Volume Rendering with advanced GPU scheduling strategies

Philip Voglreiter*[1], Markus Steinberger*[1], Rostislav Khlebnikov*[1], Bernhard Kainz†[2], and Dieter Schmalstieg*[1]

[1]Institute for Computer Graphics and Vision, Graz University of Technology
[2]Department of Computing, Imperial College London

Abstract

Modern GPUs are powerful enough to enable interactive display of high-quality volume data even despite the fact that many volume rendering methods do not present a natural fit for current GPU hardware. However, there still is a vast amount of computational power that remains unused due to the inefficient use of the available hardware. In this work, we demonstrate how advanced scheduling methods can be employed to implement volume rendering algorithms in a way that better utilizes the GPU by example of three different state-of-the-art volume rendering techniques.

Index Terms: D.4.1 [Operating Systems]: Process Management—Scheduling; I.3.3 [Computer Graphics]: Picture/Image Generation—Display Algorithms

1 Introduction and Related Work

While many volume rendering techniques seem to fit the *single instruction, multiple data* (SIMD) execution model of current GPUs well, a more in depth analysis often shows that processor utilization is far from perfect. One reason for this problem is divergence. If a subset of threads executing on a SIMD unit finish early, their cores are left idle. Another problem is synchronization with the CPU, which in case leads to underutilization. In the following we show how these problems manifest in common volume rendering approaches and how advanced scheduling strategies can be used to mitigate them.

Volume ray casting [3] is probably the most widespread method for displaying volumetric data. For improved image quality and depth perception, global lighting approaches are used, *e.g.*, image plane sweep volume illumination (IPSVI, Sunden et al. [6]) creates a deep shadow map on the fly. For unstructured grid data, object order methods, such as particle-based volume rendering (PBVR, [2, 4]), are preferred. These techniques employ mutually occluding, opaque particles rather than visibility sorting.

Advanced scheduling approaches can increase the overall resource utilization on GPUs. In particular, persistent thread implementations [1] allow for more efficient work distribution and the ability to create work directly on the GPU [5].

2 Test System and Datasets

We base our tests on Softshell [5], a GPU scheduling framework for CUDA. We use three regular-grid datasets: Stanford Dragon (64^3), Bonsai ($512^2 \times 182$), and MECANIX ($256^2 \times 372$). For PBVR, we use two tetrahedral datasets: Stanford Dragon (1.25×10^9 equally sized cells) and the simulation of a radio frequency ablation (1.25×10^9, min/max cell size ratio 1:1000).

*e-mail:*surname*@icg.tugraz.at
†e-mail:b.kainz@imperial.ac.uk

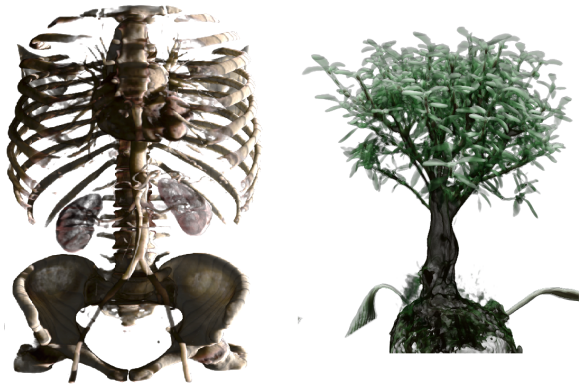

Figure 1: The Mecanix (left) and Bonsai (right) datasets rendered with single scattering (IPSVI).

3 Case Study: Ray casting

We perform a 1-to-1 assignment of rays and threads with equidistant sampling steps. For optimization, we employ early ray termination, which stops rays after reaching a certain opacity threshold. As threads finish early, thread divergence is inevitable. We analyze thread divergence for three datasets: The rather homogenous Dragon as well as Mecanix and Bonsai, which are comprised of distinct, slim features. As shown in Table 1, the utilization for the standard appraoch (STD) is between 78% and 84% for a viewport of 1024^2 pixels. Decreasing the viewport, though, raises divergence to about 60%.

Ray re-convergence Building on Softshell [5], we apply a thread re-convergence strategy. Every L iterations of the sampling loop, we check the ratio of active threads. If it drops below P%, we stop the execution of threads, save their context and regroup them with other threads via global memory. Varying L and P allows to alter the aggressiveness of the re-convergence method. Performance measurements show that, although we create higher utilization, the gain in performance is limited by the introduced overhead and consecutively speedups hover around 1. In some cases, as shown in Table 1, we even lose performance. In case of ray casting, optimal parameter settings strongly vary and are subject to optimization themselves.

4 Case Study: Plane Sweep Volume Illumination

Image Plane Sweep Volume Illumination [6] renders volumes in a scan-line fashion with single scattering and shadowing. Essentially, a shadow map is created slice-by-slice during back-to-front rendering. Intermediate synchronization via the CPU is necessary after each slice, leading to multi-pass rendering.

Optimization For optimal performance, we perform two steps. First, we increase the parallelism by using n threads per ray, taking n samples in parallel and compositing them in local shared memory. Second we spawn threads dircetly on the GPU, to avoid the multi-

Table 1: Speedup measured for DVR with different re-convergence strategies. STD is the baseline CUDA implementation, RC corresponds re-convergence with $L = 20$ and $P = 0.7$, A-RC is more aggressive with $L = 1$ and $P = 0.95$. For big viewports, groups of rays are more coherent, leading to hither utilization (util) and thereby less iterations (its). Both RC and A-RC introduce considerable overhead and limit the achievable speedup (speed↑).

		Dragon			Mecanix			Bonsai		
method	viewport	speed↑	its	util	speed↑	its	util	speed↑	its	util
STD	1024^2	1.00	365k	84%	1.00	7.02M	78%	1.00	2.67M	82%
RC	1024^2	0.81	352k	87%	0.76	6.17M	88%	1.15	2.53M	87%
A-RC	1024^2	0.67	328k	94%	0.54	6.09M	89%	0.95	2.44M	90%

pass scheme and execute rays from different slices simultaneously. Each completed ray checks its neighbors for completion and in case all data for its immediate successors is available, it starts their execution. In this way, we fill free execution units without having to synchronize with the CPU.

Results We compare a plain CUDA implementation of IPSVI with our scheduled variant. Table 2 presents the measurements. We observe a strong underutilization for the original and achieve the most significant speedups for low resolutions, while even at higher settings, the scheduled version maintains a speedup factor of 3.

Table 2: Speedup factors over the non-scheduled version observed for different resolutions while rendering MECANIX and Bonsai with IPSVI. The vertical resolution depicts the number of scan line iterations aligned to an image axis, while horizontal values correspond to the number of fragments on a single scan line. For a low number of such fragments, the parallel sampling approach contributes most to the tremendous speedup, while even at sufficient inherent parallelism, we maintain a factor of 4 over the standard CUDA version.

	MECANIX			Bonsai		
res	64	256	1024	64	256	1024
64	26.0	15.4	6.1	27.5	16.6	7.1
256	30.3	16.9	5.4	28.6	15.5	5.7
1024	29.9	15.7	4.3	28.2	15.4	4.0

5 Case Study: Particle Based Volume Rendering

PBVR [2, 4] is an object-order method used to efficiently render unstructured grid data. Simulating light emission of a dense field of particles with respect to mutual occlusion allows for volumetric rendering of such grids. The independence of the cells allows for parallel, per-cell particle generation and projection on-the-fly. First, incorporating cell size and transfer function, we determine per-cell particle counts. Next, we randomly position the particles. To account for opacity gradients, we apply a term attracting particles towards high opacities. We project the particles on the screen by combining depth buffering (mutual occlusion) and super-sampling (multiple samples per pixel).

Thread divergence during particle generation Globally, sufficiently many threads are ready for execution. However, local thread groups are non-coherent due to varying particle counts resulting from both cell sizes and transfer functions. We again apply our re-convergence strategy to increase performance.

Results In Table 3 we present the timings for both datasets. As expected, uniform datasets (Dragon) barely profit from scheduling, while rendering the non-uniform ablation dataset considerably benefits from re-convergence in a conservative setup.

6 Conclusion

We have shown that GPU-specific issues are inherent to state-of-the art volume rendering techniques. However, advanced scheduling strategies dealing with these issues can increase performance considerably. Besides testing more complex algorithms, we also want to investigate automatic adjustment of scheduling parameters on the fly in upcoming work.

Table 3: Speedup and render times measured for PBVR with re-convergence strategies. STD is the baseline CUDA implementation, RC corresponds to re-convergence with $L = 20$ and $P = 0.7$, and A-RC is more aggressive with $L = 1$ and $P = 0.95$. While rendering the considerably irregular Ablation dataset, we observe speedups even for the aggressive version. This is remarkable given the high frequency, and induced massive overhead, of coherency queries using A-RC. Note how RC outperforms A-RC for larger particle counts.

	part.	STD	RC	speed↑	A-RC	speed↑
Dragon	$5 \cdot 10^7$	31	35	0.89	45	0.69
	$1 \cdot 10^8$	56	58	0.97	89	0.64
	$1 \cdot 10^9$	375	371	1.01	848	0.44
	$4 \cdot 10^9$	1320	1230	1.07	3562	0.37
Ablation	$5 \cdot 10^7$	95	140	0.68	66	1.44
	$1 \cdot 10^8$	148	142	1.05	122	1.21
	$1 \cdot 10^9$	927	161	5.75	1186	0.78
	$4 \cdot 10^9$	3180	243	13.08	5180	0.61

Acknowledgements

This work was supported by the Austrian Science Fund (P23329), and the European Union (ICT-2011.5.2 600641). Bernhard Kainz was supported by a Marie Curie Intra-European Fellowship within the 7th. European Community Framework Programme (FP7-PEOPLE-2012-IEF F.A.U.S.T. 325661).

References

[1] T. Aila and S. Laine. Understanding the efficiency of ray traversal on gpus. In *Proc. ACM SIGGRAPH HPG*, pages 145–149. ACM, 2009.

[2] B. Csébfalvi and L. Szirmay-Kalos. Monte carlo volume rendering. In *Proc. of IEEE Visualization*, pages 449–456, 2003.

[3] M. Levoy. Efficient ray tracing of volume data. *ACM Trans. Graph.*, 9(3):245–261, 1990.

[4] N. Sakamoto, J. Nonaka, K. Koyamada, and S. Tanaka. Particle-based volume rendering. In *Visualization, 2007. APVIS*, pages 129 –132, 2007.

[5] M. Steinberger, B. Kainz, B. Kerbl, S. Hauswiesner, M. Kenzel, and D. Schmalstieg. Softshell: dynamic scheduling on gpus. *ACM Trans. Graph.*, 31(6):161:1–161:11, 2012.

[6] E. Sundén, A. Ynnerman, and T. Ropinski. Image Plane Sweep Volume Illumination. *IEEE TVCG(Vis Proceedings)*, 17(12):2125–2134, 2011.