# Building with Bricks: CUDA-based Out-of-Core GigaVoxel Rendering

Cyril Crassin
LJK / INRIA / Grenoble Universities
France

Fabrice Neyret
LJK / INRIA / Grenoble Universities / CNRS
France

Elmar Eisemann
Saarland University / MPI Informatik
Télécom ParisTech; Institut Télécom ; CNRS - LTCI
Germany/France

***Abstract*—For a long time, triangles have been considered the state-of-sthe-art primitive for fast interactive applications. Only recently, with the dawn of programmability of graphics cards, different representations emerged. Especially for complex entities, triangles have difficulties in representing convincing details and faithful approximations quickly become costly. In this work we investigate Voxels. Voxels can represent very rich and detailed objects and are of crucial importance in medical contexts. Nonetheless, one major downside is their significant memory consumption. Here, we propose an out-of-core method to deal with large volumes in real-time. Only little CPU interaction is needed which shifts the workload towards the GPU. This makes the use of large voxel data sets even easier than the, usually complicated, triangle-based LOD mechanisms that often rely on the CPU. This simplicity might even foreshadow the use of volume data, in game contexts. The latter we underline by presenting very efficient algorithms to approximate standard effects, such as soft shadows, or depth of field.**

## I. Introduction

Graphics hardware advances rapidly and every year its performance grows. This allows us to represent more and more complex objects, but we are still far from representing a complete artificial world at a convincing level of detail. Even the capture of real-world data, e.g., via 3D scanners, seems to advance at a pace that counteracts any advancements of graphics hardware. For medical and biological applications, gigabytes of data need to be visualized and achieving this goal in real-time is far from being trivial.

In principle, graphics hardware relies on standard rasterization to display a triangular mesh on the screen, but the recent development, away from a fixed-function pipeline to a more general device with programmable units, opens up the road for many exciting possibilities. In this work, we will present a new rendering algorithm that enables the display of gigantic volume-data sets.

Volume data is important in many applications, but efficient algorithms, like [15], hint at a more extensive use of voxels in the future. Voxels are well-suited to represent detailed elements and filtering operations lead to a multi-resolution representation that can be associated to various levels of detail. This property is interesting, as it implies that a high visual quality can be obtained even if the data set is not entirely present in memory. Instead, constant-sized blocks of volume data (so-called *bricks*) are used to suitably represent the entire data set in accordance to the current viewpoint. This reduces the memory consumption, leads to a high visual quality (because the filtering fights aliasing), and, further, faster rendering. The problem is that the data in bricks and the spatial brick structure itself need to be updated when the camera moves. This is a difficult process and the evaluation and data transfer should be performed in a suitable manner.

This work presents an efficient out-of-core volume rendering system, that exploits the parallel programming language CUDA to reach a new level of interactivity even for very complex data sets. We show that this technique could soon be integrated in many interactive applications. We underline its strength by demonstrating various rendering effects, including shadow rendering and depth of field.

## II. Previous Work

### A. Volume rendering

In many contexts, voxels are used to represent visually complex elements, e.g., fur [1], vegetation [2], or pseudo-surfaces [3], but rendering voxel data was difficult and used a lot of memory. Early solutions were cumbersome and sliced volumes [4], but since then, rendering techniques have evolved tremendously. Many GPU-based approaches have been published and a recent overview of real-time volume rendering techniques is given in [5]. The increasing GPU memory allows to hold more and more information, but very detailed information still implies a high memory cost. Unfortunately, most efficient rendering solutions assume that the entire volume fits in the GPU's memory. This highly limits the quality of the rendering and real-time approaches applied voxels mostly for distant representations (e.g., [6], [7], [8]).

Algorithms for the rendering of large volume sets exist and much research focuses on this topic. Boada et al. [9] choose varying resolution levels throughout the volume. LaMar et al. [10] choose brick resolutions according to the distance of the observer. Guthe and Strasser [11] compressed data in a view-dependent manner, but the approach involved much

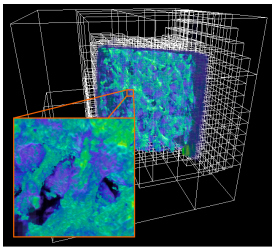

Fig. 1. An octree hierarchy is used to represent the much more detailed original volume

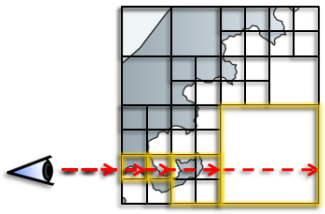

Fig. 2. A ray-tracing algorithm shoots rays through a volume hierarchy to produce the final image. Each node contains a constant sized 3D texture (brick). Depending on the viewpoint and the volume data, the octree is refined such that the resulting image is of high quality, without being obliged to keep the entire data set in memory.

CPU work. A strong view-dependent GPU-based method is still needed.

View-dependence is important, as in practice, for a single image, the full volume is rarely needed. Especially due to occlusion much data can be avoided. Further, one can benefit from clustered data, e.g., by compaction [12] or fast traversal of empty space, avoidance of occluded regions [13], or by stopping rays when a certain opacity level is reached.

Such techniques have been involved in the work by Gobbetti et al. [14] and Crassin et al. [15]. Our new approach is based on this techniques which is why we will take a closer look at these solutions in the next section.

### B. Multi-resolution brick-based Out-of-core methods

Here, we give a simplified presentation of the original GigaVoxel algorithm [15] which is intended to help the understanding of the following sections.

The main concept of the algorithm is shared with [14]. The space is subdivided via an octree. Each octree node stores a brick, a small 3D texture of some constant size (e.g., $16^3$) as illustrated in Figure 1. The nodes, as well as the 3D textures are maintained in a memory pool that resides on the GPU side. The rendering was performed using a ray-tracing approach. The octree is refined (or coarsened) based on the current viewpoint and the underlying data, as shown in Figure 2 We still rely on these principles.

One major property of the GigaVoxel algorithm is that the actual level of refinement is associated directly to the results of the rendering. In practice, this has been achieved by recovering information about the data that was used by each ray during the traversal of the structure. If information is declared missing, the corresponding ray reports this miss.

The CPU recovers these ray results and processes them. Missing data is uploaded and the octree is restructured accordingly from the CPU side. Further, an LRU-caching scheme is applied on the CPU side. Basically, each data element obtains a time stamp which is reset upon usage. When new data is needed to be uploaded the CPU, it overwrites with preference those elements in GPU memory who's usage lies most in the past.

The original system relied on a combination of shaders and GP GPU techniques to achieve interactive results.

## III. Our Method

Our new system is still relatively complex as it builds upon previous work. It combines a new fast ray-tracing algorithm and a novel efficient query mechanism. It would not be possible to describe the entire solution in this document, therefore, we will concentrate on a few improvements with respect to previous work.

First, we will show how we can decide on the GPU-side which elements are the least used and should be replaced if necessary and how to transfer a compact list to the CPU in order to initiate the update phase. With this approach, we reduce the throughput of information from the GPU to the CPU which is crucial, as the exchange of data between CPU and GPU still represents an important bottleneck in the current hardware architectures. We will analyze this process in Section III-A.

Second, this document presents two examples for efficient effects that are difficult to achieve for triangles, but can be produced rather convincingly with a voxelized scene representation (Section III-B).

### A. GPU-side LRU Caching

Previously, each ray sent back usage information of the entire node hierarchy. To reduce bandwidth, only a partial transfer was performed that exploited temporal and spatial coherence. In our new extension, we rely on a different mechanism: We maintain a *usage list* that contains all elements in the order of there most recent usage. The oldest elements are therefore always those that are situated at the beginning of the list. To make this GPU-LRU-scheme possible, we proceed as follows.

Each node and data entry has an associated *usage stamp* on the GPU side. During the ray traversal, each ray activates the usage stamps of the elements that are visited. During this marking step, we also set a flag that indicates whether or not a refinement or data upload is needed. Coarsening of the structure is implicitly handled. It is possible to employ a strategy that allows us to avoid any atomic operations in this step.

Once, all rays finished the traversal, we update the usage list. We flag all the elements in the usage list, that have been used in the current frame, which can be tested by relying on the usage stamps. This is done by looping over all the usage

stamps. Then we perform two stream reductions, to separate all elements in the usage list that were used in the current frame from the others. By relying on an order-maintaining reduction, the concatenation of the two list then gives us exactly the updated usage list we were looking for, where the least-recently-used elements are still in the beginning and the most-recently-used ones at the end. Any memory uploads can then directly replace the first elements in this list, making the approach very efficient.

### B. Voxel Effects

The regular structure of voxel data is well suited for multi-resolution representations. We exploited this property to reduce aliasing, accelerate the ray-tracing process, and decrease memory consumption. Nevertheless, the same LOD mechanisms can be applied for other effects. These effects are of interest to improve the realism of a depicted scene, but also to facilitate understanding of content.

The principle of our solutions is based on ray-differentials [16]. Low-pass filtering is at the basis of many interesting, yet complex effects and often very costly. Our idea is to rely on the multi-resolution structure to recover low-pass filtered information. This enables us to achieve convincing simulations that only add little supplementary overhead. In fact, in the case of depth-of-field, we will see that this effect can even accelerate rendering.

*1) Shadow Rendering:* Shadows are crucial cues for the understanding of spatial relationships and especially soft shadows help interpretation [17]. For standard triangular representations soft shadow algorithms usually involve complex and costly computations. In the case of our voxel data, it is possible to approximate the result with a heuristic approach that delivers high-quality rendering.

To compute a soft shadow, multiple rays, in form of a cone, are sent to the light source. A good approximation is to set the final shadow intensity as the ratio between blocked rays, that impinged on a surface and rays that reached the source. Testing all these rays separately is infeasible in real time. Instead, we interpret the cone as a filter that is applied to the data. To some extent this follows approaches for triangular representations [18], [19].

In practice, we start from an initial *impact volume* which corresponds to the pixel footprint of the location where the eye ray meets the surface. From here, we cover the cone with multiple lookups of varying size as illustrated in Figure 3. For each lookup, we receive the average of the voxel data it covers. The lookups access varying mipmap levels in the data hierarchy. Assuming a random distribution inside the averaged region, this mipmap value indicates the probability that a passing ray is blocked by the contained surface. By assuming independence of these values, the probability for a ray to be blocked by the surfaces is a multiplication of these values. The result then defines the final shading value. An example is shown in Figure 4.

For triangular models, large light sources are a challenge and not many algorithms can deal with this difficult situation.

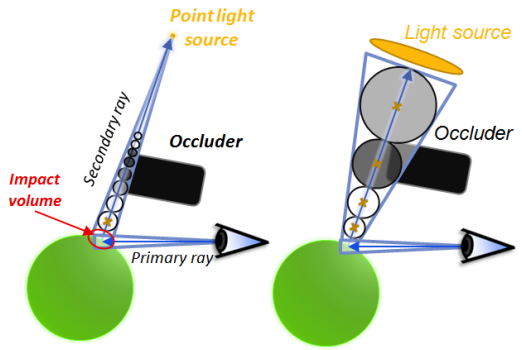


Fig. 3. To evaluate visibility of a light source, rays are shot in a cone towards the source. We approximate the ratio of blocked rays by various heuristically-combined mipmap lookups.

Fig. 4. Example of real-time shadows of our approximate algorithm.

Interestingly, the softer the shadow, the faster our algorithm becomes. This is due to the fact that more low-resolution values are recovered from the actual volume if the light source is larger. The technique is fully compatible with our out-of-core algorithm.

*2) Depth of Field:* To render depth-of-field effects, just as for shadows, we cover the set of rays with multiple lookups. In this particular case the rays form a double cone that meets on the focal plane. This is illustrated in Figure 5. A result is shown in Figure 6.

Interestingly, this time, these rays are eye rays and are directly responsible for the content of the final image. Consequently, much less high-resolution data is necessary to produce the output on the screen. This has an interesting effect: A larger depth-of-field blur results in a faster rendering and a better cache behavior. This is very different for triangle-based approaches that can be relatively expensive even for advanced algorithms [20] (although it should be pointed out that their

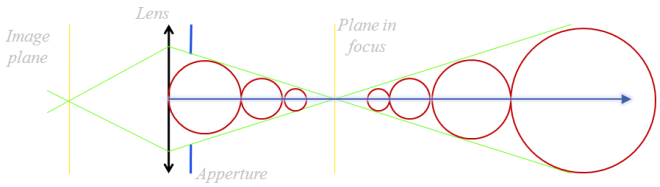


Fig. 5. Depth-of-field rendering, just like shadow computations, are obtained by shooting ray bundles.

Fig. 7. Left: large data set ($2048^3$), Middle: procedural data of a Mandelbrot set (basically infinite resolution), Right: on-the-fly voxelization of 3D models.

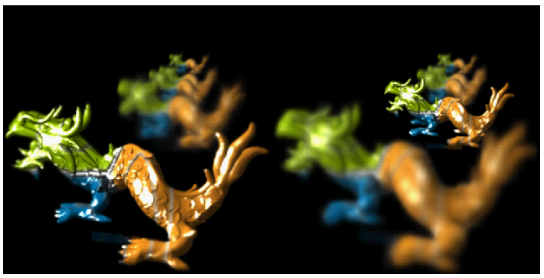

Fig. 6. Example of the out-of-focus effect with our approximate algorithm.

result is more accurate).

## IV. Results

Our approach reaches interactive rates for complex scenarios. We tested our technique on various scenes, including large volume data, procedural data, and on-the-fly voxelized content (Figure 7). The framerate varies between 20-60 fps on a GT280 graphics card depending on the viewpoint for a $1024 \times 764$ resolution. In comparison with GigaVoxels [15], the data queries are approximately twice as fast.

## V. Conclusion

We showed that our new algorithm improves the original GigaVoxel [15] approach by better exploiting parallel computing. We showed how to reduce the GPU/CPU transfer, as well as CPU interaction. We, further, simplified and accelerated the process of detecting data misses. Finally, we demonstrated that our approach allows the approximation of complex effects very efficiently. The resulting images are of very high quality, making our technique an interesting means to depict large and complex volume data representations.

## References


[1] J. T. Kajiya and T. L. Kay, "Rendering fur with three dimensional textures," in *SIGGRAPH*, 1989, pp. 271–280.

[2] P. Decaudin and F. Neyret, "Rendering forest scenes in real-time," in *Rendering Techniques (EGSR)*, june 2004, pp. 93–102. [Online]. Available: http://www-evasion.imag.fr/Publications/2004/DN04

[3] F. Neyret, "Modeling animating and rendering complex scenes using volumetric textures," *IEEE Transactions on Visualization and Computer Graphics*, vol. 4, no. 1, pp. 55–70, Jan.–Mar. 1998.

[4] P. Lacroute and M. Levoy, "Fast volume rendering using a shear-warp factorization of the viewing transformation," in *SIGGRAPH*, 1994, pp. 451–458.

[5] K. Engel, M. Hadwiger, J. Kniss, C. Rezk-Salama, and D. Weiskopf, *Real-time Volume Graphics*. AK-Peters, 2006.

[6] A. Meyer and F. Neyret, "Multiscale shaders for the efficient realistic rendering of pine-trees," in *Proceedings of GI (Graphics Interface)*, 2000.

[7] E. Gobbetti and F. Marton, "Far voxels: a multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms," in *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*. ACM, 2005.

[8] P. Decaudin and F. Neyret, "Volumetric billboards," *Computer Graphics Forum*, 2009. [Online]. Available: http://evasion.imag.fr/Publications/2009/DN09

[9] I. Boada, I. Navazo, and R. Scopigno, "Multiresolution volume visualization with a texture-based octree," *Vis. Comput.*, vol. 13, no. 3, 2001.

[10] E. LaMar, B. Hamann, and K. I. Joy, "Multiresolution techniques for interactive texture-based volume visualization," in *Proceedings of Visualization (VIS)*, 1999, pp. 355–361.

[11] S. Guthe and W. Strasser, "Advanced techniques for high quality multiresolution volume rendering," in *Computers & Graphics*. Elsevier Science, 2004, pp. 51–58.

[12] M. Kraus and T. Ertl, "Adaptive texture maps," in *ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware (HWWS)*, 2002, pp. 7–15.

[13] W. Li, K. Mueller, and A. Kaufman, "Empty space skipping and occlusion clipping for texture-based volume rendering," in *Proceedings of IEEE Visualization (VIS)*, 2003, p. 42.

[14] E. Gobbetti, F. Marton, J. Antonio, and I. Guitian, "A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets," *Vis. Comput.*, vol. 24, no. 7, pp. 797–806, 2008.

[15] C. Crassin, F. Neyret, S. Lefebvre, and E. Eisemann, "Gigavoxels: ray-guided streaming for efficient and detailed voxel rendering," in *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games*. New York, NY, USA: ACM, 2009, pp. 15–22.

[16] H. Igehy, "Tracing ray differentials," in *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 179–186.

[17] D. Kersten, D. C. Knill, P. Mamassian, and I. Blthoff, "Illusory motion from shadows," *Nature*, vol. 379, no. 31, 1996.

[18] E. Eisemann and X. Décoret, "Plausible image based soft shadows using occlusion textures," in *Proceedings of the Brazilian Symposium on Computer Graphics and Image Processing, 19 (SIBGRAPI)*, ser. Conference Series, R. L. Oliveira Neto, Manuel Menezes deCarceroni, Ed., IEEE. IEEE Computer Society, 2006. [Online]. Available: http://artis.imag.fr/Publications/2006/ED06a

[19] E. Eisemann and X. Décoret, "Occlusion textures for plausible soft shadows," *Computer Graphics Forum*, vol. 27, no. 1, pp. 13–23, 2008. [Online]. Available: http://artis.imag.fr/Publications/2008/ED08

[20] S. Lee, E. Eisemann, and H.-P. Seidel, "Depth-of-Field Rendering with Multiview Synthesis," *ACM Transactions on Graphics (Proc. ACM SIGGRAPH ASIA)*, vol. 28, no. 5, pp. 1–6, 2009.