Journal of Computational Physics 351 (2017) 473–477

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

Short note

A short note on the use of the red–black tree in Cartesian adaptive mesh refinement algorithms

Jaber J. Hasbestan [1], Inanc Senocak [*,2]

Department of Mechanical Engineering and Materials Science, University of Pittsburgh, Pittsburgh, 15261, PA, USA

ARTICLE INFO

Article history:
Received 29 July 2017
Received in revised form 26 September 2017
Accepted 27 September 2017
Available online 29 September 2017

Keywords:
Adaptive mesh refinement (AMR)
Data locality
Hash table
Linear-octree generation
Red–black tree
Z-order curve

1. Introduction

Mesh adaptivity is an indispensable capability to tackle multiphysics problems with large disparity in time and length scales. With the availability of powerful supercomputers, there is a pressing need to extend time-proven computational techniques to extreme-scale problems. Cartesian adaptive mesh refinement (AMR) is one such method that enables simulation of multiscale, multiphysics problems. AMR is based on construction of octrees. Originally, an explicit tree data structure was used to generate and manipulate an adaptive Cartesian mesh. At least eight pointers are required in an explicit approach to construct an octree. Parent-child relationships are then used to traverse the tree. An explicit octree, however, is expensive in terms of memory usage and the time it takes to traverse the tree to access a specific node. For these reasons, implicit *pointerless* methods have been pioneered within the computer graphics community, motivated by applications requiring interactivity and realistic three dimensional visualization. Lewiner et al. [1] provides a concise review of pointerless approaches to generate an octree. Use of a hash table and Z-order curve are two key concepts in pointerless methods that we briefly discuss next.

A hash table potentially provide $O(1)$ access to its elements owing to the use of an array as the underlying data structure. However, $O(1)$ access is only guaranteed as long as there are no collisions in the hash table. In practice, this requirement is difficult to fulfill. Collisions do happen when the hash function generates the same index for different keys. Typically, a remainder operator is used as the hash function to fit the keys inside a given array.

* Corresponding author.
E-mail addresses: jaber@pitt.edu (J.J. Hasbestan), senocak@pitt.edu (I. Senocak).
[1] Postdoctoral Research Associate.
[2] Associate Professor.

https://doi.org/10.1016/j.jcp.2017.09.056

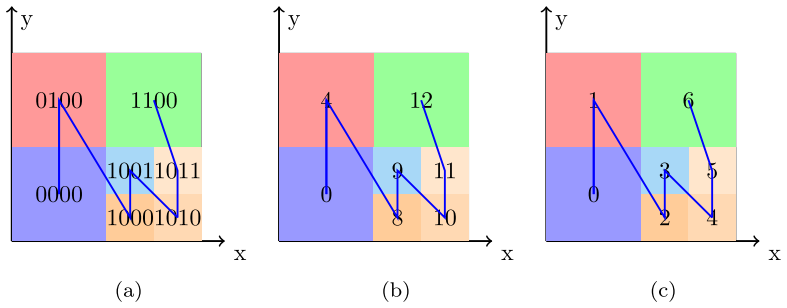

Fig. 1. Illustration of mesh element indexing that arise when a hash table or a red-black tree is used in the AMR algorithm, (a) Morton code representation of mesh elements with two levels of adaptation, (b) integer index values corresponding to the Morton code from (a) in the hash table approach, (c) sorted Morton ordering that results by default when a red–black tree approach is used.

Data locality is key to superior memory performance. It is desirable to store neighboring data closer to each other to preserve data locality. One option to ensure data locality is to track data with space-filling curves. The use of a Z-order curve [2] is popular for that reason.

Morton encoded Z-order curve was introduced in octree construction to enhance data locality [1,3,4]. However, when a hash table is combined with the Z-order curve, perfect data locality can not be guaranteed for complex geometry applications. As we later present in this short note, the Z-order curve compliance can be broken because of collisions in the hash table. Nonetheless, this combination currently represents the state-of-the-art in octree generation in AMR. For example, Burstedde et al. [5] developed `p4est` to generate forest of octrees for large-scale scientific computing. `p4est` has become a popular parallel AMR software and has been adopted in several applications. Burstedde et al. presented a novel encoding scheme to keep track of interoctree connectivity with a 2:1 balance algorithm. In their approach they adopted a linear-octree storage which only stores the leaves of the octree. Burstedde et al. were able to generate a multi-octree mesh with as many as 5.13×10^{11} elements using 220,320 processes.

In a Z-order curve enhanced hash table approach, one needs to convert the Morton code representation of the nodes of the linear-octree to an integer value. This number is then converted to the index of the underlying array in the hash table data structure. In three dimensional space three bits are used to identify the node at each level, which means that the maximum level for deep mesh adaptation is limited to 10 and 21 levels for 32-bit and 64-bit systems, respectively. Conversion to an index is also limited by the maximum integer number representable on a computer, beyond which an overflow is inevitable. This limitation in adaptation level can be relaxed at the expense of accepting collisions in the hash table. But the desirable $O(1)$ complexity of the hash table is then destroyed in the process. Perfect hashing with $O(1)$ access is only possible if the size of the underlying array is large enough to accommodate the full octree, which amounts to 2^{3n}, where n is the maximum level of adaptation and the factor of 3 is due to the three dimensionality of the domain. Allocating such a large array is impractical. Furthermore, because the size of a tree is unknown at compile-time for dynamic mesh adaptation, one needs to leave many empty slots to be filled in later, leading to waste of memory.

To illustrate these issues and the associated remedies, consider the mesh elements as shown in Fig. 1. Fig. 1(a), illustrates the Morton encoding in two-dimensional space for a quad tree with two levels of adaptation. Figs. 1(b) and 1(c), demonstrate the indexing and ordering in hash table and in red–black tree approaches, respectively. As we see from Fig. 1(b) the integer index is not contiguous anymore (i.e. 0, 4, 8, 9 ...), whereas with a red–black tree approach the Morton order is sorted by default and the element indexing complies with the Z-order curve.

To illustrate the collision issue in the hash table approach, let us assume that the initial size of the hash table is 8. Now we try to fit the 7 elements, shown in Fig. 1, in the 8 slots by using the remainder operator, which is used in the hash function. For elements with hash values of 4 and 12, the remainder operator will give 4 for both elements, which is a collision in the 5th slot. In other words, both elements are assigned to the same slot in the hash table. To conform to the Z-order curve and have zero collisions at the same time, the size of the array should be at least 12, which is an increase of 50%. We also need to allocate extra space for possible refinements that might occur in the next step. Obviously, this remedy to avoid collision in the hash table can be prohibitively expensive in terms of memory when a mesh with a large number of refinements are considered. Subsequently, one has to relax the no-collision condition and try to use the remainder operator to fit the mesh elements in the array. Once the no-collision condition is relaxed, $O(1)$ access can not be guaranteed. Another draw back of using a hash table is that rehashing might occur if one does not predict the final octree size with a good approximation. We note that rehashing is computationally expensive.

For large-scale multiphysics problems with a computational mesh on the order of hundred million elements and more, which are now becoming feasible with ever increasing supercomputing power, allocating extra space for a hash table may not be feasible. Paying closer attention to the Z-order curve in Fig. 1 reveals that a sorting process is needed to ensure perfect data locality, which is not available in the hash table approach. To address this shortcoming, we propose adopting a red–black tree [6] (i.e., instead of using a hash table), as the fundamental data structure in the implementation of a Cartesian AMR algorithm. With this proposition we accomplish the following: