Highly Efficient Controlled Hierarchical Data Reduction techniques for Interactive Visualization of Massive Simulation Data

Jérôme Dubois and Jacques-Bernard Lekien

CEA, DAM, DIF, F-91297 Arpajon, France

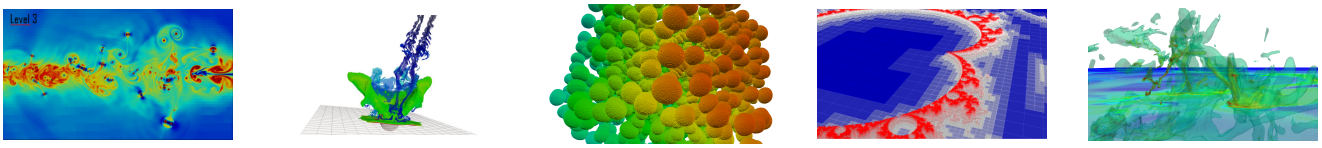

Figure 1: *Enabling new interactions with Tree-Based Adaptive Mesh Refinement techniques, illustrated with our HyperTreeGrid implementation in VTK. From left to right, 2D 100 million cells Shockwave data and 3D 300 million cells asteroid fall simulation result explored smoothly on a laptop, 3D 1 billion cells bubbles expansion simulation, 2D 6 billion cells Mandelbrot calculation result and 3D 72 billion cells astrophysics simulation data visualized interactively respectively on one, two and sixteen 32-cores, 128 GBytes memory nodes of the Tera-1000-1 supercomputer.*

Abstract

With the constant increase in compute power of supercomputers, high performance computing simulations are producing higher fidelity results and possibly massive amounts of data. To keep visualization of such results interactive, existing techniques such as Adaptive Mesh Refinement (AMR) can be of use. In particular, Tree-Based AMR methods (TB-AMR) are widespread in simulations and are becoming more present in general purpose visualization pipelines such as VTK. In this work, we show how TB-AMR data structures could lead to more efficient exploration of massive data sets in the Exascale era. We discuss how algorithms (filters) should be designed to take advantage of tree-like data structures for both data filtering or rendering. By introducing controlled hierarchical data reduction we greatly reduce the processing time for existing algorithms, sometimes with no visual impact, and drastically decrease exploration time for analysts. Also thanks to the techniques and implementations we propose, visualization of very large data is made possible on very constrained resources. These ideas are illustrated on million to billion-scale native TB-AMR or resampled meshes, with the HyperTreeGrid object and associated filters we have recently optimized and made available in the Visualisation Toolkit (VTK) for use by the scientific community.

CCS Concepts

• ***General and reference*** → ***Design; Performance;*** *Evaluation;* • ***Human-centered computing*** → ***Visualization;*** • ***Software and its engineering*** → *Designing software;* • ***Information systems*** → *Data compression;*

1. Introduction

In high performance computing, different kinds of meshes can be used to represent simulation data, each having their own advantages and drawbacks [FG08]. Adaptive Mesh Refinement (AMR) techniques mitigates between memory for mesh description and simulation properties storage by refining where phenomena occur, in a semi-structured way. Such meshes can be block-based [BC89] or tree-based [BO84]. Tree-based AMR representation (TB-AMR) allows to exploit very large data, and has the extra capability to make the high fidelity resampling of existing non AMR data sets possible while keeping memory footprint controlled.

In this paper, we give some insights on how TB-AMR meshes and adapted smart visualization filters design can help make the exploration process smooth and efficient to explore massive data sets interactively. We combine these ideas with their hierarchical properties to optimize further the exploratory visualization process. Implemented in the Visualization Toolkit (VTK) [Moo98], our contributions enable high savings in time for the analysts, high efficiency on large resources for visualization, and even make it possible to fit a Petascale class data set onto the limited resources found in a small-factor computer such as a laptop. We conclude that the techniques we proposed and tested can be the foundations for enabling the visualization of future Exascale class simulations.

DOI: 10.2312/evs.20191167

2. Tree-Based AMR for visualization

In this section, we briefly recall the key concepts behind TB-AMR meshes and their VTK implementation: the HyperTreeGrid.

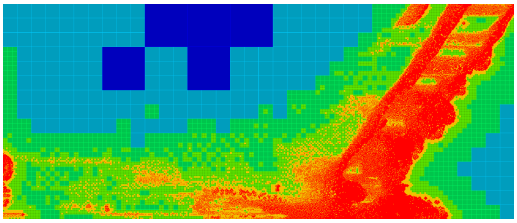

Figure 2: *Tree-Based AMR visualization of a 2D shockwave simulation, with refinement factor of 3, thus having 9 children for each coarse cell. Colors represent tree levels, from blue root unrefined cells to most refined red cells in this visualization.*

2.1. Tree-Based AMR data structure

A TB-AMR data structure is generally represented as a tree, where each node can be refined by a fixed factor where details are needed. One example is the octree [Mea82]. The concept of a grid of trees [NSLD99], illustrated in Figure 2, can be introduced to offer for instance some load-balancing optimizations.
In VTK, this is instantiated with the HyperTreeGrid (HTG), a data object representing a rectilinear grid of trees (HyperTree). HTG supports arbitrary uniform refinement, generally 2 or 3 and coarse nodes are always completely refined. The data structure is thus very regular, and only stores the index (pointer) to the elder child for each coarse node. A (bit)mask attached to the HTG is introduced to mask children nodes not being part of the mesh. In section 5, we will see that this design choice makes interesting optimizations possible. Other approaches such as those of [LK11, KSA13] exist to implement an octree, where a bitmask and pointers restricted to the actual children are present in each node. Declaration of HTG leaf nodes can be avoided by smartly build the trees, thus resulting in a drastic reduction of the memory footprint. Interested readers can turn to [HLP17b, HLP17a] for the HTG development foundation.

2.2. Operating on TB-AMR meshes and filter design

For visualization and data analysis of the simulation results, algorithms are applied to extract information of interest. This process is often referenced as filtering, and thus algorithms are generally called filters. It is crucial to take advantage of intrinsic hierarchical properties of AMR meshes in filters. To make the design of algorithms easier and consolidate code development, iterators can be designed to handle trees traversal and respect TB-AMR features. The mesh can then be seen as a container operated on by an algorithm with the use of iterators, such as in the C++ Standard Template Library design [Err00]. In the frame of VTK HyperTreeGrid, we introduce special iterators called cursors and supercursors. It is mandatory to use them to respect HTG-related information like *bitmask* or *Depth-Limiter* during traversal. Both are used to traverse a single tree and supercursors are neighborhood-aware. They can be oriented or non-oriented and thus respectively allow only to descend or move up and down in the tree. Depending on their complexity, HTG traversal time can be 50x longer.
Therefore, when designing HTG-aware filters one should pay attention to instantiate the proper (super)cursor(s) to avoid the use of an unnecessary costly iterator. Also, one should take advantage of the HTG features like the bitmask to optimize filters by virtually pruning trees. This is illustrated in the next section.
TB-AMR data structures generally have cells of different levels, thus introducing the concept of T-Junctions. Filters such as Contour must be written carefully to handle these, otherwise artifacts can appear. Solutions to address problems such as T-Junctions or crack-free surfaces are presented in [HLP17b, HLP17a], which lay down the foundations of our VTK HTG design. Regarding T-Junction, we proposed to generate the dual unstructured mesh of the primal AMR-mesh on the fly using the (super)cursors. It is beyond the scope of this paper to develop this technique, which has the nice property of putting "primal" AMR cell-centered attributes to the vertices of the dual mesh, thus providing exact crack-free Contour.

3. Hierarchy-aware algorithms

We focus now on two of the filters we implemented to illustrate how we can drastically reduce the memory footprint and offer acceleration. Next, we consider how such data structures offer great opportunities to accelerate the rendering process of very large scenes.

3.1. Optimized filtering

A filter takes input data and generates output filtered data. Input and output data can be of same nature, like the Threshold filter implementation we provide in VTK. On the contrary they can be of different nature, like for the Contour filter we created in VTK, which takes an HTG and generates an unstructured mesh.
The Threshold filter extracts cells of interest based on user provided thresholds (min/max) and a selected simulation field array. The output of the Threshold filter can be a new mesh with its new field arrays. At worst, the memory footprint would thus double with a naive approach when output mesh is equal to input mesh. For the visualization of very large simulation data this might be problematic as the memory resource per processing element is constrained. To be more efficient memory-wise, we propose to shallow copy the input grid of trees to generate an identical output grid at almost no memory cost. The field array being attached to the grid, it is also shallow-copied in the process. Then we attach a new bitmask to the output grid to keep filtered cells only. The added memory cost for thresholding is very limited with our approach.
Contour filter, illustrated in Figure 3, creates an isosurface or isoline based on a user provided isovalue and a selected field array. How to compute the isosurface is beyond the scope of this paper and interested readers can turn to [Mas56] for some pointers. It is enough to know that neighborhood information is required when traversing the mesh to generate the isosurface. Consequently, one would logically want to use a complex supercusor and traverse the grid of trees. Due to the cost of this complex supercursor and the nature of Contour filter, we have considered to introduce a two-times traversal algorithm. First traversal is made with a cheap cursor (relative cost 1) and a new bitmask is created to virtually prune

the trees. Second traversal with a more complex supercursor (relative cost 50) is done on the pruned trees to handle calculations. We tested this approach on a variety of data sets and we observed speed-ups of up to 10x with no penalties. These results greatly depend on how effectively pruned the trees can be, therefore compensating the two-times traversal process.

3.2. Optimized rendering

Rendering can also benefit from both the geometry exposed by the grid of trees and the per level refinement of each tree. We propose to use rendering information such as the camera focal point and the screen resolution (pixel size) in the rendering process. By knowing where the user is looking at, one can easily select the trees or nodes actually involved in the rendering. It is then possible to avoid traversal of nodes outside the screen. One can also stop traversing a tree as soon as information gets subpixel. For very deep trees, our approach drastically reduces the rendering time. Furthermore, sometimes a whole subtree has the same values to render. It is then useless to generate the finer cells as one can create a single larger cell representing it. This helps optimize the size of the actual output used for rendering and provide acceleration. To do so we implemented a two-times traversal scheme. In a first traversal subtrees are identified and tagged. Then the actual optimized output for rendering is generated by traversing a second time the tagged subtrees solely. In section 5, our implementation offers 10x to 1000x higher framerates and a decrease in memory footprint of 13x.

4. Controlled hierarchical data reduction

Generally, simulation codes and visualization tools do not present the same execution profiles. the former can be highly compute intensive while the latter might be more memory intensive. Thus depending on the visualization needed, full simulation resolution might either (1) not be of interest to produce general far-away pictures or (2) not be possible on the visualization-dedicated resources. Such hardware might be a small visualization cluster, workstations or even handheld devices with very limited memory.
For visualization purpose it might be of interest to depth-limit the trees. At most, removing $levels_{removed}$ levels of the TB-AMR mesh removes $f^{d*levels_{removed}}$ nodes, i.e. $8^{levels_{removed}}$ for a 3D octree. Depending on the trees depth and desired treatments or pictures, removing several levels might not be noticeable and would greatly decrease storage requirements and execution times. Analysts can choose the resolution required at any time based on their expertise and thus greatly accelerate the exploration process.
Also, when TB-AMR is represented as a grid of trees, it is trivial to remove large mesh portions by limiting spatial boundaries to the trees participating in a smaller volume. Memory footprint will be drastically reduced and filtering or rendering process greatly accelerated. This approach has been used to optimize rendering in previous section and can be extended to filters.

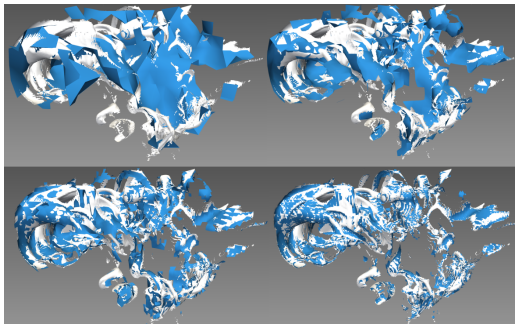

Figure 3: *Depth-limiting the water splash Contour of YA31. From left to right, top to bottom, level 0 to 3 depth-limited blue Contour is shown. White Contour is the ground truth using maximum level of 6. Level 4 depth-limited Contour (not shown here) shows no difference from ground truth, from this viewpoint.*

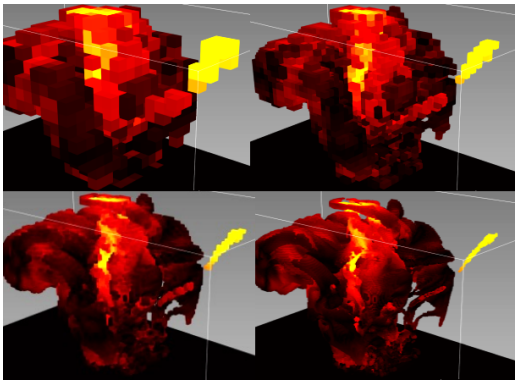

Figure 4: *Depth-limiting thresholding of YA31 water splash to level 0 to 3, from left to right and top to bottom. Until level 2, when threshold values are changed result is instantaneous.*

4.1. Enabling interactive filtering

Analysts might want to depth-limit the filtering process because they are in an exploratory phase. We implemented a HTG depth-limiting filter to offer this possibility. HTG is shallow-copied and the HTG depth-limiter scalar is positioned appropriately and so applying this filter is inexpensive. This strategy leads to impressive results, such as a real-time direct volume rendering of a large dataset as seen in [DHL18], or the possibility to interactively test the isovalue of Contour filter several times per second before actually applying Contour on the full-depth trees. This also enables the user to try more camera viewpoints, more values for interactive filters, or even identify hotspots more easily as values put on coarse cells can be driven. Coarse values can come from the simulation results, and we implemented a HTG coarse cell evaluation filter to assign computed values to them, such as maximum or minimum of children values, to help identify hotspots more rapidly for instance. To be noted, input HTG is full resolution and so related memory consumption is not impacted by depth-limitation. Only output generated afterwards will benefit from it in terms of memory.

4.2. Reducing storage

We propose to apply this depth-limiting strategy for reading and storing HTG as well. Therefore, loading and HTG building times

will be reduced and further filters execution will be sped-up. To be noted, overall memory consumption will be decreased because the actual instantiated mesh resolution is lower. This strategy is applied to the storing step as well. If the full resolution mesh and corresponding simulation fields have been loaded and some filtering applied, then the resulting mesh can be reduced. Depth-limiting the stored trees will also reduce future loading times from storage and data exploration at the expense of a controlled reduced resolution.

5. Results

We evaluate the presented techniques with the following simulation data represented in Figure 1: (1) Shock100, 2D 100 million cells shockwave simulation; (2) YA31, 3D 300 million cells, simulated asteroid fall [PR17]; (3) S1, 3D 1 billion cells, structured bubbles expansion; (4) Mandel6, 2D 6 billion cells, Mandelbrot calculations [Hol87]; (5) Astro72, 3D 72 billion cells, astrophysics simulation. YA31 and S1 illustrate performance of meshes we resampled to HTG. In [DHL18], YA31 is explored on one core of a supercomputer node or laptop whereas original unstructured visualization required hundreds. In this paper, we do not expose how YA31 and S1 were resampled, but suffice it to say that we keep the full simulation resolution. Astro72 is a very large dataset, requiring TeraBytes of memory for complex visualization pipelines. With usual techniques it is only exploitable on multiple nodes of a cluster. Shock100 and Mandel6 are 2D meshes illustrating optimized TB-AMR rendering. Experiments were conducted on either a Dell Latitude 7480, Core i5-6300U, 16 GigaBytes memory and 512 GigaBytes SSD or up to 16 nodes of the Tera-1000-1 supercomputer [Ter17], 2* Xeon E5-2698v3 16C 2.3GHz and 128 GigaBytes memory per node and Peta-class Lustre storage. Results are executed on appropriate hardware depending on each dataset memory consumption. For Astro72, the largest dataset, we use 16 Tera-1000-1 nodes, and for the smaller datasets or those reduced with our techniques we can use less nodes or even just one core of the laptop. Table 1 shows a reduction of execution times from minutes to less than a second for different filters and volume rendering on a single timestep of YA31. Pictures corresponding to the first levels are presented in Figures 3 and 4. To be noted, loading time from storage remains constant at about 30 seconds because we load the full-depth trees. Loading times of depth-limited data

Table 1: *Execution times (seconds) achieved for depth-limited Contour, Threshold and volume rendering (splatting) for YA31 on one timestep. Speed-up is tightly correlated to the amount of cells.*

Depth limit	0	1	2	3	4	5	6
Contour	0.1	0.3	1.7	4	13	30	117
Thresh.	0.1	0.2	0.5	2	9	25	105
Volume	<1	1.5	2	7	20	64	285
Cells (10e6)	0.008	0.75	0.6	5	18	64	306

sets are shown in Table 2. Gains obtained allow analysts to load different timesteps of ensemble data faster to identify specific moments of interest. In YA31, the asteroid strikes the water after a dozen timesteps, so one can identify this moment in seconds with depth-limited loaded timesteps instead of minutes at best. The water splash created by the impact reaches its maximum height several

Table 2: *Loading time in seconds and space savings when depth-limiting the trees. For each data set, each load is realized with the same fixed resources for the sake of comparison. In the first column, dataset name and maximum level are provided.*

Depth limit	L0-3	L4-L_{max-1}	L_{max}
Shock100 ; 7	<0.5 ; <5MB	0.5-1 ; 32-221MB	2 ; 1.3GB
S1 ; 7	<1 ; <40MB	<10 ; 0.2-8.8GB	<60 ; 67GB
Mandel6 ; 10	4-7 ; <100MB	7-20 ; 227MB-54GB	30 ; 144GB
Astro72 ; 12	<6 ; <2.4GB	9-240 ; 20GB-2TB	240 ; 2TB
YA31 ; 6	<1 ; <100MB	<5 ; 405MB-1.5GB	20 ; 6.9GB

hundred timesteps later. It can be retrieved in minutes interactively with depth-limitation during loading. To be noted, storage gains are directly related to the number of cells refined for each levels. For Astro72, last levels refine few cells and so storage gains are limited. We are able to easily control resolution of Astro72 and make this large data set displayable on a laptop for collaborative purposes.
Table 3 presents optimized rendering performance. For the moderate Shock100 data set, interactivity evolves from stuttered manipulation to a smooth experience. The billion scale Mandel6 is only exploitable with the optimized rendering we introduced, as standard approach provides 1 frame every 20 seconds and consumes dozens of GigaBytes of video memory. As depth-limiting to sub-pixel information is involved, performance is sensitive to the display configuration: reducing the rendering window size results in higher performance, linearly. Our optimized rendering also frees up video memory, occupying 13x less for Mandel6. More video memory is then available for GP-GPU capable visualization pipelines.

Table 3: *Framerates and video memory occupancy of 2D optimized and naive rendering. Screen resolution is 3440x1440.*

	Optimized Rendering		Standard Rendering	
Shock100	8-20 fps	1 GB	2 fps	4 GB
Mandel6	1-5 fps	3 GB	0.05 fps	40 GB

6. Conclusion and future work

In this paper, we reflected on the design of Tree-based AMR meshes and optimized filters or rendering and introduced optimization of the visualization process with controlled hierarchical data reduction. We implemented these concepts in the HyperTreeGrid object we contributed in VTK and provided associated filters and rendering capabilities. Using HyperTreeGrid, we illustrated that analysts can greatly reduce their processing time with controlled reduction, achieving speed-ups of 10x to 1000x and corresponding decrease in memory footprint. Our implementation also leads to very efficient interactive visualization at the billion-scale. Controlled space savings are also enabled when loading or storing simulation results. The techniques we exposed can serve as building blocks for analyzing *in-situ* Exascale simulation results, like in [CML18]. In the future, we aim at further improving load-balancing of TB-AMR meshes, and explore visualization pipeline optimization for shared memory or GPUs.

References

[BC89] BERGER M., COLELLA P.: Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics 82*, 1 (1989), 64 – 84. URL: http://www.sciencedirect.com/science/article/pii/0021999189900351, doi:https://doi.org/10.1016/0021-9991(89)90035-1. 1

[BO84] BERGER M. J., OLIGER J.: Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics 53*, 3 (1984), 484 – 512. URL: http://www.sciencedirect.com/science/article/pii/0021999184900731, doi:https://doi.org/10.1016/0021-9991(84)90073-1. 1

[CML18] CAPUL J., MORAIS S., LEKIEN J.-B.: Padawan: A python infrastructure for loosely coupled in situ workflows. In *Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization* (New York, NY, USA, 2018), ISAV '18, ACM, pp. 7–12. URL: http://doi.acm.org/10.1145/3281464.3281470, doi:10.1145/3281464.3281470. 4

[DHL18] DUBOIS J., HAREL G., LEKIEN J.-B.: Interactive visualization and analysis of high resolution hpc simulation data on a laptop with vtk. IEEE VIS'18 SciVis Contest. 3, 4

[Err00] ERRICOLO D.: C++, a better language for engineering applications. *IEEE Antennas and Propagation Magazine 42*, 4 (Aug 2000), 95–101. doi:10.1109/74.868057. 2

[FG08] FREY P., GEORGE P.-L.: *Mesh generation*, 2 ed. John Wiley & Sons, 2008. 1

[HLP17a] HAREL G., LEKIEN J., PÉBAŸ P. P.: Two new contributions to the visualization of AMR grids: I. interactive rendering of extreme-scale 2-dimensional grids II. novel selection filters in arbitrary dimension. *CoRR abs/1703.00212* (2017). URL: http://arxiv.org/abs/1703.00212, arXiv:1703.00212. 2

[HLP17b] HAREL G., LEKIEN J., PÉBAŸ P. P.: Visualization and analysis of large-scale, tree-based, adaptive mesh refinement simulations with arbitrary rectilinear geometry. *CoRR abs/1702.04852* (2017). URL: http://arxiv.org/abs/1702.04852, arXiv:1702.04852. 2

[Hol87] HOLMES P.: An introduction to chaotic dynamical systems (robert l. devaney); nonlinear dynamics and chaos: Geometrical methods for engineers and scientists (j. m. t. thompson and h. b. stewart) (with the assistance of r. ghaffari and c. franciosi and a contribution by h. l. swinney). *SIAM Review 29*, 4 (1987), 654–658. URL: https://doi.org/10.1137/1029136, arXiv:https://doi.org/10.1137/1029136, doi:10.1137/1029136. 4

[KSA13] KÄMPE V., SINTORN E., ASSARSSON U.: High resolution sparse voxel dags. *ACM Trans. Graph. 32*, 4 (July 2013), 101:1–101:13. URL: http://doi.acm.org/10.1145/2461912.2462024, doi:10.1145/2461912.2462024. 2

[LK11] LAINE S., KARRAS T.: Efficient sparse voxel octrees. *IEEE Transactions on Visualization and Computer Graphics 17*, 8 (Aug 2011), 1048–1059. doi:10.1109/TVCG.2010.240. 2

[Mas56] MASON R. M.: The digital approximation of contours. *J. ACM 3*, 4 (Oct. 1956), 355–359. URL: http://doi.acm.org/10.1145/320843.320854, doi:10.1145/320843.320854. 2

[Mea82] MEAGHER D.: Geometric modeling using octree encoding. *Computer Graphics and Image Processing 19*, 2 (1982), 129 – 147. URL: http://www.sciencedirect.com/science/article/pii/0146664X82901046, doi:https://doi.org/10.1016/0146-664X(82)90104-6. 2

[Moo98] MOORE J. C.: Visualizing with vtk. *Linux J. 1998*, 53es (Sept. 1998). URL: http://dl.acm.org/citation.cfm?id=327469.327474. 1

[NSLD99] NORMAN M. L., SHALF J., LEVY S., DAUES G.: Diving deep: data-management and visualization strategies for adaptive mesh refinement simulations. *Computing in Science Engineering 1*, 4 (July 1999), 36–47. doi:10.1109/5992.774839. 2

[PR17] PATCHETT J. M., ROSS G.: Deep water impact ensemble data set. URL: https://sciviscontest2018.org/wp-content/uploads/sites/19/2017/09/DeepWaterImpactEnsembleDataSet_Revision1.pdf. 4

[Ter17] Tera-1000-1. URL: https://www.top500.org/system/178790. 4