Advances in Engineering Software 82 (2015) 75–86

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Data structures and algorithms for high-dimensional structured adaptive mesh refinement

Magnus Grandin

Division of Scientific Computing, Dept. of Information Technology, Uppsala University, Box 337, SE-751 05 Uppsala, Sweden

ARTICLE INFO

Article history:
Received 23 September 2014
Received in revised form 1 December 2014
Accepted 10 December 2014
Available online 28 January 2015

Keywords:
Structured adaptive mesh refinement
Anisotropic mesh
High-dimensional
Hierarchical data structure
kd-tree
Morton order
2:1 balancing

ABSTRACT

Spatial discretization of high-dimensional partial differential equations requires data representations that are of low overhead in terms of memory and complexity. Uniform discretization of computational domains quickly grows out of reach due to an exponential increase in problem size with dimensionality. Even with spatial adaptivity, the number of mesh data points can be unnecessarily large if care is not taken as to where refinement is done. This paper proposes an adaptive scheme that generates the mesh by recursive bisection, allowing mesh blocks to be arbitrarily anisotropic to allow for fine structures in some directions without over-refining in those directions that suffice with less refinement. Within this framework, the mesh blocks are organized in a linear kd-tree with an explicit node index map corresponding to the hierarchical splitting of internal nodes. Algorithms for refinement, coarsening and 2:1 balancing of a mesh hierarchy are derived. To demonstrate the capabilities of the framework, examples of generated meshes are presented and the algorithmic scalability is evaluated on a suite of test problems. In conclusion, although the worst-case complexity of sorting the nodes and building the node map index is n^2, the average runtime scaling in the studied examples is no worse than $n \log n$.

1. Introduction

Structured adaptive mesh refinement (SAMR) is an active area of research within the scientific computing community [1]. By adjusting the resolution of the computational mesh dynamically to features in the solution or the computational domain, widely varying scales of resolution can be represented simultaneously. In effect, the computational efficiency of a simulation is improved, possibly by orders of magnitude, allowing for larger computations and/or shorter execution times due to a reduction in the number of gridpoints [2]. For problems in two and three dimensions, there are efficient algorithms and data structures available, relying on quad/octrees for structuring the mesh blocks [2–5]. However, extending quadtrees and octrees to higher dimensional trees is problematic since they yield a fan-out of $2^{\mathcal{D}}$ nodes at every branch, which leads to an exponential increase in the potential number of tree nodes to handle. The contribution of this paper is a framework capable of generating and propagating meshes of arbitrary dimensionality. Our approach is based on recursive bisection and generates far fewer mesh nodes compared to $2^{\mathcal{D}}$-trees of corresponding refinement.

In order to construct a practical refinement scheme that works well even in higher dimensions, the framework presented in this paper is built on a structured block-based refinement strategy [1] allowing blocks to be refined *anisotropically*. The mesh nodes and their mutual relationships are maintained in a *kd-tree* [6]. With anisotropic refinement, a block is not restricted to be refined equally in all dimensions, potentially leading to a more efficient discretization in terms of the number of created blocks since mesh blocks are refined only in the dimensions in which they would benefit from finer resolution. The grid is refined successively by dividing blocks in half, dimension by dimension. If a block needs refinement in more than one dimension, this is done by subsequent division in several steps. This paper does not consider details regarding error estimation and how to determine when refinement/coarsening is required. For the anisotropic refinement strategy to be useful though, the error estimator must be able to detect the discretization error per dimension. An example of such an error estimator is given in [7].

A kd-tree is a binary tree representation of a hierarchical subdivision of a $\mathcal{D}$-dimensional hyperrectangle by recursive bisection [6,8–11]. The interior nodes of a kd-tree represent hyperplane cuts, aligned with the Cartesian coordinate axes, and the leaf nodes contain the actual data. In a general kd-tree, a cut can be placed anywhere along the split dimension of a block. However, by restricting the cuts to always be placed in the middle of the block (which we do by imposing a halving of the blocks on each refinement) the scheme is simplified significantly. Furthermore, implementations of kd-trees usually assign split dimensions to nodes cyclically, such

E-mail address: magnus.grandin@it.uu.se

http://dx.doi.org/10.1016/j.advengsoft.2014.12.001
0965-9978/

that a node at tree level l is split in dimension (l mod $\mathcal{D}$). In the implementation presented here, this restriction is relaxed and nodes are allowed to be split arbitrarily without any intermediate refinement in the other dimensions. Thus, blocks can become as elongated as is needed and in principle there is no restriction on the aspect ratio of the blocks.

It is often motivated for reasons of efficiency and memory requirements that a pointer representation of a tree structure should be avoided. By storing locality information in each mesh node and structuring the nodes in a linear order according to this information, the internal structure of a tree is available implicitly and no pointers are needed for searching and navigating it [2–4,12]. With the leaf nodes stored in linear order (e.g. the Morton order space-filling curve [13]), tree search is replaced by binary search, which is further advantageous in terms of search complexity; a tree representing an adaptively refined mesh is potentially very unbalanced with a search complexity approaching $\mathcal{O}(n)$ in the number of leaf nodes, whereas binary search in the linear representation is always $\mathcal{O}(\log_2 n)$ [4]. This paper extends previous work by other authors [2,4] and builds upon linear Morton order trees. The data structures and algorithms are generalized to enable an extension to higher dimensionalities. However, the anisotropic node refinement does not map directly to an efficient consecutive order of elements solely from the locality information in the leaf nodes. In order to alleviate this, the linear tree representation is extended with a lightweight representation of the internal node structure (a *node map index*), corresponding to a *hierarchical* Morton order of nodes. The node map index is constructed at the same time as the nodes are sorted and stored in a compact array representation for efficient traversal.

The remainder of this paper is organized as follows. In Section 2, some related work is discussed. A key concept in our implementation is the notion of location codes, which is described in Section 3. In Section 4 the construction of a hierarchical Morton order index is derived together with some implementation details. The algorithms for tree search are discussed in Section 5, followed by the adaptive mesh algorithms in Section 6. Finally, Section 7 gives some results of our implementation and Section 8 concludes the paper.

2. Related work

To our knowledge, there are no dynamically adaptive frameworks available that support anisotropic SAMR on hyperrectangular domains in $\mathcal{D}$ dimensions. Klöfkorn and Nolte described the implementation of the `SPGrid` interface of the DUNE framework [14,15], which allows anisotropic structured grids of higher dimensionality but only handles static meshes. An alternative approach to tackling higher dimensions is to discretize the domain into structured simplex meshes [16–19]. In [11], a two-level approach of combining hypercubes and simplices is presented.

The most typical uses of kd-trees are within domain decomposition for clustering of scattered point data, nearest neighbor search and raytracing, although they were originally developed for efficient searching in multidimensional data bases [6]. It is straightforward to adopt the kd-tree structure to any form of domain decomposition problem. This paper demonstrates how kd-trees can be used for adaptive mesh refinement by subdividing the computational domain until every block in the domain constitutes a fine enough grid resolution for the error in that part of the grid to be below some tolerance. Although presented in this setting, the techniques presented in this paper can potentially be useful in more general kd-trees and for other applications.

Samet and Tamminen [20,21] developed a tree structure that is closely related to the kd-tree, which they refer to as the binary image tree (bintree). It is a pointerless tree structure developed for connected component labeling [21] of $\mathcal{D}$-dimensional images, connecting pixels of an image into a hierarchy of larger objects. The tree structure representation of the pointerless bintree is similar to the way our trees are represented, and their algorithms for connected component labeling share some features with the search algorithms presented in this paper.

Throughout this paper, the Morton order space-filling curve is used for linear indexing of mesh elements, due to its simplicity and direct correspondence to recursive bisection and binary trees. Other alternatives have better locality properties, e.g. the Hilbert-curve and the Peano-curve [22], but are more complex to compute and not straightforward to use with the anisotropic refinement presented in this paper.

3. Location codes

A mesh block represents a hyperrectangle (orthotope) subdomain of the discretized $\mathcal{D}$-dimensional space. The *location code* [2,4] of an orthotope is a unique identifier that encodes the location and refinement of the corresponding mesh element. The location code has two components; (1) a coordinate in space with respect to the orthotope anchor, and (2) refinement level information (per dimension). The anchor of an orthotope is chosen by convention to be the lowermost corner in every direction, given by $\mathcal{D}$ integer coordinates. Fig. 1 depicts an example node hierarchy and the corresponding mesh with location codes indicated for each mesh node. The integer representation of coordinates directly correspond to the orthotope's location in the unit hypercube and allows for efficient bitwise integer operations. The actual coordinates of mesh blocks in the domain can be obtained by scaling the unit hypercube coordinates to the domain boundaries. Due to the anisoptropic refinement, separate refinement levels must be stored in each dimension ($\mathcal{D}$ integers). This is a generalization of the location codes used with linear quad/octrees, which only have one integer component for the refinement level due to their isotropic nature.

The coordinates of an orthotope are constructed and stored as follows [2]. Each coordinate is represented by $\mathcal{B}$ bits, for a total of $\mathcal{D}\mathcal{B}$ bits for the entire coordinate set of an orthotope. Subdividing an orthotope at level l in dimension x_i updates the lth bit in the ith coordinate accordingly. The leftmost bit of a coordinate is the most significant and corresponds to the first subdivision in the corresponding dimension, the second leftmost bit corresponds to the second subdivision, and so forth. The bits that are below the refinement level of an orthotope are set to 0. Thus, the integer size $\mathcal{B}$ relates to the maximum level of refinement in any dimension and requires only $\lceil \log_2 \mathcal{B} \rceil$ bits per dimension for storage of the refinement levels. It is straightforward to combine the coordinates and refinement levels in a single integer per dimension, and given 32-bit integers for the storage this still allows for 27 levels of refinement per dimension to be represented.

3.1. Atoms

For some operations on the tree it is necessary to reason about the smallest possible domain units in the mesh. Throughout this paper, these components are referred to as atoms (also referred to as smallest or least descendants by other authors [2,4]). By definition, an atom is of refinement level $\mathcal{B}$ in all dimensions. The domain and all its subcomponents can be viewed as overlying a fine grid of atoms, such that an orthotope is spanned by its first and last atoms (i.e. the first and last atoms in the range of linear indices that the orthotope covers). Fig. 2 illustrates how an orthotope depends on a hierarchy of coarser orthotopes and how these are mapped onto the grid of atoms. Further indicated in the figure are the anchors and the first and last atoms of each orthotope.