

GPU-based Dynamic Tubular Grids for Sparse Volume Rendering

David Mayerich*

Beckman Institute for Advanced Science and Technology
University of Illinois, Urbana-Champaign

John Keyser†

Department of Computer Science and Engineering
Texas A&M University

ABSTRACT

Dynamic Tubular Grids (DT-Grids) are designed to encode grid-aligned data in level-set simulations. While they are extremely efficient for storing sparse volumetric data, they require logarithmic time for random access. We demonstrate that DT-Grids can be used to efficiently render sparse volumetric data that would otherwise not be able to fit in texture memory. For many real-world biomedical data sets, such as 3D images of microvascular and neuronal networks, this results in over 10X compression compared to representation as a 3D voxel grid. DT-Grids also have several advantages over GPU-based octrees, requiring less memory and fewer indirect lookups per voxel. Finally, storing additional information per voxel does not increase DT-Grid overhead. This is an important feature for visualizing data sets from emerging imaging methods, such as Array Tomography and infrared spectroscopy, which allow many channels to be imaged per spatial voxel.

Index Terms: I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types; I.4.10 [Image Processing and Computer Vision]: Image Representation—Volumetric

1 INTRODUCTION

The visualization of large-scale biomedical data is important for understanding the structure and function of biological tissue. Recent advances in high-throughput microscopy [5, 7] and infrared spectroscopy [2] pose a particular challenge in visualization. These imaging methods produce large data sets representing complex yet sparsely-packed structures. Two examples of these are anatomical data sets representing neuronal and microvascular networks. These structures span large regions of tissue but are composed of very fine filaments that require a high-resolution representation. In addition, these data sets often have a high dynamic range (32+ bits) and contain multiple spectral components or channels, making them cumbersome to visualize since they often exceed the storage capabilities of GPUs.

We demonstrate that GPU-based Dynamic Tubular Grids [8] support efficient rendering for large sparse data sets with high-precision and multiple spectral components. We compare GPU-based DT-Grids to GPU-based octrees, known as N^3 -trees [4], which are often used for large-scale volume visualization [1]. We show that DT-Grids offer:

- Efficiency over N^3 -trees in terms of memory requirements and rendering time.
- Byte-order insertion and therefore direct streaming from out-of-core storage, making them ideal for converting large-scale volume data since the original grid does not have to be kept in main memory.

*e-mail: mayerich@illinois.edu

†e-mail: keyser@cs.tamu.edu

- Construction such that the precision required to represent the structure is independent of the data type being represented. While this is standard for CPU-based DT-Grids and octrees, other GPU-based structures such as N^3 -trees, hashing functions [3], and uniform grids have overhead that scales proportionally with the data type being represented.

2 GPU-BASED DT-GRIDS

Dynamic Tubular Grids are an efficient run-length encoding structure for sparse grid-aligned data sets and are known to provide constant-time data look-up when voxels are accessed in lexicographic order (e.g. convolution). Like octrees, DT-Grids require a sequence of indirect look-ups for random access. Implementation details for the CPU-based structure can be found in the original article by Nielsen and Museth [8]. We describe a simplified version of this structure to accelerate three-dimensional DT-Grids for parallel random access on the GPU.

We convert a sparse uniform grid to a GPU-based DT-Grid by eliminating empty voxels and projecting all valid data onto the XY plane. This results in a stack of voxels in z -order at each position in (x, y) . These voxel stacks are referred to as *columns* and are stored end-to-end in a single texture map (**texValue**). In each column, a connected component is a series of pixels that were connected in the original volume data set. Since empty voxels are eliminated, spatial context between connected components in the z -direction must be maintained. An additional 16-bit RGBA texture map (**texCoord**) is used to store the position and z -coordinate of each connected component in a column. Finally, a 16-bit RGB integer texture (**texGrid**) represents the XY plane and stores a pointer to each column.

Random access into the GPU-based DT-Grid involves a texture fetch from **texGrid**, which provides a list of connected components in **texCoord**. A binary search is then performed to find the correct connected component and position of each voxel stored in **texValue**. For most sparse biomedical data sets, the number of iterations required to retrieve the final voxel value is far less than the number of look-ups required to traverse an equivalently encoded octree.

3 RESULTS

We evaluate the performance of GPU-based DT-Grids for volume visualization of multi-channel 3D textures. Three data sets are represented using 4 channels with 32-bit floating point precision (Fig. 1) and a fourth spectroscopy data set is represented using a single channel with 32-bit floating point precision.

We encode each full-precision data set as both an N^3 -tree and DT-Grid. The memory reduction resulting from encoding is compared to the original data on a uniform grid (Fig. 2, left). In order to compare render time to 3D textures, we also cast the data to an 8-bit single-channel format so that it will fit on the graphics card as a 3D texture map. Since cache coherence plays a significant role in rendering time, we average the time required to render each data set along all three primary axes. The average render time (in milliseconds) is recorded over several thousand frames (Fig. 2, right).

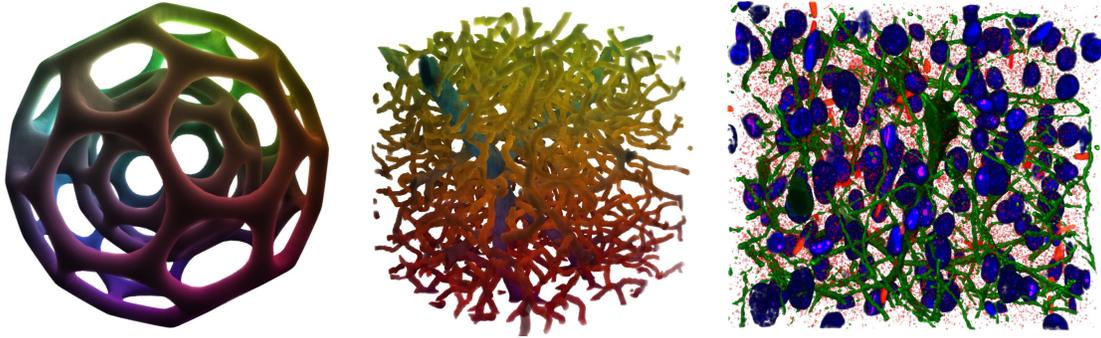


Figure 1: Volumetric data sets represented as a grid of 4 32-bit floating point values. (left) 512^3 (2.15GB) high-genus solid. (center) 512^3 (2.15GB) data set of brain microvessels captured using Knife-Edge Scanning Microscopy. Blue and red channels map to vessel diameter (largest to smallest) and the green channel reflects depth from the cortical surface. (right) $781 \times 687 \times 268$ (2.3GB) grid of neurons imaged using Array Tomography. Each channel represents the density of a different protein at that voxel location. Green voxels represent a subset of neuronal fibers, blue voxels represent DNA, and red voxels represent protein density at synaptic junctions.

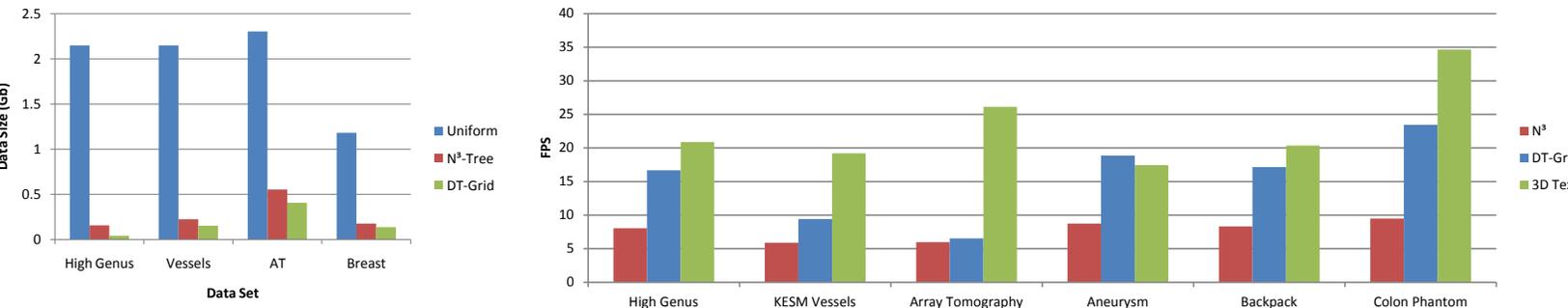


Figure 2: (left) Reduction in data set size using N^3 -trees and DT-Grids to eliminate empty space. (right) Average time required to render all voxels in three volumetric data sets (Fig. 1) and a $600 \times 300 \times 1641$ spectroscopy image of a breast biopsy. Speedup over N^3 -trees is obtained through greater cache coherence along two axes. When visualization is constrained to these two axes, rendering time becomes competitive with 3D texture maps. The spectroscopy image is an orthographic projection along one axis, therefore only the constrained time is presented.

4 CONCLUSION

The rendering time for DT-Grids is significantly lower than N^3 -trees as a result of both (a) fewer texture fetches required to evaluate the final fragment color and (b) a greater percentage of coherent and sequential texture fetches. The rendering time for 3D texture maps is larger along the x -axis while DT-Grids take longer to render along the encoding axis. This is due to incoherent texture fetches inherent in slice-based volume rendering. Methods for smoothing the frame rate have been reported previously in the literature for uniform grids [9] and are applicable to DT-Grids. However, constraining rendering to the two most efficient axes provides performance comparable to uniform grids for data that is too large to fit on the graphics card as a 3D texture map. DT-Grids allow additional data elements to be stored per-voxel without increasing the required overhead. Additional channels can be used to store information such as lighting, time-dependent functions, and identifiers for use in selective visualization [6].

ACKNOWLEDGEMENTS

The authors wish to thank Brad Busse, Steven Smith, Michael Walsh, and Rohit Bhargava for their biomedical data. This work was supported in part by NIH/NINDS grant #R01-NS4252 and the Beckman Institute for Advanced Science and Technology.

REFERENCES

- [1] C. Crassin, F. Neyret, S. Lefebvre, and E. Eisemann. GigaVoxels : Ray-Guided streaming for efficient and detailed voxel rendering. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D)*. Feb. 2009.
- [2] D. C. Fernandez, R. Bhargava, S. M. Hewitt, and I. W. Levin. Infrared spectroscopic imaging for histopathologic recognition. *Nature Biotechnology*, 23(4):469–474, Apr. 2005.
- [3] S. Lefebvre and H. Hoppe. Perfect spatial hashing. *ACM Transactions on Graphics*, 25(3):588, 2006.
- [4] S. Lefebvre, S. Hornus, and F. Neyret. Octree textures on the GPU. *GPU Gems 2*, pages 595–613, 2005.
- [5] D. Mayerich, L. C. Abbott, and B. H. McCormick. Knife-Edge scanning microscopy for imaging and reconstruction of Three-Dimensional anatomical structures of the mouse brain. *Journal of Microscopy*, 231(1):134–143, July 2008.
- [6] D. Mayerich and J. Keyser. Visualization of cellular and microvascular relationships. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1611–1618, Dec. 2008.
- [7] K. D. Micheva and S. J. Smith. Array tomography: A new tool for imaging the molecular architecture and ultrastructure of neural circuits. *Neuron*, 55:25–36, 2007.
- [8] M. Nielsen and K. Museth. Dynamic tubular grid: An efficient data structure and algorithms for high resolution level sets. *Journal of Scientific Computing*, 26:261–299, 2006.
- [9] D. Weiskopf, M. Weiler, and T. Ertl. Maintaining constant frame rates in 3D texture-based volume rendering. In *Computer Graphics International, 2004. Proceedings*, pages 604–607, 2004.