A Hyper Tree Grid Implementation for AMR Mesh Manipulation and Visualization in VTK*

Thierry Carrard[1], Charles Law[2], and Philippe Pébay[2]

[1] Commissariat à l'Énergie Atomique et aux Énergies Alternatives
`thierry.carrard@cea.fr`

[2] Kitware
{`charles.law,philippe.pebay`}`@kitware.com`

Summary. Adaptive Mesh Refinement (AMR) grids are particularily efficient to mitigate the tension between numerical accuracy and computational cost. The Visualization Toolkit (VTK) offers data structures and algorithms to treat 2D and 3D AMR meshes, either "patch-based" or as binary trees. This paper summarizes our extension of the former capability to the case of generic subdivision and arbitrary rectilinear geometry, using an optimized, generic tree traversal technique together with a fast dual grid construction.

1 Introduction

At the DIF center of *Commissariat à l'Énergie Atomique et aux Énergies Alternatives* (CEA), massive numerical simulations are routinely run on petascale supercomputers such as Tera100 [1]. Consequently, a dedicated parallel visualization tool (Love [3]) based on VTK [4] and ParaView [6] was developed at CEA/DIF. Among supported CEA simulation codes, Adaptive Mesh Refinement (AMR) ones such as Hera [5], a tree-based AMR hydrodynamics simulation code, are of particular interest to track fine details in a large domain. The goal of the work presented hereafter was thus to add support in VTK for generic tree-based AMR data sets.

AMR enables a trade-off between numerical accuracy and computational cost, by refining the mesh only in regions where criteria based on error estimation. Since the first description of an AMR methodology with the Berger-Oliger [2] type, several implementations have been proposed and developed, such as [5]. Our work was aimed at supporting tree-based AMR grids, which have a lower memory footprint at the cost of more complex processing algorithms. Moreover, when AMR meshes are to be used as the input of an unstructured grid algorithm such as iso-contouring, topological irregularities

*This work was supported by CEA, Direction des Applications Militaires Île-de-France (DIF), Bruyères-le-Châtel, 91297 Arpajon, France.

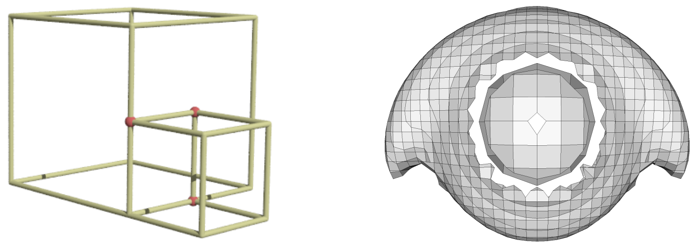

Fig. 1. Example of partly connected vertices (in red) and the corresponding cracks appearing after a contour algorithm is applied

resulting in strong visual artifacts, e.g., cracks, may appear. These are caused by the very nature of AMR meshes where neighboring cells at different refinement levels result in partly connected vertices ("T-junctions"). Linear interpolation, commonly used by visualization algorithms, produces discontinuities across T-Junctions, which in turn result incorrect outputs surface and thus visual artifacts, as shown in Figure 1.

VTK is an open-source, C++ toolkit used by thousands of users, which supports many data types and hundreds of algorithms. Prior to this work VTK mostly supported block-structured AMR data sets, with only limited support for binary tree-based AMR based on an octree refinement scheme [7]. However, simulation codes such as Hera can use ternary subdivisions, and start from an arbitrary rectilinear grid at the root level, and neither of these features was supported. These simulations could thus not be post-processed with VTK. This paper henceforth describes the design and implementation of the novel `vtkHyperTreeDataSet` which tackles both geometric (arbitrary rectilinear root cells) and topological (genericity of the subdivision) issues.

2 Method

To avoid errors such as those described in §1, two distinct approaches are possible: either implement specialized algorithms for tree-based AMR grids, or transform the input to a conforming unstructured grid, allowing for reuse of existing algorithms. We chose the latter approach using a dual grid construction, upon which all filters designed for vertex-centered attributes can natively operate and produce correct results.

Our dual grid transformation for AMR grids is easy to understand: a dual vertex is defined in the center of every leaf in the tree. In 2D (resp. 3D), two dual vertices are connected by a dual edge if and only if the corresponding leaf cells share an edge (resp. a face) or a portion thereof. Dual cells are defined accordingly. Furthermore, our construction displaces dual boundary entities onto the primal boundary, in order to avoid near-boundary artifacts when applying visualization algorithms to the dual, as shown in Figure 2.

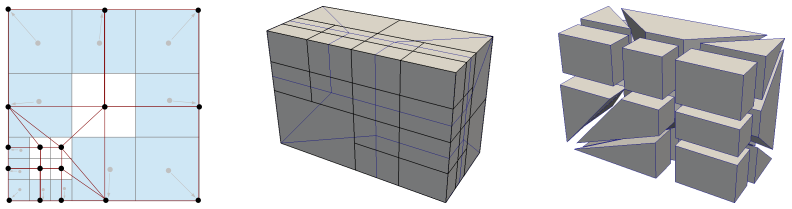
Fig. 2. Left: a 2D ternary tree-based AMR grid, overlaid with its dual. Center: a 3D binary tree-based AMR grid overlaid with its dual. Right: dual cells shrunk.

Since the dual grid is well behaved and has no T-junctions, attributes can be interpolated over the dual mesh without creating discontinuities. Moreover, the dual grid provides an efficient solution to extending existing visualization algorithms to AMR data by implicitly converting cell attributes into vertex attributes. This avoids explicit re-sampling of attribute fields which is expensive computationally and in memory, and tends to average and smooth fields.

In order to support the widest variety of visualization algorithms, our data model has a double API, as some visualization filters work best processing dual grid cells, whereas others execute most efficiently accessing the primal tree. Filters also have the option of using an iterator to loop through cells, or using a random access API to retrieve cells by an index. Although our implementation supports random cell access as required by the VTK API, we strongly discourage its use, which triggers a conversion to a fully described unstructured grid, canceling the benefits of tree-based storage. As an alternative, the data structure provides an iterator that traverses dual cell with a depth first search. Such traversal of trees is efficient but does not provide spatial neighborhood information that many visualization operations require. As a consequence, the ability to provide surrounding cells of a central leaf while traversing the tree is crucial.

To provide neighborhood information we devised and implemented a compound iterator that tracks a neighborhood of cursors while traversing the tree, using a 3^3 grid of cursors. The center cursor simply follows a depth first search of the nodes, while the surrounding 26 point to its adjacent nodes. During tree search, a pre-computed traversal look-up table tells for each child being visited, how to populate the new grid of cursors. If a neighboring node is a leaf, its corresponding cursor value does not change. Multiple neighborhood cursors may end up pointing to the same leaf. The cursor is initiated using a region of the top level grid of roots. On the boundaries of the data set, some cursors will be set to null values, which are used to detect boundaries that need to be handled with special processing. Maintaining a neighborhood of cursors does not impact the efficiency of the depth first search much, but solves the need for easy access to immediately adjacent leaves.

A dual grid is generally more complex than the primal tree: e.g., 3D dual grids contain pyramidal and wedge cells in addition to hexahedral cells. In order to handle this difficulty, we first assign ownership of dual vertices to a single leaf, which allows for a depth first search to process vertices without duplication. Out of the 8 leaves that potentially touch a vertex in 3D (resp. 4 in 2D), the deepest leaf is assigned ownership, breaking ties by choosing the leaf with the smallest leaf index. In addition, we use degenerate hexahedra (resp. quadrilaterals) to represent all dual grid cells, as the compound iterator naturally supplies the 8 (resp. 4) surrounding leaves for each a vertex. Cases where cursors point to the same leaves are ignored. The only extra processing required is to filter out degenerate cells created by subsequent visualization algorithms: e.g., iso-contouring the dual grid will produce some degenerate triangles that are then removed to clean the resulting iso-surface mesh.

3 Results

The first noticeable result is the availability in VTK of a novel data object for non-uniform rectilinear, tree-based AMR data sets with generic subdivision, the first and only publicly available such capability, in our knowledge.

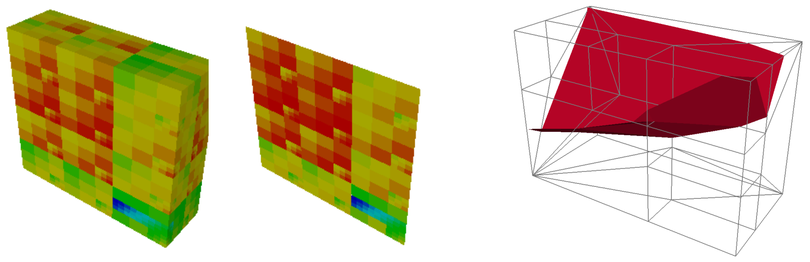
Fig. 3. Left: a 3D ternary tree-based AMR grid with 3x4x2 root cells, non-uniform geometry, and various levels of subdivisions, direction) planar cut thereof. Right: iso-contour across a 3D binary tree-based AMR grid overlaid over the edges of the wireframe representation of the dual mesh.

Figure 3, left, shows 3D AMR grid with 3x4x2 root cells, and non-uniform geometry in each direction, generated with various levels of ternary subdivisions within a VTK pipeline: to the left is shown a surface rendering of the object, using the specialized outside geometry extraction filter which we developed as well. Note that this data object currently only supports cell-based data, which in this case were generated using a combination of Euclidean distance to center with local index-based noise. Together with this `vtkHyperTreeGrid` instance is shown its intersection with a plane normal to the z-axis, computed by of another filter specifically devised for that type of input. Note that both specialized filters produce simpler VTK types and can thus be directly hooked downstream to standard VTK pipelines, for subsequent analysis, rendering, and interaction.

In §1, we discussed with iso-contouring the typical difficulties that AMR meshes pose to visualization algorithm. Iso-contouring thus provides a typical test case to demonstrate the advantages of our approach based on duality: Figure 3, right, shows the output of an iso-contouring algorithm taking as input the dual grid of the `vtkHyperTreeGrid` instance shown in Figure 2. In that case, although the primal grid does indeed contain a number of T-junctions, the computed iso-contour does not have cracks. The same approach works generically for both binary and ternary subdivision, and for an arbitrary number of hyper tree root cells with non-uniform geometry along each axis.

4 Conclusion

We presented a new generic hyper tree grid object for binary and ternary AMR non-uniform rectilinear grids. Our implementation took care of lowering the memory impact and keep a high computational efficiency, thus rendering processing of generic tree-based AMR meshes practicable for common visualization needs. This new data object, along with the specialized algorithms we discussed here, and some test programs, are all part of VTK. Please refer to `http://www.vtk.org/Wiki/VTK` for details on obtaining and building VTK.

In addition, we created a tree-based AMR grid generator to exercise and illustrate the properties of this data object, which can also be used for the prototyping and development of additional specialized filters. In can be found in the `Examples/HyperTree/Cxx` directory of VTK, and has multiple options allowing for the creation of synthetic `vtkHyperTreeGrid` instances with arbitrary size, depth, and subdivision scheme.

References

1. CEA's Tera supercomputer. `http://www-hpc.cea.fr/en/complexe/tera.htm`.
2. M.J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *J. Comput. Phys.*, 53(3):484–512, 1984.
3. D. Aguilera *et al.* Parallel software and hardware for capability visualization of HPC results. *ASTRONUM*, 2007.
4. L. Avila *et al. The VTK User's Guide*. Kitware, Inc., eleventh edition, 2010. ISBN 978-1-930934-23-8. `http://www.kitware.com/products/books/vtkguide.html`.
5. H. Jourdren. Hera: A hydrodynamic amr platform for multi-physics simulations. In *Adaptive Mesh Refinement Theory and Application*, volume 41 of *LNCSE*, pages 283–294. Springer, 2005.
6. A. Squillacote. *The ParaView Guide: A Parallel Visualization Application*. Kitware Inc., 2007. ISBN 1-930934-21-1, `http://www.paraview.org`.
7. M.-M. Yau and S. N. Srihari. A hierarchical data structure for multidimensional digital images. *Communications of the ACM*, 26(7):504–515, July 1983.