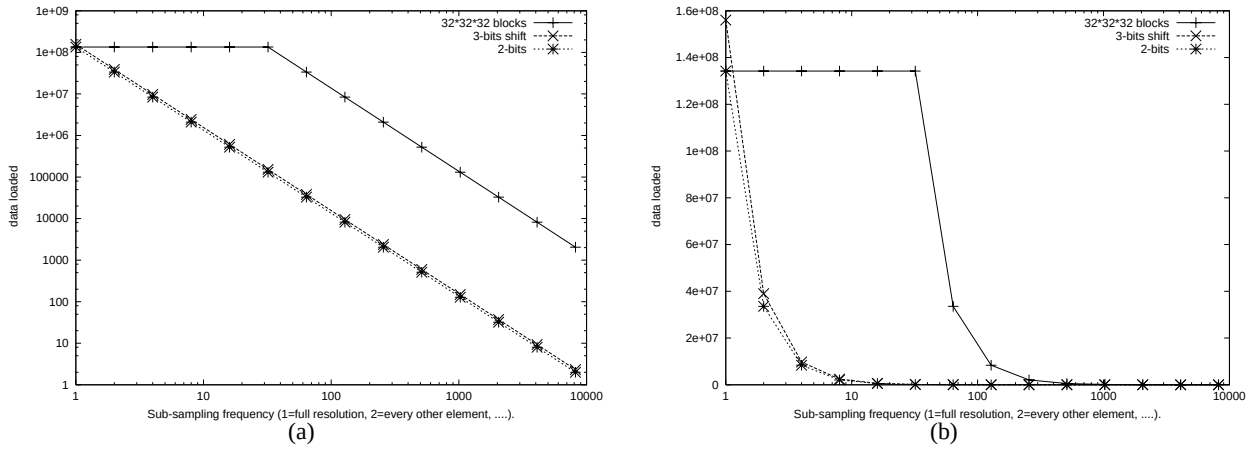

Figure 6: Maximum data loaded from disk (vertical axis) per slice computed depending on the level of subsampling (horizontal axis) for a 8G dataset. (a) Comparison of the blocking approach with the 1-bit shift and 3-bits shifts schemes proposed here. The values on the vertical axis are reported in logarithmic scale to highlight the difference in orders of magnitude at any level of resolution. (b) Same plot with vertical axis in linear scale that highlights the difference in absolute values at the finest level of resolution.

6 Preliminary Results: Orthogonal Slicing

This section presents some preliminary results based on the simple application of computing orthogonal slices of a 3D rectilinear grid of data at different levels of resolution. The data layout proposed here is compared with the two most common array layouts: the standard row major structure and the $h \times h \times h$ block-wise decomposition of the data. Both practical performance tests and formal complexity analysis lead to the conclusion that the data layout proposed here allows to achieves substantial speedup when used at coarse resolution or in a progressive fashion with acceptable performance penalty if used only at the highest level of resolution.

6.1 Out-of-core Complexity Analysis

The out-of-core analysis reports the number of data blocks transferred from disk under the assumption that each block of data of size b is transferred in one operation independently from how much data in the block is actually used. At fine resolution the simple row major array storage achieves the best and worst performances depending on the slicing direction. If the overall grid size is g and the size of the output is t then the best slicing direction requires to load $O(t/b)$ data blocks (which is optimal) but the worst possible direction requires to load $O(t)$ blocks (for $b = \Omega(\sqrt[3]{g})$). In the case of simple $h \times h \times h$ data blocking (which has best performance for $h = \sqrt[3]{b}$) the blocks of data loaded at fine resolution are $O(\frac{t}{\sqrt[3]{b^2}})$. Note that this is much better than the previous case because the performance is close to (even if not) optimal independently from the particular slicing direction. For subsampling rate of k the performance degrades to $O(\frac{tk^2}{\sqrt[3]{b^2}})$ for $k < \sqrt[3]{b}$. This means that at coarse subsampling the performance goes down to $O(t)$. The advantage of the scheme proposed here is that independently from the level of subsampling each block of data is used for a portion of $\sqrt[3]{b^2}$ so that independently from the slicing direction and subsampling rate the worst case performance is $O(\frac{t}{\sqrt[3]{b^2}})$. This implies that the fine resolution performance of the scheme is equivalent to the standard blocking scheme while at coarse resolutions it can get orders of magnitude better. More importantly this allows to produce coarse resolution outputs at interactive rate independently from the total size of the data-set.

A more accurate analysis can be performed to take into account the constant factors that are hidden in the big O notation and determine exactly which approach requires to load into memory more data from disk. We can focus our attention to the case of a 8G bytes data-set with disk pages with of the order of 10K each as shown in diagram of Figure 6. One slice of data is 4M bytes large. In the standard blocking case one would use $32 \times 32 \times 32$ blocks. The data loaded from disk for a slice is 32 times larger than the output, that is 128M bytes. As the subsampling increases up to a value of 32 (one sample out of 32) the amount of data loaded does not decreases because each $32 \times 32 \times 32$ block needs to be loaded completely. At lower subsampling rates the data overhead remains the same: the data loaded is 32768 times larger than the data needed. In the 3-bits shift case the data layout is equivalent to an octtree which maps to a quadtree on the slice. The data loaded is grouped in blocks along the hierarchy that gives an overhead factor in number of blocks of $1 + \frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \cdots < \frac{4}{3}$ while each block is 28K. This means that at the finest resolution there is a overhead of about 16.3%. The advantage is that each time the subsampling rate is doubled the amount of data loaded from external memory is reduced by a factor of four. The same analysis for the 1-bit shift case of section 4 would induce similarly an overhead factor of 2 for the number of blocks at finest resolution. To have each block cover an equivalent region of the mesh they need to have half size that is 16K. A reduction by a factor of four of the number of blocks used is obtained each time the subsampling rate is doubled.

6.2 Experimental Tests

A series of tests have been performed to verify the performance of the approach. The out-of-core component of the scheme has been implemented simply by mapping a 1D array of data to a file on disk using the `mmap` function. In this way the I/O layer is implemented by the operating system that pages in and out a portion of the data array as needed. No multi-threaded component is used to avoid blocking the application while retrieving the data. The blocks of data defined by the system are typically 4Kbytes. Figure 7(a) shows performance tests executed on a Pentium III Laptop 500Mhz with 128M of RAM accessing a 1Gbyte dataset ($1k \times 1k \times 1k$ grid of char). The two schemes proposed here, 3-bits shift from section 5 and 1-bits shift from section 4 show the best scalability

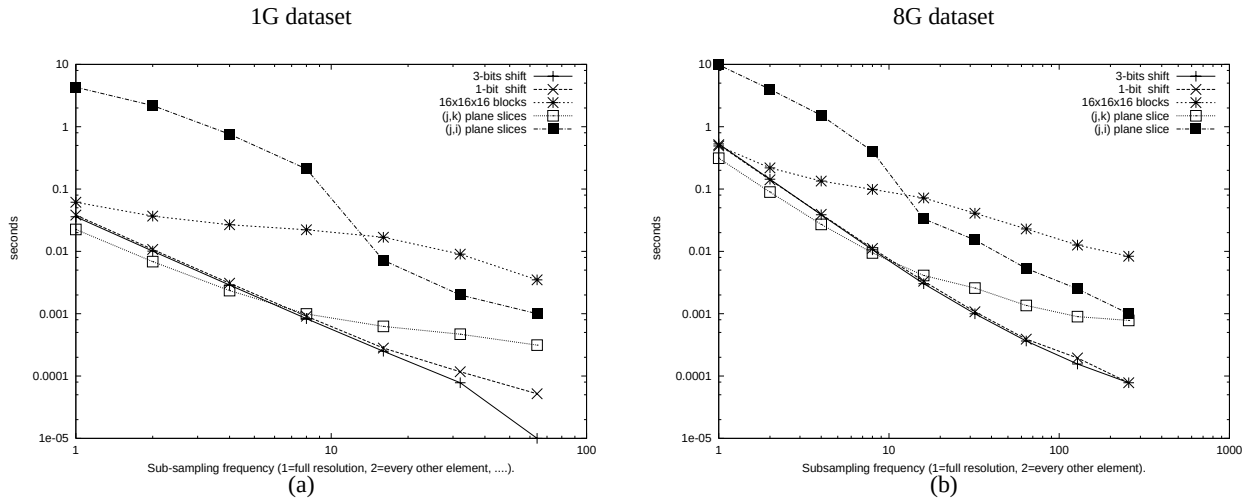

Figure 7: Two comparisons of the computation times of four different data layout schemes. The vertical axis is the computation time in seconds. The horizontal axis is the level of subsampling of the slicing scheme (test at the finest resolution are on the left). The two schemes proposed here, 3-bits shift from section 5 and 1-bits shift from section 4, show best overall performance.

in performance. The blocking scheme with 16^3 chunks of regular grids shows the next best compromise in performance. The (i, j, k) row major storage scheme has the worst performance compromise because of its dependency on the slicing direction: best for (j, k) plane slices and worst for (j, i) plane slices. Figure 7(b) shows the performance results for a test run on an SGI MIPS R12000 300Mhz with 600M of memory available for the application. In this case the dataset is 8G ($2k \times 2k \times 2k$ grid of char).

7 Variations on a Theme

The basic scheme presented in section 3 has been implemented in sections 4 and 5 for the subsampling of cube grids with power of two side and based on the Z-order space filing curve. Many variations on the same theme can be also implemented depending on the requirements of a specific dataset while maintaining the same performance gain. The following of this section reports a few simple examples of possible variations.

7.1 Simple Rectangular Grids

The case of a rectangular grid of size $2^{n_1} \times \cdots \times 2^{n_d}$ can be cast in the same framework with a simple modification of the bit interleaving function shown in Figure 2 of the 3D case. A global index of $n = \sum_{i=1}^{d} n_i$ bits can be built simply interleaving unevenly the bits from each index. Figure 8 shows and example of this generalized Z-order. What bits go at the beginning or at the end of the global index determined which data belongs to the coarse levels of resolution and which is fine resolution information.

7.2 Edge Bisection Refinement

If the regular grid is used for edge bisection refinement [24, 11, 19] it must have size $(2^{n_1}+1) \times \cdots \times (2^{n_d}+1)$. Three aspects need to be taken care of: (i) the grid may be rectangular, (ii) the resolution in each direction is $2^{n_i} + 1$ instead of 2^{n_i}, and (iii) the order of refinement is not exactly the same as in the binary tree of section 4.

The best bit interleaving to generate the global index is achieved by using as leading bits the fist $m_i = n - n_i$ of each index, where $n = \min(n_1, \ldots, n_d)$. The rest is done as a regular interleaving as in Figure 2. In this way the hierarchy corresponds to the edge

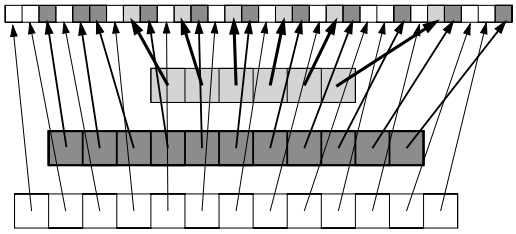

Figure 8: Generalization of construction of the 1D index from the Lebesgue's Z-order space filling curve for restricted rectangular grids.

bisection applied to a coarse mesh of size $(2^{m_1}+1) \times \cdots \times (2^{m_d}+1)$.

The fact that the resolution in each direction is $2^{n_i} + 1$ instead of 2^{n_i} can be dealt with by storing first the grid of size $(2^{n_1}) \times \cdots \times (2^{n_d})$ followed by d grids of dimension $d-1$ and $\binom{d}{2}$ grids of dimension $d-2$ and so on. The index is computed in the usual way form the d indices $(i_1, \ldots, i_d)$ if all of them have leading bit equal to zero. If one or more indices have leading bit set to 1 a lower dimensional grid index is computed using only the indices with leading bit equal to 0. The additional offset necessary can be precomputed and stored in a table of $\sum_{i=0}^{d} \binom{d}{i}$ elements.

Finally the hierarchy needs to be aligned as much as possible with the order of insertion of the vertices in an edge bisection hierarchy. The binary hierarchy applied to the Z-order curve yields the structure of Figure 3 where the number of vertices is double along each axis independently. In the edge bisection hierarchy (see Figure 9a) the new vertices are added first at the center of each square and then in the middle of vertical/horizontal edges alternatively.The same hierarchy is achieved by replacing the Z-order space filling curve with the curve of Figure 9b. In terms of computation of the global index one needs to combine the bit interleaving step with the mapping of each pair of bits with the following table:

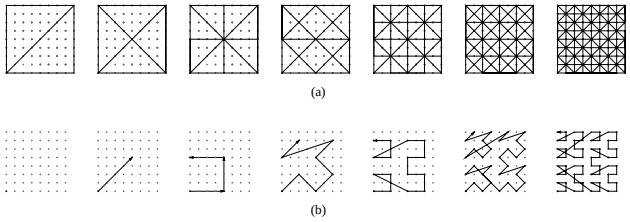

Figure 9: (a) 2D edge bisection refinement sequence. (b) Curve that replaces the Z-order to obtain a hierarchy fully synchronized with the edge bisection subdivision.

0	0	$\mapsto$	0	0
1	1	$\mapsto$	1	0
0	1	$\mapsto$	0	1
1	0	$\mapsto$	1	1

Similar result can be achieved in the 3D case using the following bit remapping table:

0	0	0	$\mapsto$	0	0	0
1	1	1	$\mapsto$	1	0	0
0	1	1	$\mapsto$	0	1	0
1	0	1	$\mapsto$	1	1	0
1	1	0	$\mapsto$	0	0	1
1	0	0	$\mapsto$	1	0	1
0	1	0	$\mapsto$	0	1	1
0	0	1	$\mapsto$	1	1	1

Unfortunately in the 3D case the simple index remapping adjusts the hierarchy only partially because in the edge bisection the number of vertices is not exactly doubled at each refinement (even if it is on average). In particular in the standard binary tree sequence one vertex is replaced by two then by four and finally eight vertices. In the edge bisection one vertex is replaced by two then by five and finally by eight. This makes it impossible to have a fully aligned binary hierarchy. A more complicated implementation of the basic approach of section 3 would be needed.

7.3 Not Only Subsampling

It is important to remark that the scheme introduced here deal only with the out-of-core data layout of a hierarchy. With reference to the general scheme of section 3 there is no actual constraint for the set S_i' to be equal to $S_i \backslash S_{i-1}$. Depending on the application it may be sufficient that the sequence $\{S_0, \ldots, S_i'\}$ of the coarse levels up to the resolution i allows to reconstruct the original S_i within a certain approximation error. In other terms this seems an appropriate data layout for a wavelet representation, even if it is not obvious how to deal with the problem of the eventual compression that one might also want to achieve, since it changes the location (but not the order) of the data.

8 Conclusions and Future Direction

The present paper introduces a new indexing and data layout scheme that is useful for out-of-core hierarchical traversal of large datasets. Practical tests and theoretical analysis for a simple case of orthogonal slicing show the performance improvements that can be achieved with this approach especially in a progressive computation setting. In the near future this scheme is going to be used as a basis for out-of-core computation of general slices, isocontours and navigation of large terrains.

Future direction that are being considered include the combination with wavelet compression schemes, the extension to general rectangular grids and to non-rectilinear hierarchies.

References

[1] James Abello and Jeffrey Scott Vitter, editors. *External Memory Algorithms and Visualization*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society Press, Providence, RI, 1999.

[2] Lars Arge and Peter Bro Miltersen. On showing lower bounds for external-memory computational geometry problems. In James Abello and Jeffrey Scott Vitter, editors, *External Memory Algorithms and Visualization*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society Press, Providence, RI, 1999.

[3] T. Asano, D. Ranjan, T. Roos, and E. Welzl. Space filling curves and their use in the design of geometric data structures. *Lecture Notes in Computer Science*, 911:36–44, 1995.

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

[5] L. Balmelli, J. Kovačević, and M. Vetterli. Quadtree for embedded surface visualization: Constraints and efficient data structures. In *IEEE International Conference on Image Processing (ICIP)*, Kobe Japan, October 1999.

[6] L. Balmelli, J. Kovačević, and M. Vetterli. Solving the coplanarity problem of regular embedded triangulations. In *Proceeding of the Workshop on Vision, Modeling and Visualization*, November 1999.

[7] Y. Bandou and S.-I. Kamata. An address generator for a 3-dimensional pseudo-hilbert scan in a cuboid region. In *International Conference on Image Processing, ICIP99*, volume I, 1999.

[8] Y. Bandou and S.-I. Kamata. An address generator for an n-dimensional pseudo-hilbert scan in a hyper-rectangular parallelepiped region. In *International Conference on Image Processing, ICIP 2000*, 2000. to appear.

[9] Siddhartha Chatterjee, Alvin R. Lebeck, Praveen K. Patnala, and Mithuna Thottethodi. Recursive array layouts and fast parallel matrix multiplication. In *Proceedings of the 11th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 222–231, Saint-Malo, France, June 27–30, 1999. SIGACT/SIGARCH and EATCS.

[10] Yi-Jen Chiang and Cláudio T. Silva. I/O optimal isosurface extraction. In Roni Yagel and Hans Hagen, editors, *IEEE Visualization ý7*, pages 293–300. IEEE, November 1997.

[11] Mark A. Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstein. Roaming terrain: Real-time optimally adapting meshes. *IEEE Visualization '97*, pages 81–88, November 1997.

[12] Jeremy D. Frens and David S. Wise. Auto-blocking matrix-multiplication or tracking BLAS3 performance from source code. *ACM SIGPLAN Notices*, 32(7):206–216, July 1997.

[13] M. T. Goodrich, J.-J. Tsay, D. E. Vengroff, and J. S. Vitter. External-memory computational geometry. In *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science (FOCS '93)*, Palo Alto, CA, November 1993.

[14] M. Griebel and G. W. Zumbusch. Parallel multigrid in an adaptive pde solver based on hashing and space-filling curves. 25:827:843, 1999.

[15] D. Hilbert. Über die stetige abbildung einer linie auf ein flachenstück. *Mathematische Annalen*, 38:459–460, 1891.

[16] S.-I. Kamata and Y. Bandou. An address generator of a pseudo-hilbert scan in a rectangle region. In *International Conference on Image Processing, ICIP97*, volume I, pages 707–714, 1997.

[17] J. K. Lawder. *The Application of Space-filling Curves to the Storage and Retrieval of Multi-Dimensional Data*. PhD thesis, School of Computer Science and Information Systems, Birkbeck College, University of London, 2000.

[18] J. K. Lawder and P. J. H. King. Using space-filling curves for multi-dimensional indexing. In Brian Lings and Keith Jeffery, editors, *proceedings of the 17th British National Conference on Databases (BNCOD 17)*, volume 1832 of *Lecture Notes in Computer Science*, pages 20–35. Springer Verlag, July 2000.

[19] Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hughes, Nick Faust, and Gregory Turner. Real-time, continuous level of detail rendering of height fields. *Proceedings of SIGGRAPH 96*, pages 109–118, August 1996.

[20] Y. Matias, E. Segal, and J. S. Vitter. Efficient bundle sorting. In *Proceedings of the 11th Annual SIAM/ACM Symposium on Discrete Algorithms (SODA '00)*, January 2000.

[21] R. Niedermeier, K. Reinhardt, and P. Sanders. Towards optimal locality in meshindexings, 1997.

[22] Rolf NIEDERMEIER and Peter SANDERS. On the manhattan-distance between points on space-filling mesh-indexings. Technical Report iratr-1996-18, Universität Karlsruhe, Informatik für Ingenieure und Naturwissenschaftler, 1996.

[23] M. Parashar, J.C. Browne, C. Edwards, and K. Klimkowski. A common data management infrastructure for adaptive algorithms for pde solutions. In *SuperComputing 97*, 1997.

[24] V. Pascucci and C. L. Bajaj. Time critical isosurface refinement and smoothing. In *IEEE Symposium on Volume Visualization and Graphics 2000*, October 2000. To Appear.

[25] M. C. Rivara. Algorithms for refining triangular grids suitable for adaptive and multigrid techniques. *International Journal for Numerical Methods in Engineering*, 20:745–756, 1984.

[26] Hans Sagan. *Space-Filling Curves*. Springer-Verlag, New York, NY, 1994.

[27] Hanan Samet. *Applications of Spatial Data Structures*. Addison-Wesley, Reading, Mass., 1990. chapter on ray tracing and efficiency, also discusses radiosity.

[28] J. S. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, March 2000.

[29] David S. Wise. Ahnentafel indexing into morton-ordered arrays, or matrix locality for free. In *Euro-Par 2000 – Parallel Processing*, volume 1900 of *Lecture Notes in Computer Science*, pages 774–784. Springer, August 2000.