# Additional Material - Evaluation of Mesh Compression and GPU Ray Casting for Tree Based AMR data in VTK

Antoine Roche[1,2,3] and Jérôme Dubois[1,3]

[1]Université Paris-Saclay, CEA, Laboratoire en Informatique Haute Performance pour le Calcul et la simulation, 91680 Bruyères-le-Châtel, France
[2]MSc Student, Université de Reims Champagne-Ardennes, France
[3]CEA, DAM, DIF, F-91297 Arpajon, France

## 1. Additional Information

In this addendum to our poster paper, we add three elements: 1) experiments conducted on surface models issued from the Computer Graphics field, 2) Sphere volume model results, and 3) Some numbers on depth-limitation using a HTG feature and SVDAG rendering. Additionally, Figure 8 shows the unfiltered display of YA31 Asteroid data set.

**Surface models.** Overall, HTG does not behave that well to represent and render surface models, and SVDAG performs extremely well. Such situation can be encountered either when 1) model is surface-like to begin with, 2) after VTK filtering we obtain a resulting surface-like mesh. In the case of a HTG representation of such mesh, memory footprint can be very large, especially when rendering it with native surface representation as an unstructured mesh. Using techniques from SVDAG directly inside HTG could greatly help improve such SciVis situations. All tables, and Figures 1 to 6 are related to this item.

**Sphere model.** Sphere model is a synthetic model displaying 7 out of 8 octants of a sphere. It simply illustrates leaf cells of different levels, and shows the weakness of the provided single SVDAG implementation which generates large amounts of attribute (color) values for blue voxels in Figure 6. Our multi-SVDAG approach fixes this, with memory occupancy on par with HTG for the attribute array. We are convinced additional work could be conducted directly in the provided SVDAG implementation to correct this as well, and would provide results similar to the multi-SVDAG. Either way, the compression ratios and fast rendering are very convincing, and a direct implementation inside the VTK HyperTreeGrid could be very beneficial for the SciVis community.

**Depth-limitation.** VTK HTG reader and writer have a depth limitation capability. It is possible to prune the trees of a model virtually, thus decreasing grid resolution. This can be useful for previewing or applying filters faster on limited hardware, in exchange of a controlled loss of precision. Data can still be reloaded at full resolution afterwards, updating the whole VTK pipeline automatically and implying heavier computations as well as a larger memory footprint. We illustrate the depth limitation process with Fuel Assembly, and then the rendering as a SVDAG. Results are similar with the YA31 Asteroid data set, where loading data is decreased from 110s at full resolution (level 7) to 20s at level 6, 5s at level 5 and 1s at level 4, while still providing a decent overview of the data. HTG management of coarse nodes values is important to allow this efficiently, and combining depth limitation to SVDAG rendering offers nice improvement for the SciVis end-user. Figure 7 illustrates the depth-limitation process.

**Table 1:** *Amount of voxels and memory ratio of surface models voxelized as a SVDAG or HTG in a $1024^3$ voxel grid, compared to the original unstructured mesh. For Bunny, Dragon and Hairball, HTG has a larger memory footprint than the unstructured mesh. This is due to 1) HTG defining all children nodes when subdividing, and associating a color to all of them (even masked) ; 2) rendering of HTG is done as an unstructured mesh. Memory footprint of HTG voxelization highly depends on the shape of the model. SVDAG offers very nice performance and we think that applying a similar strategy to HTG could help improve user experience in VTK for surface-like TB-AMR meshes.*

| | Bunny | Dragon | Lucy | Hairball |
|---|---|---|---|---|
| HTG | 10M ; 0.1x | 7M ; 0.4x | 4M ; 5.8x | 100M ; 0.5x |
| SVDAG | 400K ; 2.3x | 300K ; 34x | 2M ; 393x | 5M ; 41x |

**Table 2:** *HTG voxel count depending on the grid resolution for the Lucy surface model and SVDAG compression ratio.*

| Grid | 32 | 64 | 128 | 256 | 512 | 1024 | 2048 |
|---|---|---|---|---|---|---|---|
| HTG | 10K | 40K | 200K | 700K | 3M | 10M | 50M |
| SVDAG | 13x | 14x | 15x | 16x | 19x | 22x | 25x |

© 2020 The Author(s)
Eurographics Proceedings © 2020 The Eurographics Association.

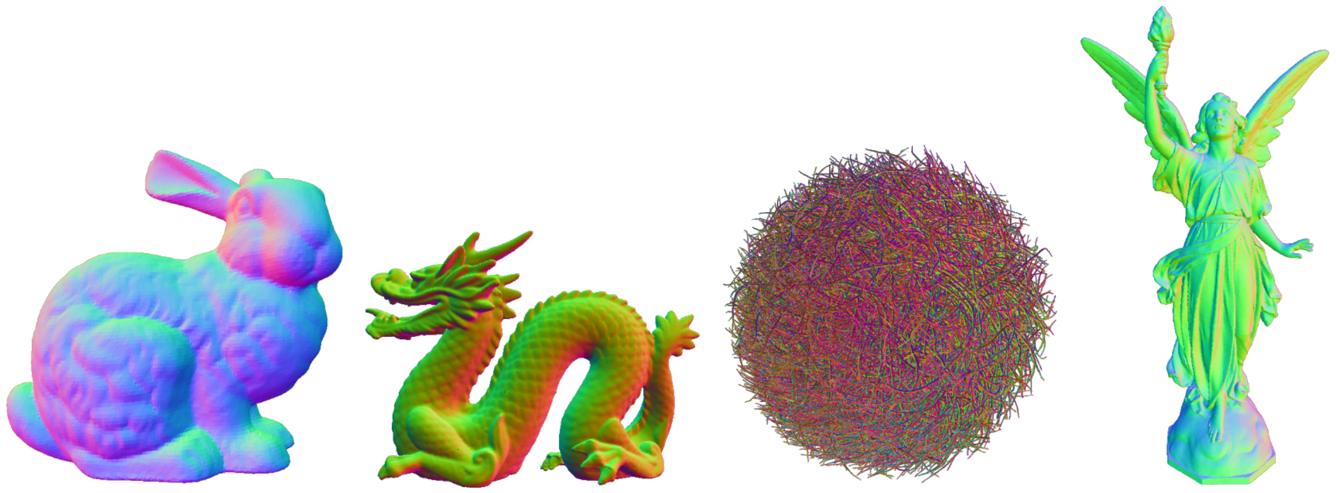

**Figure 1:** *Surface models voxelized as HTG and rendered as SVDAG. From left to right: Bunny, Dragon, Hairball and Lucy.*

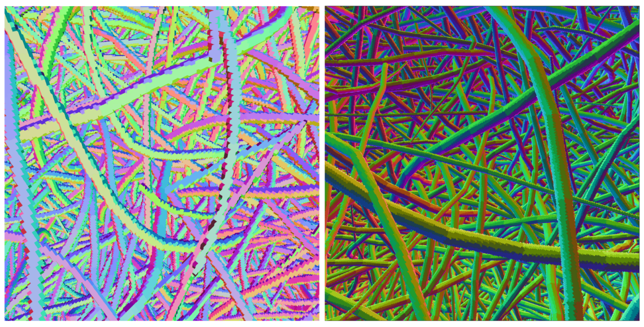

**Figure 2:** *Hairball voxelized as HTG and rendered as SVDAG, in grid resolutions of 1K and 4K. Native HTG rendering was impossible with a grid of 4K on a 8GB GPU because of the large unstructured mesh generated for surface rendering. SVDAG rendering made it possible.*

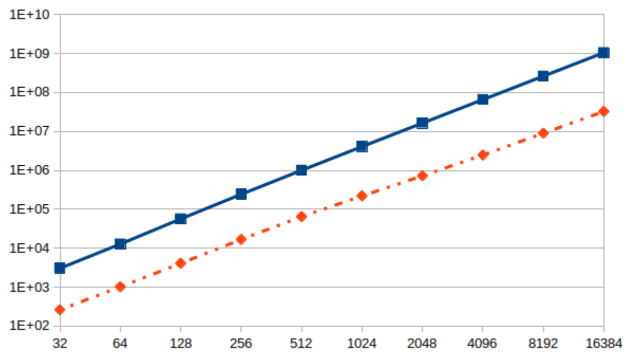

**Figure 3:** *Number of vertices for the HTG and SVDAG representation of Lucy model at different grid resolutions.*

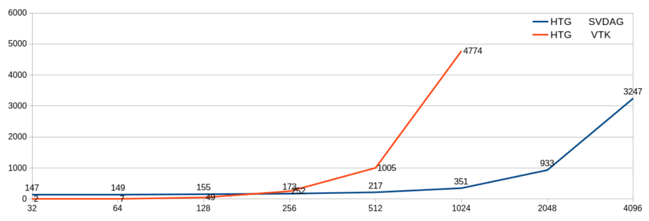

**Figure 4:** *GPU memory usage of native HTG rendering or SVDAG Ray Cast rendering, in MBytes and depending on the grid resolution for the Hairball. For this data set, and on a GPU with 8 GBytes, HTG rendering was impossible on grid resolutions superior to 1K. To be noted, HTG generates the surface to render, inside and outside. On the other hand, SVDAG rendering only computes what is required for the display, applying several optimizations such as culling. HTG surface view is computed once, and SVDAG rendering is a continuous computation depending on the camera movements. Memory was retrieved with the nvidia-smi tool, and we observe SVDAG has a constant minimum memory occupancy for this data set.*

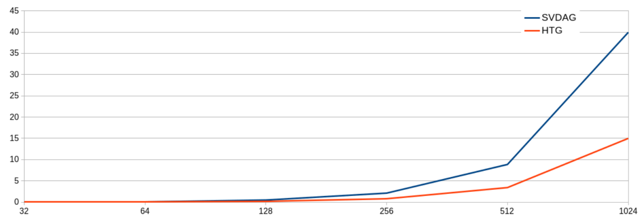

**Figure 5:** *Build time of HTG and SVDAG for the Hairball data set, in seconds and at different grid resolutions. Experiments conducted serially, and no particular optimizations applied.*

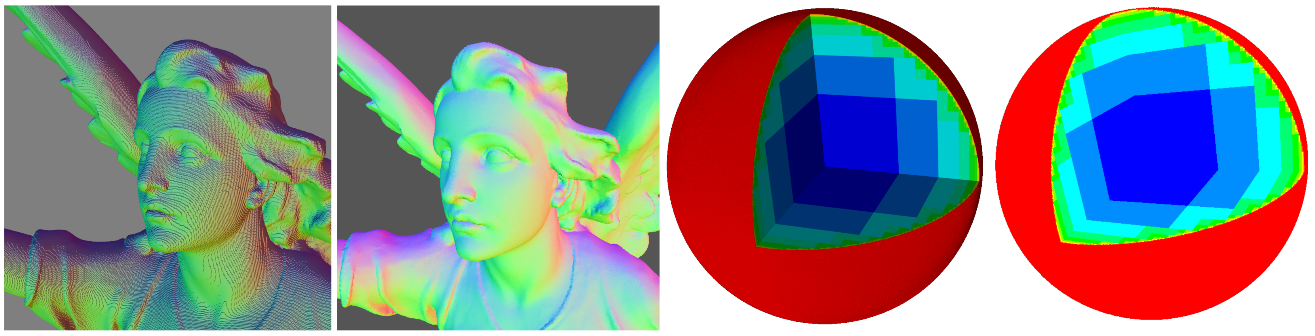

**Figure 6:** *Lucy and the Sphere models rendered with HTG on the left and SVDAG on the right. Lucy is colored by the normal, and sphere by the level of the cell in the tree. For provided SVDAG implementation, dark blue leaf cell generates a large amount of finest level leaf cell. SVDAG extremely efficient compression compensates this, but extreme uncompressed amount of generated attribute values (dark blue) induces a large memory overhead on the GPU, of about 3 GBytes. The proposed multi-SVDAG approach copes with this, and overall GPU memory consumption is less than 100 MBytes.*

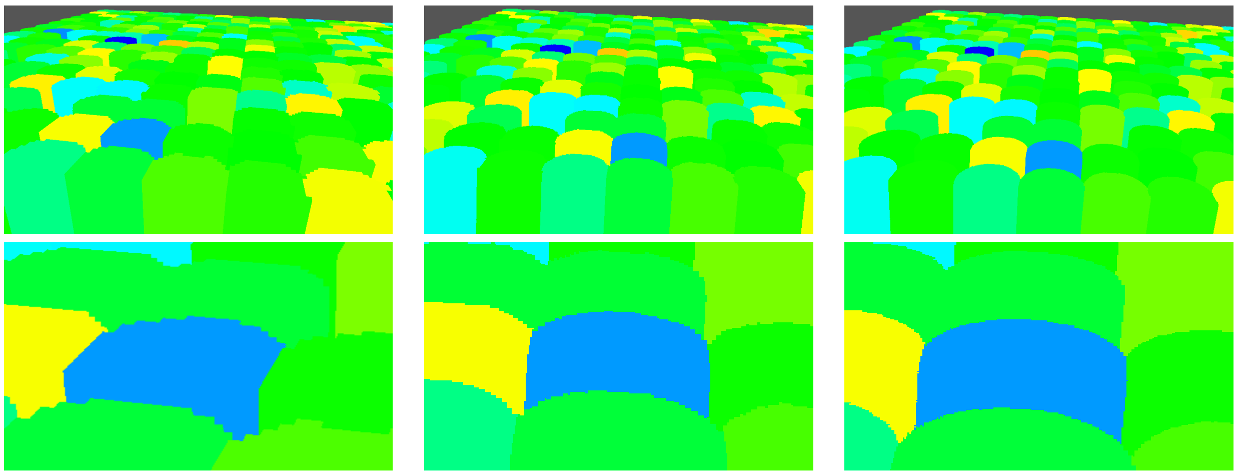

**Figure 7:** *Fuel Assembly data set loaded with depth limitations of 4, 5 and 6 (Max.) thanks to the VTK HyperTreeGrid reader capability, and ray casted as a SVDAG on the GPU. Second row shows a zoom. Removing one level during HTG loading reduces time and memory by a factor of 5x, and therefore loading HTG at level 4 is 25x faster, with 25x less voxels and attribute values. Therefore, native HTG rendering as well as SVDAG generation are faster. Speed-up may vary with other models, and maximum expected speed-up for an octree is 8x per level. This is an example of combining strengths of VTK filtering with fast efficient SVDAG rendering.*

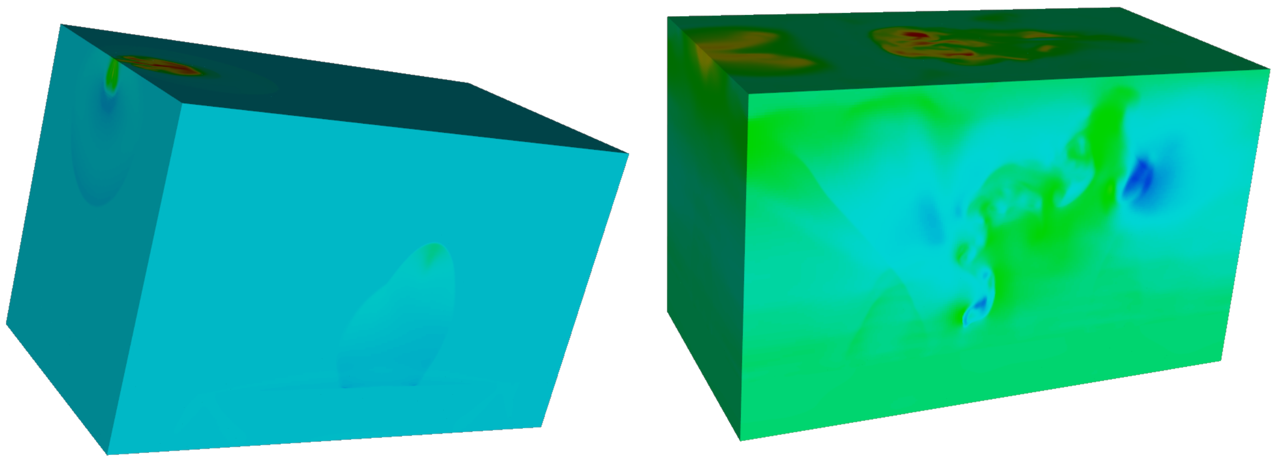

**Figure 8:** *Unfiltered (raw) display of YA31 at two time steps. We can only see details emerging on the faces of the rectangular box, and either filtering or direct volume rendering must be applied to see what is happening inside.*

© 2020 The Author(s)
Eurographics Proceedings © 2020 The Eurographics Association.