



## Short note

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

Jaber J. Hasbestan<sup>1</sup>, Inanc Senocak<sup>\*,2</sup>

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

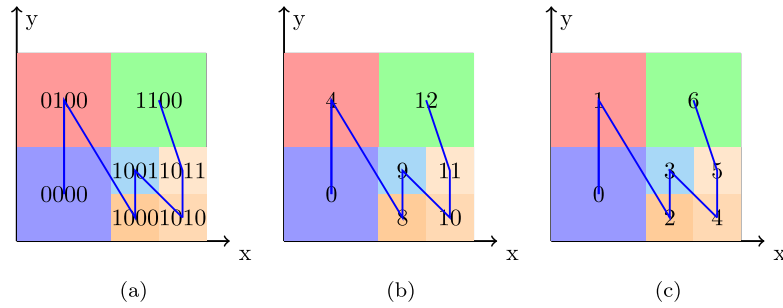
Mesh adaptivity is an indispensable capability to tackle multiphysics problems with large disparity in time and length scales. With the availability of powerful supercomputers, there is a pressing need to extend time-proven computational techniques to extreme-scale problems. Cartesian adaptive mesh refinement (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).

<sup>1</sup> Postdoctoral Research Associate.<sup>2</sup> Associate Professor.



**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.

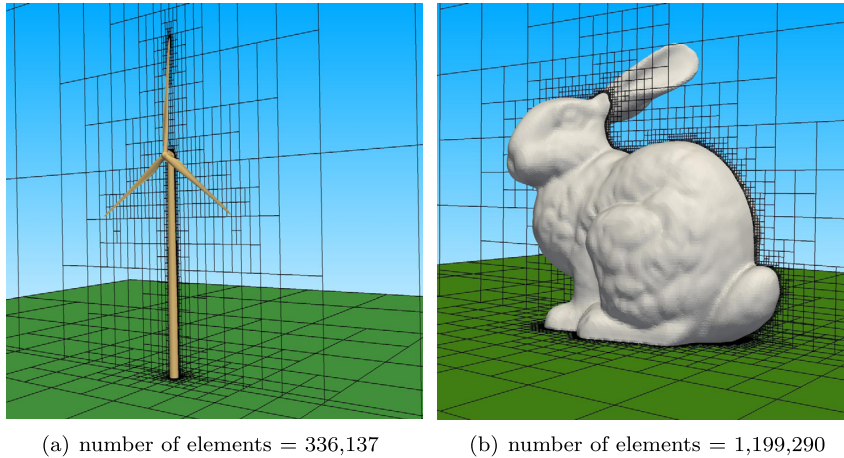
Morton encoded Z-order curve was introduced in octree construction to enhance data locality [1,3,4]. However, when a hash table is combined with the Z-order curve, perfect data locality can not be guaranteed for complex geometry applications. As we later present in this short note, the Z-order curve 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  $5.13 \times 10^{11}$  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:



**Fig. 2.** Mesh adaptation around the wind turbine and Stanford bunny geometries. Maximum adaptation level is 11.

1. *Remove the limitation on maximum level of mesh adaptation.* With a red–black tree there is no need to convert the Morton code to an integer index, because a red–black tree only requires a comparison of the bits.
2. *Abide by the Z-order curve exactly.* The inherent sorted representation in a red–black tree enables perfect data locality.
3. *Demonstrate the potential of the red–black tree data structure for parallel adaptive mesh refinement.* The use of a red–black tree does not adversely effect the parallelization of the overall AMR algorithm.

To the best of our knowledge, we are not aware of any AMR software library that makes use of the red–black tree data structure. Additionally, the issue of loss of data locality in the Z-order curve enhanced hash table has not been discussed in the open literature.

## 2. Cartesian adaptive mesh refinement using a red–black tree

A sorted tree is required to conform to the Z-order curve perfectly and satisfy data locality strictly. Therefore, we propose to adopt a red–black tree in place of a hash table in an AMR algorithm. Red–black trees [7] are widely used in different areas of computer science due to its ability to guarantee worst-case situations for insertion, removal and search operations. Its use ranges from computational geometry to process scheduling in Linux kernels. In this short note, we demonstrate the potential of a red–black tree for adaptive mesh generation for extreme-scale simulation problems.

At its core, a red–black tree is a binary tree with an extra bit for defining the color, which is either red or black for every node. The memory usage of a red–black tree is much less compared to an explicit octree. Although accessing each element is still  $O(\log N)$  for a large mesh, data locality and the speed of memory access offsets the  $O(\log N)$  element access cost of a red–black tree, as we demonstrate in this short note. Red–black tree data structure is readily available in the C++ standard library as `Set/Map` classes. The `Set` class is a special case of the `Map` class in which the associated value is the key itself. We use the Morton code representation of the linear-octree as the comparison criteria for insertion of the new elements in the red–black tree. Note that when comparing two keys represented by bits, conversion of the key to integer is not required anymore, which is a major advantage. We make use of the bitwise operation (`xor`) to perform this comparison. Removing the integer conversion step subsequently eliminates the limitation on the maximum level of adaptation. The `bitset` class is used in conjunction with the `Set/Map` classes to perform these operations.

In simulation problems where there is huge disparity in length scales governing the problem, it is desirable to have a gradual transition from a higher-resolution zone to a lower-resolution zone. This is typically achieved by confining the maximum number of the elements sharing an edge to 2 elements, which has been referred to as 2:1 balanced tree in the literature. We guarantee 2:1 balance in our implementation as well.

## 3. Results and performance metrics

Fig. 2 illustrates the versatility of the current adaptive mesh generation software for two arbitrarily complex geometries. Both the wind turbine and the Stanford are immersed inside a three dimensional Cartesian domain. Table 1 documents the elimination of the maximum level of adaptation constraint for 2:1 balanced linear-octree mesh generation around the wind turbine geometry with 70,406 surface triangles.

To analyze the performance of the AMR techniques described in this short note, we investigate three techniques to generate adaptive meshes around the Stanford bunny with maximum adaptation levels of 10, 12, 14 and 16. The first method is the red–black tree based AMR. The second method is the hash table based AMR with a Z-order curve compliant hash function. Here, the hash function is overwritten in an attempt to comply with the Z-order curve. The third method is

**Table 1**

Number of mesh elements as a function of adaptation level for a Z-order curve compliant mesh that adopts a red–black tree approach. Wind turbine geometry with 70,406 surface triangles is used. Note that the hash table approach limits the maximum level of adaption to 21 on 64-bit systems.

Level	20	21	22	23
Mesh size	1,246,330	2,925,973	6,007,583	10,714,012

**Table 2**

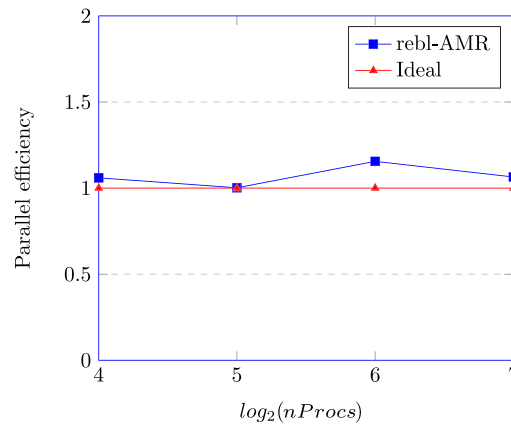
Comparison of total run-time, Stanford bunny is used for this analysis.

Level	10	12	14	16
Red–black tree [seconds]	3.3	21.14	69.75	145.87
Z-order curve enhanced hash function [seconds]	3.7	26.85	87.85	180.01
C++ STL default hash function [seconds]	4.89	39.95	138.19	272.97

**Table 3**

Comparison of memory consumption. Stanford bunny is used for this analysis.

Level	10	12	14	16
Red–black tree [MB]	61.4	281.8	792.9	1500
Hash table [MB]	59.6	261.7	675.5	1300



**Fig. 3.** Strong scaling analysis for the parallel version of `rebl-AMR`. Mesh adaptation with 18 levels of refinement is performed for a sports car geometry. The total number of elements in the final mesh is 189,220,200.,  $nProcs$  is the number of processors

also a hash table based AMR, but it adopts the default hash function available in the GNU C++ STL standard library. This third approach is provided for completeness. C++ Standard Template Library (STL) provides a hash function, in addition to permitting programmers to provide their own hash function via a class interface to override the default function.

Table 2 presents total run-time to generate meshes with different levels of adaptation for the Stanford bunny geometry. The red–black tree presents superior performance compared to the hash table based counterparts because of strict conformity to the Z-order curve. Z-order curve enhanced hash table AMR comes second in performance relative to the red–black tree AMR. Note that strict Z-order curve compliance is not guaranteed in this approach but the technique benefits from trying to comply to the Z-order curve as permitted in each case. The advantage of the Z-order curve becomes obvious when we adopt the third approach in which the C++ STL default hash function is used without Z-order curve enhancement. The red–black tree AMR demands more memory than the hash table approach in general. Note that we observed several cases in which the hash table approach consumed more memory than the red–black tree AMR. But it is difficult to generalize this statement because of the underhood operations that take place with the hash table approach. To demonstrate the memory consumption, we use the Valgrind [8] memory profiler. The amount of memory consumption is presented in Table 3.

Finally, we demonstrate that the red–black tree is amenable to parallelization. We note that the scalability of an AMR algorithm will not only depend on the underlying data structure, but also on the 2:1 balance algorithm, the communication patterns and load balancing. We adopt a 2:1 balance algorithm that is similar to the one presented in Burstedde et al. [5] and Isaac et al. [9]. We use a weighted Hilbert space-filling curve approach available in the Zoltan package [10] to ensure load-balanced distribution. With these features, we can achieve the strong scaling shown in Fig. 3. A sports car geometry is used for the present analysis with 18 levels of adaptation. Only the segment related to the mesh refinement is timed in the analysis. We observe parallel efficiency of more than one because of better cache utilization as the problem size per processor diminishes in a strong scaling analysis. The parallel design of the current AMR algorithm will be the subject of

a future paper and it is beyond the scope of this current short note in which we aim to demonstrate the benefits of a red-black tree in AMR.

#### 4. Conclusions

We presented the benefits of adopting a red-black tree to generate Cartesian adaptive meshes with deep levels of adaptation. Exact arrangement of mesh elements according to a Z-order curve, and therefore data locality, is preserved when a red-black tree is used in combination with a Morton encoding scheme. Data locality in this fashion is guaranteed because sorting is implicit in a red-black tree by design. Equally important, there is no limit on the maximum level of adaptation with a red-black tree. When deep mesh adaptation is necessitated by extreme-scale simulation problems, this feature becomes indispensable. A red-black tree supports deep adaptation because two instances of the `bitset` class can be compared without the need to convert to an integer index as is typically done in a hash table.

These aforementioned benefits of using red-black tree come at the expense of only a slight increase in memory usage, which is not prohibitive at all. Using the proposed red-black tree AMR algorithm, which we brand as *rebl-AMR* (pronounced as *rebel AMR*), we were able to create a computational mesh with  $\approx 1$  billion (i.e. 1,070,218,199) nodes with a mesh adaptation of 17 levels for the Stanford bunny geometry with 5,680,256 triangles. The mesh generation took 1 hour 54 minutes on a single compute-node of a Linux cluster with 64 GB of DDR3 1600 MHz memory and an Intel Xeon E5-2670 2.60 GHz processor. Both the red-black tree and the hash table AMR methods are available as open-source software [11].

#### Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 1440638. The authors thank Guralp Singh for his technical assistance.

#### References

- [1] T. Lewiner, V. Mello, A. Peixoto, S. Pesco, H. Lopes, Fast Generation of Pointerless Octree Duals, *Computer Graphics Forum*, vol. 29, Wiley Online Library, 2010, pp. 1661–1669.
- [2] G.M. Morton, A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing, International Business Machines Company, New York, 1966.
- [3] J. Baert, A. Lagae, P. Dutré, Out-of-Core Construction of Sparse Voxel Octrees, *Computer Graphics Forum*, vol. 33, Wiley Online Library, 2014, pp. 220–227.
- [4] T. Karras, Thinking parallel III, <https://devblogs.nvidia.com/parallelforall/thinking-parallel-part-iii-tree-construction-gpu/>, 2012.
- [5] C. Burstedde, L.C. Wilcox, O. Ghattas, p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees, *SIAM J. Sci. Comput.* 33 (3) (2011) 1103–1133.
- [6] B. Schneier, Red-black trees, *Dr. Dobbs's J.* 17 (4) (1992) 42–46.
- [7] L.J. Guibas, R. Sedgewick, A dichromatic framework for balanced trees, in: 19th Annual Symposium on Foundations of Computer Science, 1978, IEEE, 1978, pp. 8–21.
- [8] N. Nethercote, J. Seward, How to shadow every byte of memory used by a program, in: *Proceedings of the 3rd International Conference on Virtual Execution Environments*, ACM, 2007, pp. 65–74.
- [9] T. Isaac, C. Burstedde, O. Ghattas, Low-cost parallel algorithms for 2:1 octree balance, in: 2012 IEEE 26th International Parallel & Distributed Processing Symposium, IPDPS, IEEE, 2012, pp. 426–437.
- [10] E.G. Boman, U.V. Catalyurek, C. Chevalier, K.D. Devine, The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: partitioning, ordering, and coloring, *Sci. Program.* 20 (2) (2012) 129–150.
- [11] J.J. Hasbestan, I. Senocak, Rebl-AMR: red-black tree based adaptive mesh refinement for immersed boundary simulations, [https://github.com/GEM3D/Hash\\_vs\\_RB](https://github.com/GEM3D/Hash_vs_RB), 2017, version 0.1, <https://doi.org/10.5281/zenodo.836102>.