Eurographics Conference on Visualization (EuroVis) (2015)
E. Bertini, J. Kennedy and E. Puppo (Editors)

Short Papers

State of the Art in Mobile Volume Rendering on iOS Devices

A. Schiewe[1], M. Anstoots[1], and J. Krüger[1,2]

[1]Center of Visual Data Analysis and Computer Graphics (CoViDAG) & HPC Group, University of Duisburg-Essen, Germany
[2]Scientific Computing and Imaging (SCI) Institute, University of Utah, USA

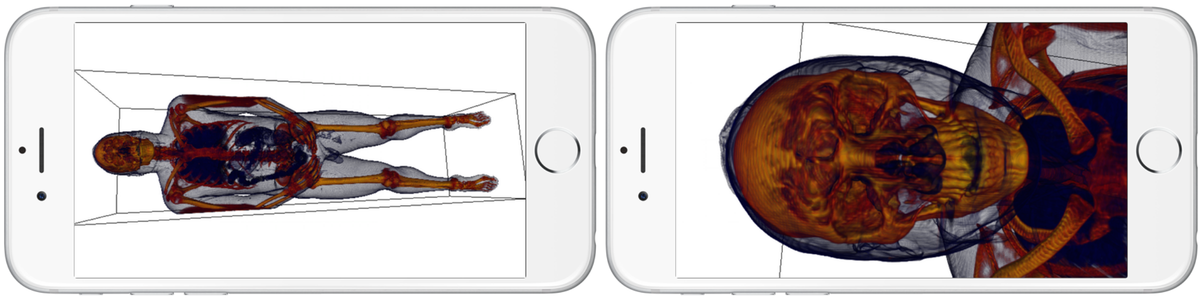

Figure 1: *Next generation low-level graphics APIs allow for unprecedented performance on mobile devices. The Visible Human CT dataset with a resolution of 256×256×942 voxels ray-casted at interactive frame rates of 5 FPS (left) and up to 7 FPS (right).*

Abstract
The ubiquity of ever-increasing computing power with mobile devices has put last generation desktop-grade hardware in everyone's palms. Mobile computing hardware is rapidly approaching today's desktop-grade hardware capabilities enabling applications of advanced rendering algorithms to previously untouched environments such as medical care. Recent developments in graphics APIs have introduced novel low-level APIs such as AMD's Mantle API for desktops and Apple's Metal API for mobile hardware. Microsoft's DirectX 12 and the OpenGL successor Vulkan will be available in the near future. AAA game titles were announced for which publishers see an advantage—as promised by the creators of the new APIs—over traditional portable implementations. The new APIs are mostly advertised to allow for more draw-calls per frame compared to for example, OpenGL-based solutions. Visualization algorithms and in particular direct volume rendering do not exhibit a significant amount of draw-calls as a bottleneck. This work evaluates and highlights the utility of recent low-level APIs for mobile devices and puts them into perspective with available alternatives.

Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computing Methodologies]: Computer Graphics—Methodology and Techniques I.3.2 [Computing Methodologies]: Computer Graphics—Graphics Systems

1. Introduction

Volume rendering has been shown to be an effective tool in a number of application areas, ranging from engineering to medicine. Recently, mobile volume rendering has gained attention as portable devices such as tablets and smartphones are present in practically any area of social life and many workplaces. In a number of recent works, mobile rendering has been demonstrated to be advantageous over similar visualization methods on the desktop [BTJ*13].
To bring visualization in general and volume rendering in particular to the mobile device, two principal directions can be followed: firstly, server-based rendering where the datasets and the rendering engine reside on a server machine or desktop and rendered images are streamed to the mobile client and secondly, on-device rendering, where the datasets are streamed to the device up front or on demand and rendered directly on the mobile client. The former method has the advantage of leveraging the superior computing horsepower of the desktop whereas the second method is more robust to network limitations. (Furthermore, we will show that from

DOI: 10.2312/eurovisshort.20151139

a power consumption standpoint, remote rendering can be more efficient.)
In this paper, we will focus on the former method and demonstrate how recent advantages in mobile hardware and software impact client-based rendering on these devices. Therefore, we implemented a mobile ray-caster using OpenGL ES 3.0 and Metal functionalities. Finally, we compare these implementations to previous OpenGL ES 2.0-based solutions.

2. Related Work

For a complete introduction to volume rendering techniques, we refer the reader to the book by Engel et al. [EKRSW06]. Early mobile volume rendering systems [MW08] based on the OpenGL ES 1.x specification were forced to utilize 2D texture-based approaches with object-aligned slices [RSEB*00] due to the lack of alternatives.
For the first time, the programmable rasterization pipeline for mobile devices was introduced with OpenGL ES 2.0, this enabled GPU-based direct volume ray-casting implementations as pioneered by Krüger et al. [KW03] during the evolution of desktop-grade graphics boards. At this time, mobile ray-casting implementations for OpenGL ES 2.0 hardware were reported to be inefficient [RA12, Noo12] compared to the alternative slice-based solution. Only a few graphics chips had hardware support for 3D textures to avoid costly manual trilinear interpolation operations in the shader code. Special slice-based sampling techniques were developed to bring the image quality level of sliced-based approaches closer to that of ray-casting-based solutions without sacrificing as much performance as switching completely to a ray-caster [Krü10]. Currently available mobile volume rendering systems [VES, BTJ*13] and WebGL-based solutions [NJ12, ML12, MCL14] still build on the OpenGL ES 2.0 specification. According to our knowledge at the time of writing, we are not aware of any published OpenGL ES 3.x class volume rendering system.

In contrast to local rendering methods, remote and potentially distributed server-based approaches [HHN*02, LS07, PKI08, TK14, NDB*14] are a viable alternative, especially when it comes to the display of large datasets that would not fit the mobile system-on-a-chip's available memory.

3. Graphics APIs

This work evaluates the potential of different volume rendering implementations for—but not limited to—the iOS platform, which enables new avenues of graphics programming not yet seen on any other mobile platform, in particular the new low-level graphics API Metal. The general question arises: "Is it worth investing resources to support another API besides OpenGL ES?" The industry tries to answer this question by advertising this new API to excel with very low operation overhead and more direct control by the programmer, resulting in many more draw-calls that can be issued in the same time frame compared with conventional APIs. However, visualization purposes in contrast to games, especially for volume ray-casting, the number of draw-calls is countable, namely two single draw-calls per frame to render the front-facing and back-facing sides of the proxy-geometry unit cube as the entry point for the ray-casting shader. In the remainder of the paper, we address to answer the question of resource expenditure.

4. Test Environment

Our target test hardware includes an iPhone 6 Plus and an iPad Air 2 device. All reported tests are done with the iPhone running the latest iOS version 8.1.3. The measured values could not be reproduced with the iPad running the same iOS version. In fact, the performance numbers across the local renderers were almost identical and even slightly worse for Metal. We account this to graphics driver optimization issues. Developer versions as of iOS 8.3 beta confirm the findings for the iPhone presented in this paper.
The devices' graphics capabilities are driven by Imagination Technologies' PowerVR G6430 and GX6650 graphics processors, which render into 1536×2048 and 1242×2208 frame buffers for display at so-called *retina* resolutions.

Unless otherwise noted, all performed tests used, for volume rendering purposes, well-known *Bonsai* dataset from [BM] with a downsampled resolution of 128^3 voxels at 8 Bits per voxel and a total of 8 MiB in memory size. In addition to the scalar volume data, we store precomputed gradients for lighting computations in our volume datasets. We did not use the original 256^3 sized *Bonsai* dataset because we could not reliably load it throughout all test runs with an OpenGL ES 2 graphics context. In order to utilize the graphics hardware to its full capacity, we instead increased the sampling rate of the dataset to 400% and render at full *retina* output resolution. Please see Figure 4 for reference images of the rendered frame sequences on the iPhone.
For equivalent sampling rates of the 2D texture slice-based OpenGL ES 2.0 implementation and the two 3D texture-based ray-casters implemented with OpenGL ES 3.0 and Metal, we set the ray-casters ray-marching step size to a value that the sample count matches at least the number of slices drawn by the slice-based volume renderer (SBVR). This means that the ray-casters do even more work in terms of sampling in cases where early ray termination does not occur. The main test scenario rotates the dataset constantly around its x-axis. This results in at most $\sqrt{2}$ times more samples for the ray-caster than for the SBVR at 45-degree viewing angles of the volume.

5. Performance and Efficiency Evaluation

5.1. Battery consumption

The first scenario tests the number of frames that can be rendered with a fully charged device. We disconnected the device from the power supply and started the endless rotating *Bonsai*. At 20% battery level, an alert window from the operating system pops up and interrupts the test run. Because of this interruption and potential energy saving mechanisms triggered with lower battery levels, we did not continue our

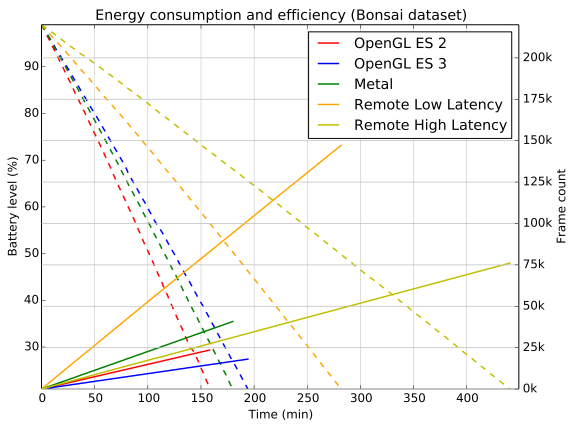

Figure 2: *Energy consumption (dashed lines) and number of total rendered frames (solid lines) over time for a test run from a battery level of 100% down to 20% for a variety of different renderers on the iPhone 6 Plus with the highest quality settings.*

Renderer	Frames per battery level	Frames per second
OpenGL ES 2	298.35	2.49
OpenGL ES 3	228.82	1.55
Metal	516.85	3.78
Remote Low Latency	1863.48	8.71
Remote High Latency	962.72	2.88

Table 1: *Average values derived from the energy consumption test shown in Figure 2.*

test runs below 20%.
Figure 2 and Table 1 show the results of local and remote renderers. Our remote renderer uses high-quality JPEG-based image compression as done by others [NDB*14] and streams the data to the mobile device. The mobile device itself uses an OpenGL ES 2 context to display the received data. We conducted two remote renderer scenarios: one with a high maximum bandwidth and low latency (limited by 433 Mbit/s Wi-Fi connection; average ping of 3.6 ms; one hop to the render server) and one with a low maximum bandwidth and high latency (limited by 12.6 Mbit/s ADSL2+ connection; 52.4 ms ping; 14 hops) Wi-Fi network connection. The renderer with low latency used an effective transfer rate of 15 Mbit/s and the renderer with high latency 8 Mbit/s. Both transferred the same images but at different speeds.

In terms of performance and energy consumption, the low latency remote renderer clearly outperforms the other approaches with approximately 9 FPS and almost 1900 frames per battery level (FPBL). Metal is able to claim the second place and is superior to the other local renderers with respect to speed at 4 FPS, but the remote renderer with high latency places second with 960 FPBL.

Among the local renderers, OpenGL ES 3 consumed the least energy but rendered slowest. OpenGL ES 2 SBVR's frame rate placed half way between the other two local renderers and consumed the most energy of all renderers. We attribute this to a measured increased CPU utilization to manage the slice stacks and to the larger memory footprint of two times the dataset size for the two additional slice stacks. The SBVR had a total measured memory footprint of 66 MiB and the two ray-casters 47 MiB.

The remote renderer with high latency was able to run 36% longer but with 48% fewer frames than the one with the low latency. This also reflects the measured effective transfer rates and shows that the hardware does efficiently save the battery when waiting for a frame to display.

5.2. Performance

Figure 3 highlights the individual frame times of a full 360-degree rotation of the *Bonsai* dataset with maximal quality settings (see Figure 4 for reference images). Furthermore, we compare the *Bonsai* dataset to an "empty" dataset with identical dimensions where its transfer function is set to render every voxel fully transparent. This worst case scenario forces a constant high GPU utilization over the whole test run. Comparing the graphs for both datasets, the performance gain at 290-degree rotation angle clearly highlights the early ray termination advantage of the ray-casters over conventional SBVR methods. For this setting, even the slower OpenGL ES 3 renderer outperforms the OpenGL ES 2 renderer.
The OpenGL ES 2 graph resamples the plot of the absolute value of a cosine function, which peaks at 90-degree angles. This is a common SBVR frame time pattern that relates to the number of fragments generated by the rotated slice stacks. At the 45-degree view angles, the slices create the fewest fragments, and if the viewing direction is perpendicular to the stack, the fragment count is maximized.

The largest loadable volume dataset was the only by one factor downsampled original Visible Human CT scan dataset consisting of $256^2 \times 942$ voxels at 8 Bits per voxel with a total size of 236 MiB. This data can be rendered with Metal at 5 FPS for a total view and with up to 7 FPS for a close-up view. OpenGL ES 3 renders the same images with not more than 2 FPS for the first and 4 FPS for the second view. Reference images for these two setups are shown in Figure 1. Only for these performance measures the quality settings were set to 100% sampling rate and disabled *retina* resolution rendering four times less pixels. The SBVR is not able to load a dataset of this size due to the storage overhead of the additional slice stacks.

6. Discussion

Besides pure performance considerations for direct volume rendering of single volume datasets, numerous use cases require incorporation of additional primitive types such as semi-transparent triangle meshes.
In this discipline, SBVRs excel the ray-casters with a straightforward integration of mesh rendering support and comparably fast rendering performance. The geometry can be

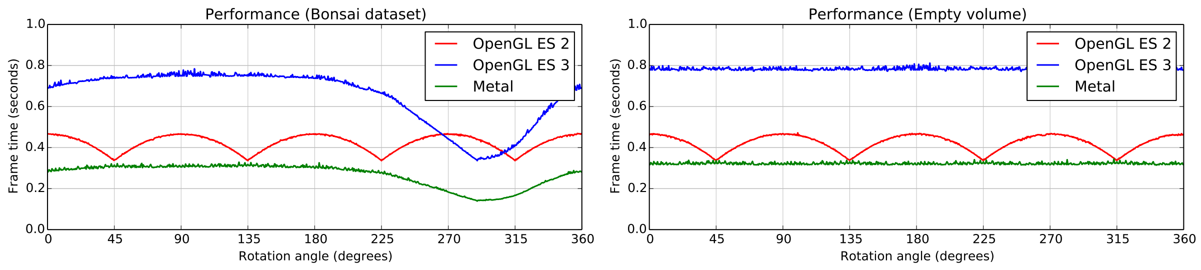

Figure 3: *Frame times for the 360-degree rotations around the x-axis of the Bonsai dataset (left) and a volume with a totally transparent transfer function (right). The empty volume prevents early ray termination and thus forces many samples, which states the worst case scenario.*

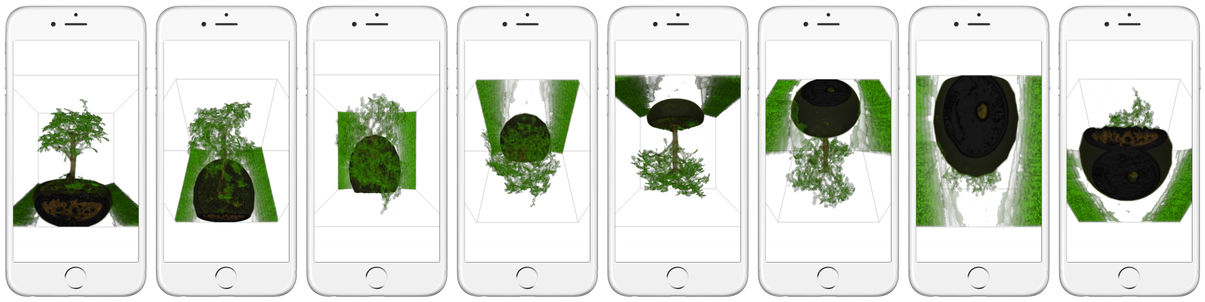

Figure 4: *Selected rendered frames for reference as they appear on the iPhone 6 Plus device running the rotation test scenario where the Bonsai dataset is successivly rotated around its x-axis.*

sorted view-dependently on the CPU and merged into the slice render call. Ray-casters require, for instance, depth-peeling [Eve01] to achieve the same goal at lower frame rates than SBVRs.

An advantage of ray-casters is the natural support for iso-surface rendering that can be done with OpenGL ES 3 and Metal at real-time frame rates.

7. Conclusions and Future Work

In this work, we analyzed the potential of modern graphics APIs on mobile devices with a special emphasis on direct volume rendering. It has been shown that new low-level APIs for local on-device rendering are superior to traditional APIs even for a fairly limited number of draw-calls. We conclude that it is worth the effort to invest resources in implementing visualization algorithms with these new APIs unless code portability is a major concern.
In contrast to local rendering, it has been shown that remote rendering is a viable battery-preserving alternative if wireless network reception is not an issue.
Hybrid rendering techniques [TK14] are feasible as an alternative to local or remote rendering only. Such techniques combine remote and local rendering into one common framework and allow for “the best of both worlds” with seamless handovers depending on the network or battery conditions.

Another advantage of Metal and possibly a disadvantage of OpenGL ES is that it is likely that new hardware features are being more quickly exposed through low-level APIs than through portable specifications across vendors and platforms. Current iOS devices operate on OpenGL ES 3.1 level hardware but so far only OpenGL ES 3.0 functionality is exposed to the programmer in contrast to the current version of Metal that also exposes modern OpenGL ES 3.1 features such as `atomic_compare_exchange`. This is the essential shader instruction to implement a *modern* ray-guided volume renderer [BHP14, FSK13, HBJP12], which is the state-of-the-art technique to render extremely large datasets on current generation desktop hardware. Proving the utility of this advanced rendering technique for mobile hardware is beyond the scope of this paper but is an interesting avenue for future work.

8. Software

The OpenGL ES 2.0 SBVR and the remote renderer implementation evaluated in this paper has been released as the *ImageVis3D Mobile* software version 4.03 and is availabe for download on the App Store. Future versions might include the other local renderers.

9. Acknowledgements

This research was made possible in part by the Intel Visual Computing Institute; the NIH/NCRR Center for Integrative Biomedical Computing, P41-RR12553-10; and by Award Number R01EB007688 from the National Institute of Biomedical Imaging and Bioengineering. The content is the sole responsibility of the authors.

References

[BHP14] BEYER J., HADWIGER M., PFISTER H.: A Survey of GPU-Based Large-Scale Volume Visualization. *Eurographics Conference on Visualization EuroVis 33*, 3 (June 2014), 1–10. 4

[BM] BARTZ D., MEISSNER M.:. "Real World" medical datasets [online]. http://volvis.org. 2

[BTJ*13] BUTSON C. R., TAMM G., JAIN S., FOGAL T., KRÜGER J.: Evaluation of Interactive Visualization on Mobile Computing Platforms for Selection of Deep Brain Stimulation Parameters. *Visualization and Computer Graphics, IEEE Transactions on 19*, 1 (2013), 108–117. 1, 2

[EKRSW06] ENGEL K., KNISS J. M., REZK-SALAMA C., WEISKOPF D.: *Real-Time Volume Graphics*. A. K. Peters, Ltd., Natick, MA, USA, 2006. 2

[Eve01] EVERITT C.: *Interactive Order-Independent Transparency*. Tech. rep., 2001. 4

[FSK13] FOGAL T., SCHIEWE A., KRÜGER J.: An Analysis of Scalable GPU-based Ray-guided Volume Rendering. *Large-Scale Data Analysis and Visualization, IEEE Symposium on* (2013), 43–51. 4

[HBJP12] HADWIGER M., BEYER J., JEONG W.-K., PFISTER H.: Interactive Volume Exploration of Petascale Microscopy Data Streams Using a Visualization-Driven Virtual Memory Approach. *Visualization and Computer Graphics, IEEE Transactions on 18*, 12 (2012), 2285–2294. 4

[HHN*02] HUMPHREYS G., HOUSTON M., NG R., FRANK R., AHERN S., KIRCHNER P. D., KLOSOWSKI J. T.: Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters. *Graphics, ACM Transactions on 21*, 3 (2002), 693–702. 2

[Krü10] KRÜGER J.: A new sampling scheme for slice based volume rendering. In *VG'10: Proceedings of the 8th IEEE/EG international conference on* (May 2010), Eurographics Association, pp. 1–4. 2

[KW03] KRÜGER J., WESTERMANN R.: Acceleration Techniques for GPU-based Volume Rendering. *IEEE Visualization* (2003), 287–292. 2

[LS07] LAMBERTI F., SANNA A.: A Streaming-Based Solution for Remote Visualization of 3D Graphics on Mobile Devices. *Visualization and Computer Graphics, IEEE Transactions on 13*, 2 (2007), 247–260. 2

[MCL14] MOVANIA M. M., CHIEW W. M., LIN F.: On-Site Volume Rendering with GPU-Enabled Devices. *Wireless Personal Communications 76*, 4 (2014), 795–812. 2

[ML12] MOVANIA M. M., LIN F.: Ubiquitous Medical Volume Rendering on Mobile Devices. In *Information Society (i-Society), 2012 International Conference on* (2012), pp. 93–98. 2

[MW08] MOSER M., WEISKOPF D.: Interactive Direct Volume Rendering on Mobile Devices. *VMV* (2008), 217–226. 2

[NDB*14] NACHBAUR D., DUMUSC R., BILGILI A., HERNANDO J., EILEMANN S.: Remote Parallel Rendering for High-resolution Tiled Display Walls. *Large-Scale Data Analysis and Visualization, IEEE Symposium on* (2014), 117–118. 2, 3

[NJ12] NOGUERA J. M., JIMENEZ J.-R.: Visualization of Very Large 3D Volumes on Mobile Devices and WebGL. *WSCG Communication Proceedings* (July 2012), 105–112. 2

[Noo12] NOON C. J.: *A Volume Rendering Engine for Desktops, Laptops, Mobile Devices and Immersive Virtual Reality Systems using GPU-Based Volume Raycasting*. PhD thesis, 2012. 2

[PKI08] PARK S., KIM W., IHM I.: Mobile collaborative medical display system. *Computer Methods and Programs in Biomedicine 89*, 3 (2008), 248–260. 2

[RA12] RODRÍGUEZ M. B., ALCOCER P. P. V.: Practical Volume Rendering in Mobile Devices. *ISVC 7431*, Chapter 67 (2012), 708–718. 2

[RSEB*00] REZK-SALAMA C., ENGEL K., BAUER M., GREINER G., ERTL T.: Interactive Volume on Standard PC Graphics Hardware Using Multi-textures and Multi-stage Rasterization. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware* (New York, NY, USA, 2000), ACM, pp. 109–118. 2

[TK14] TAMM G., KRÜGER J.: Hybrid Rendering with Scheduling under Uncertainty. *Visualization and Computer Graphics, IEEE Transactions on 20*, 5 (2014), 767–780. 2, 4

[VES] VTK OpenGL ES Rendering Toolkit [online]. http://www.vtk.org/Wiki/VES. 2