Volume Graphics (2006)
T. Möller, R. Machiraju, T. Ertl, M. Chen (Editors)

Employing Complex GPU Data Structures for the Interactive Visualization of Adaptive Mesh Refinement Data

Joachim E. Vollrath[†] Tobias Schafhitzel[†] Thomas Ertl[†]

Visualization and Interactive Systems Group (VIS)
University of Stuttgart
Universitätsstr. 38, 70569 Stuttgart, Germany

Abstract

We present a framework for interactively visualizing volumetric Adaptive Mesh Refinement (AMR) data. For this purpose we employ complex data structures to map the entire AMR dataset to graphics memory. This allows to apply hardware accelerated visualization algorithms previously only operating on uniform cartesian grids. For mapping the data to graphics memory we consider two approaches, a space-efficient, texture-based octree and a fast method based on page table address translation. We demonstrate the utility of our approach by extending an existing GPU raycasting implementation to render AMR data. As a first step to vector field visualization techniques we successfully integrated our framework into a commercial CFD postprocessing tool and visualized scalar properties of the vector field.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation–Viewing Algorithms; I.3.6 [Computer Graphics]: Methodology and Techniques–Graphics data structures and data types; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism–Raytracing

1. Introduction

AMR is a popular technique in large scale CFD simulations. For efficiency reasons, AMR data is organized in a hierarchical grid where the resolution is refined adaptively in regions of interest or where the simulation necessitates it (Color Plate I).

Runtimes for CPU-based algorithms on AMR data such as isosurface extraction are in the order of minutes for typical datasets with several million cells. Thus, it is desirable to replace them with fast and direct, GPU-based visualization techniques. However, current graphics hardware is only optimized for two- or three-dimensional data on uniform cartesian grids. Recently, the programming model of GPUs has been extended to support modern language constructs which allow to implement more complex data structures.

Our contributions are methods of mapping entire AMR datasets to graphics memory by employing GPU implementations of complex data structures that have been presented recently. Two approaches are discussed: an octree texture providing a compact representation at the cost of rendering speed, and an adaptive page table that offers high frame rates but consumes more texture memory. These approaches allow us to apply GPU-based single-pass rendering algorithms for the first time, which we demonstrate by extending a single-pass GPU raycaster to operate on adaptive grids and integrating it into a commercial CFD postprocessing tool.

Section 2 gives an overview of the related work, Sections 3 and 4 discuss the mapping approaches. Section 5 deals with AMR data visualization, followed by a discussion of our results in Section 6. Finally, we conclude this work in Section 7 and sketch our plans for future work.

2. Related Work

First introduced by Berger and Oliger [BO84], AMR has found various applications such as in astrophysics, weather and fluid dynamics simulation for engineering.

Kaufmann and Mueller [KM05] give a comprehensive general overview of volume rendering, while Pfister [Pfi05] presents hardware accelerated techniques.

[†] {vollrath|schafhitzel|ertl}@vis.uni-stuttgart.de

Both Park et al. [PBS02] and Kähler et al. [KH02] visualize AMR data interactively in a multipass approach by partitioning into bricks and the use of tree data structures for brick selection on the CPU. Park et al. render bricks with hardware-accelerated hierarchical splatting, whereas Kähler et al. rely on the classical slicing-based approach.

Our mapping of adaptive grids to graphics memory is based on the work of Lefebvre et al. [LHN05], who implemented the octree texture [BD02] on the GPU, and of Lefohn et al. [LKS*06], who presented an adaptive multiresolution data structure for the GPU.

3. Octree Texture

An octree represents a uniform cartesian grid at any depth and is therefore well suited to store AMR data. The GPU implementation represents each inner node with a cube of $2 \times 2 \times 2$ texels. These texels represent the eight children of an octree node and are either interpreted as a reference ("pointer") to the respective child if it is an inner node or as a scalar sample if the child is a leaf. This is achieved by using the alpha component to indicate the node type (empty, leaf or inner node) and the RGB-components as references within the octree texture or as scalar samples. The octree is traversed in a top-down process on the GPU, details are found in the original article by Lefebvre et al. [LHN05].

3.1. Texture Construction

The adaptive grid employed in our target CFD postprocessing tool is organized as a hierarchical data structure where each individual cell of the coarsest resolution can be considered as the root of an octree. We construct the octree texture by reconstructing a full octree covering the entire domain of the grid from these subtrees (Figure 1) and store the octree nodes in a 3D texture in depth-first order.

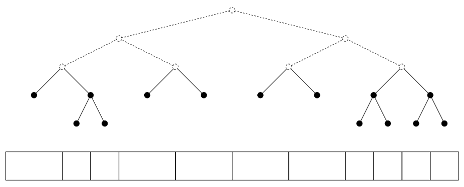

Figure 1: *Tree interpretation of an adaptive 1D grid. The dotted part has to be reconstructed.*

As described above, one color channel of each texel indicates the type of the child node. The scalar sample contained in a leaf can then be stored in one of the remaining three channels, leaving only two channels to store gradients for shading computations. Therefore, we store the gradient in spherical coordinates in the remaining two channels and convert it back into cartesian coordinates in the fragment program.

3.2. Interpolation

Lefebvre et al. [LHN05] propose a hardware accelerated interpolation scheme for the octree texture which requires all leafs of the octree to be at the same depth. Benson and Davis [BD02] propose a scheme that constructs a virtual uniform grid of the desired resolution on which trilinear or tricubic interpolation can take place. We propose an interpolation scheme more suitable for graphics hardware, which guarantees continuous interpolation within octree leafs of the same depth. Strictly speaking, the interpolation across different depths will be discontinuous, but in practice hardly any visible artifacts will occur as we show in Section 6.

For a given sampling position the octree traversal returns the sample at the nearest neighbor cell. The other seven of the eight samples needed for trilinear interpolation are retrieved by performing traversal with the sampling position offset by half the size of the nearest neighbor cell (depending on the octant in which the sampling position lies).

Interpolation of samples of the same depth in the tree is trivial and shall not be further detailed. We perform interpolation of samples at different depths with the aid of a level of detail (LOD) approach. Together with the octree texture we construct an LOD texture containing the averaged values of the child nodes for each internal node of the octree. Due to the favorable ratio between the number of internal nodes and the number of leafs (1:7 for a full octree), the additional storage requirements are tolerable. Assume that for a certain sampling position the sample value of the leaf at depth d containing this sampling position and the values at the seven neighboring cells needed for trilinear interpolation have been retrieved. Then, for each neighbor cell, the following cases may arise:

1. The neighbor cell is at depth d: Its sample is used for interpolation.
2. The neighbor cell is at depth $d+1$: The averaged value at depth d from the LOD texture is used for interpolation (Figure 2a).
3. The neighbor cell is at depth $d-1$: Its sample is used for interpolation, which is equivalent to subdividing the cell into eight children with the same value (Figure 2b).

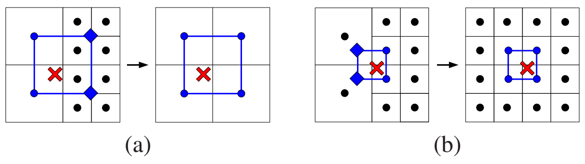

Figure 2: *The octree interpolation scheme in 2D. The sampling position is marked with a cross, required samples of a different refinement level are marked with a diamond.*

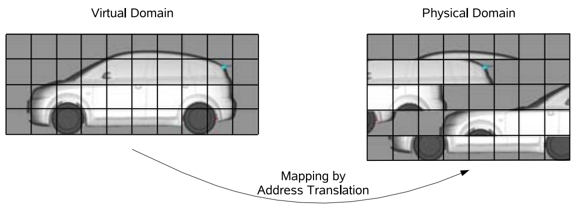

Figure 3: *Mapping between the virtual and physical domain*

4. Adaptive Page Table

Lefohn et al. [LKS*06] presented a "dynamic adaptive multi-resolution GPU data structure" which performs page table address translation on the GPU by mapping from a virtual domain to the physical domain of graphics memory. The idea is to partition space into pages with identical number of samples, stored in a page texture (Figure 3). A page table texture handles the mapping, accomplishing adaptivity by mapping multiple virtual pages to a single physical page. We implemented a cell-centered version of this data structure and refer to it as "adaptive page table" to simplify matters.

The mapping of virtual to physical pages is realized by storing a reference in the RGB color channels of each texel of the page table texture. The alpha channel stores a scaling factor depending on the refinement level of the page that is addressed.

4.1. Texture Construction

For a given page size p, the adaptive grid is partitioned into pages with $p \times p \times p$ samples. However, the refinement level of the cells covered by a page may vary. To avoid undersampling, a page is then recursively subdivided into sub-pages (each with p^3 samples) of a higher sampling rate until the level of refinement matches that of the smallest covered cell. This leads to an oversampling of some cells (Figure 4) and increases storage requirements as we discuss in Section 6.

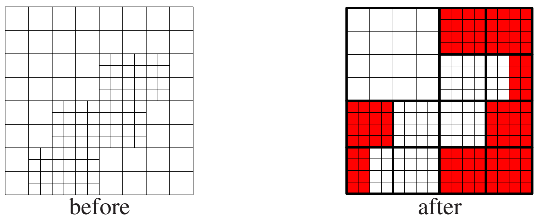

Figure 4: *Partitioning of a grid consisting of* 126 *cells with* $p = 4$. *The grid is represented by 13 pages with a total of 208 samples. Oversampled cells are highlighted in red.*

Since the pages are equally sized there is no need for any sophisticated packing in the page texture – it is simply filled with pages in the order of occurence during partitioning.

Table 1: *Overhead through sharing between physical pages*

p	8	16	32	64	128
$o(p)$	56.25%	26.56%	12.89%	6.35%	3.15%

4.2. Interpolation

The adaptive page table offers constant access complexity and can exploit hardware filtering since physical pages are stored coherently in texture memory. However, since it is not guaranteed that pages adjacent in the virtual domain are adjacent in graphics memory, continuous interpolation is ensured by additionally sharing one layer of cells between physical pages. The number of samples thus increases to $(p+2)^3$ in our cell-centered implementation. Depending on the page size p this produces an overhead:

$$o(p) = \frac{(p+2)^3 - p^3}{p^3} = \frac{6}{p} + \frac{12}{p^2} + \frac{8}{p^3}$$

Table 1 implies that it might be desirable to choose large page sizes to reduce this overhead. However, with increasing size, the aforementioned overhead through oversampling carries more weight. In practice, the actual behavior strongly depends on the structure of the adaptive grid. Therefore, we suggest to run the partitioning algorithm for various values of p and to select the best size for the dataset.

5. Visualization

To demonstrate the utility of the presented methods, we extended the single-pass raycaster by Stegmaier et al. [SSKE05]. We have chosen a raycaster due its flexibility and the greater blending accuracy.

Different visualization techniques are easily realized with the raycasting method. As a first step to vector field visualization we employ direct volume rendering, isosurface raycasting and combined techniques (Color Plate II) to visualize scalar properties of the vector field such as velocity magnitude, temperature or pressure. Results demonstrating image quality are presented in the following section.

6. Results and Discussion

Table 2 lists the footprints for three datasets. It shows that the octree approach is notably more memory-efficient. The page table approach consumes roughly three and a half times more memory due to the previously addressed overheads. An interesting conclusion can be drawn from Table 3: the octree texture shows a roughly constant overhead of about 20% for representing the adaptive grid, whereas the adaptive page table reveals a much larger variance. This proves our previous statement that the behavior of this data structure is strongly dependent on the structure of the adaptive grid.

Table 2: *Memory Requirements of 3 different AMR grids*

Model	Audi	BMW	Motorbike
Refinement levels	4	5	6
Number of Cells	2338083	3346309	5146028
Octree levels	11	13	13
Octree size (MB)	10.89	15.26	23.39
LOD size (MB)	1.02	1.43	2.19
Number of pages	1606	2621	3467
Page size (MB)	37.02	60.07	80.09
Index size (MB)	0.07	1.25	2.5

Table 3: *Ratio of number of texels and original grid cells*

Model	Audi	BMW	Motorbike
octree texture	1.22	1.19	1.19
adaptive page table	4.01	4.57	3.93

Table 4 shows performance measurements on a 3.4 GHz Pentium IV machine with a GeForce 7800 GTX card in a 500^2 viewport. Obviously, the adaptive page table dramatically outperformes the octree texture due to its better access complexity and native hardware filtering. Altogether, three to four times larger memory requirements for up to 40 times more performance with the adaptive page table must be considered a fair deal. Nevertheless, for very large AMR datasets the octree texture may be the only viable approach.

The interpolation scheme for the octree texture produces images of higher quality compared to the adaptive page table (Color Plate III), mainly because we currently employ a cell-centered implementation of the adaptive page table which produces more visible artifacts. Secondly, our octree interpolation scheme benefits from a property of the grids employed the CFD postprocessing tool: the level of refinement for two adjacent cells never differs by more than one.

7. Conclusion and Future Work

We have discussed two methods of mapping entire AMR datasets to graphics memory and visualized them using single-pass GPU algorithms. The octree approach offers a compact representation at the cost of logarithmic access complexity and a quite costly interpolation. The adaptive page table consumes roughly 3.5 times more memory but offers constant access complexity in exchange. We have extended an existing GPU raycaster to operate on both data structures and integrated it into a commercial CFD postprocessing tool to perform direct volume and isosurface rendering of scalar properties of the vector field.

The next step is to implement the node-centered page table from which we expect both an improved interpolation and reduced storage requirements. Then, we plan to store and visualize AMR vector field data using the described data structures and to apply them to time-varying data.

Table 4: *Performance in frames/second. Abbreviations: OCT = Octree Texture, APT = Adaptive Page Table, DVR = direct volume rendering, ISO = isosurface rendering.*

	Audi		BMW		Motorbike	
	OCT	APT	OCT	APT	OCT	APT
DVR	0.7	23.7	0.8	27.3	0.7	9.35
ISO	0.4	14.9	0.4	17.5	0.27	7.0

8. Acknowledgements

We thank Martin Schulz and Milosz Walter from science-computing as well as AUDI AG and BMW AG for their kind support and the permission to publish renderings.

References

[BD02] Benson D., Davis J.: Octree Textures. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), pp. 785–790.

[BO84] Berger M., Oliger J.: Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations. *Journal of Computational Physics 53*, 3 (Mar. 1984), 484–512.

[KH02] Kähler R., Hege H.-C.: Texture-based volume rendering of adaptive mesh refinement data. *The Visual Computer 18*, 8 (2002), 491–492.

[KM05] Kaufman A., Mueller K.: Overview of volume rendering. In *The Visualization Handbook*, Hansen C. D., Johnson C. R., (Eds.). Elsevier, 2005, pp. 127–174.

[LHN05] Lefebvre S., Hornus S., Nyret F.: Octree Textures on the GPU. In *GPU Gems 2, Programming Techniques for High-Performance Graphics and General-Purpose Computation* (2005), pp. 595–613.

[LKS*06] Lefohn A., Kniss J. M., Strzodka R., Sengupta S., Owens J. D.: Glift: Generic, efficient, random-access gpu data structures. *ACM Transactions on Graphics 25*, 1 (Jan. 2006), 60–99.

[PBS02] Park S., Bajaj C. L., Siddavanahalli V.: Case study: interactive rendering of adaptive mesh refinement data. In *VIS 2002: Proceedings of the conference on Visualization 2002* (2002), pp. 521–524.

[Pfi05] Pfister H.: Hardware-accelerated volume rendering. In *The Visualization Handbook*, Hansen C. D., Johnson C. R., (Eds.). Elsevier, 2005, pp. 229–258.

[SSKE05] Stegmaier S., Strengert M., Klein T., Ertl T.: A Simple and Flexible Volume Rendering Framework for Graphics-Hardware–based Raycasting. In *Proceedings of the International Workshop on Volume Graphics 2005* (2005), pp. 187–195.