Real-Time Interactive Platform-Agnostic Volumetric Path Tracing in WebGL 2.0

Žiga Lesar
Faculty of Computer and Information Science,
University of Ljubljana
Ljubljana, Slovenia
lesar.ziga@gmail.com

Ciril Bohak
Faculty of Computer and Information Science,
University of Ljubljana
Ljubljana, Slovenia
ciril.bohak@fri.uni-lj.si

Matija Marolt
Faculty of Computer and Information Science,
University of Ljubljana
Ljubljana, Slovenia
matija.marolt@fri.uni-lj.si

Figure 1: Progressive refinement of a rendering of the bonsai dataset. The resolution of the dataset was 512×512×182, and the resolution of the internal frame buffers was 512×512. At 20 frames per second the first image appeared on the screen in a fraction of a second, showing a recognizable shape. The last and most accurate image took 10 seconds to generate.

ABSTRACT

Path tracing has become a de facto standard for photo-realistic rendering due to its conceptual and algorithmic simplicity. Over the last few years, it has been successfully applied to the rendering of participating media, although it has not seen widespread adoption. Most implementations are targeted at specific platforms or hardware, which makes them difficult to deploy or extend. However, recent advancements in web technologies enable us to access graphics hardware from a web browser in a platform-agnostic manner. Therefore, in this paper, we present an implementation of a state-of-the-art volumetric path tracer developed in JavaScript using WebGL 2.0. The presented solution supports the use of arbitrary 2D transfer functions and heterogeneous volumetric data, aims to be interactive, platform-agnostic, easily extensible, and runs in real-time - both on desktop and mobile devices.

CCS CONCEPTS

• **Computing methodologies** → **Ray tracing**;

KEYWORDS

path tracing, WebGL, volume rendering

ACM Reference Format:
Žiga Lesar, Ciril Bohak, and Matija Marolt. 2018. Real-Time Interactive Platform-Agnostic Volumetric Path Tracing in WebGL 2.0. In *Web3D '18: The 23rd International Conference on Web3D Technology, June 20–22, 2018, Poznan, Poland*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3208806.3208814

Web3D '18, June 20–22, 2018, Poznan, Poland

ACM ISBN 978-1-4503-5800-2/18/06…$15.00
https://doi.org/10.1145/3208806.3208814

1 INTRODUCTION

Direct volume rendering (DVR) and ray casting have come a long way since their conception in the 1980s [Kajiya 1986; Levoy 1990]. Early models, such as maximum intensity projection (MIP) and the emission-absorption model (EAM), have received numerous improvements, extensions, and adaptations for use on the GPU. Majority of those improvements aim to improve visual realism by simulating or approximating certain global illumination effects, such as shadows, scattering, and ambient occlusion. Some effects to simulate the behavior of a camera lens have also been introduced, such as depth of field and adjustable exposure [Barsky et al. 2003; Kolb et al. 1995]. These have been proven to aid in perception of size, depth and shapes in images, leading to the research being focused on improving the quality and performance of these effects. Various approximate and fast solutions have been proposed, but producing high-quality renderings is inevitably computationally expensive. Modern graphics hardware has successfully been used to aid in computing the tasks that are easy to parallelize, but certain peculiarities that arise from the vastly different architecture and programming model force the methods to abide by their restrictions,

ultimately leading to solutions that only work in a specific setting, e.g. with a constant transfer function or a constant light setting.

Moreover, the usual approximate methods that work for solid surfaces often cannot be used to render participating media, or have to be heavily adjusted (e.g. adaptation of bidirectional path racing by Lafortune and Willems [1996]). The adaptations are in fact often prohibitively slow. A general solution that works with both solid surfaces and participating media is to use stochastic methods that quickly produce a recognizable image and improve on it over time, as shown in Figure 1. This family of methods is known as Monte Carlo ray tracing (MCRT).

MCRT has long been the de facto standard for photo-realistic rendering. It is simple to formulate, implement, and parallelize. If used in conjunction with physically based light transport, it can simulate realistic light behavior in a progressive manner, resulting in realistic renderings and an interactive user experience. There exist numerous implementations tailored to run on the GPU - with certain hardware limitations [Davidovič et al. 2014; Hachisuka 2015; Hadwiger et al. 2009; Kroes et al. 2012].

Due to the advancements in graphics hardware and web technologies, it has lately become possible to program complex graphics-based applications that run in a web browser. Many of the most recent contributions and state-of-the art applications focus on the methodology, rather than the implementation itself, resulting in non-portability. Examples of that include the use of Microsoft's DirectX to access the graphics hardware - working only on Microsoft's platforms - or using Nvidia CUDA for parallelization - working only on Nvidia graphics cards. Although these solutions are reasonably well supported and in widespread use, it is in many occasions unreasonable to restrict the software to specific platforms of hardware, availability and maintainability being our primary concerns. Web applications on the other hand focus on delivering the same capabilities in a platform-agnostic way, for example by using WebGL for high performance graphics.

Since its introduction in 2011, WebGL has been adopted by many web developers and web browser developers as a platform-agnostic way to access the capabilities of modern graphics hardware. It is based on OpenGL ES 2.0, a standard for high performance graphics on embedded systems. The ES standard follows a few steps behind the regular version of the standard, lacking in quite a few important features. Additionally, the API is designed with embedded processors' capabilities - or rather the lack thereof - in mind, which poses a unique challenge in developing sophisticated graphical applications.

Over the last few years, a new version of the WebGL standard has been in development. Although still in draft form, WebGL 2.0 has recently been enabled by default on some platforms. Since early 2017, major web browsers (Google Chrome for desktop 56, Google Chrome for Android 58, Android WebView 58, Opera 43, and Mozilla Firefox 51) have WebGL 2.0 support enabled by default, both on desktop and mobile devices. WebGL 2.0 provides significant improvements over WebGL 1.0 both in terms of capabilities and performance, by providing a more full-featured API to simplify interaction with the hardware. Two notable additions are 3D textures, which are especially welcome in volume rendering applications, and numerous new texture formats, which enable us to make use of high dynamic range (HDR) rendering without using any OpenGL extensions. Although floating point formats are present out of the box, rendering to a floating point frame buffer is still available only through an extension.

WebGL was already used in some experiments to create an advanced platform-agnostic rendering framework, e.g. by Hachisuka [2015] and Congote et al. [2011], but these attempts fall short by only focusing on solid surface rendering or implement only basic volume rendering methods. Also, to our knowledge, no attempts have been made so far to deliver such a framework using the modern WebGL 2.0 standard.

In this work, we combine MCRT and WebGL 2.0 to deliver a platform-agnostic application for photo-realistic rendering of volumetric datasets. The application is designed to be interactive at real-time speeds, while providing complete control over the transfer function, camera, and lighting. It is written modularly to allow for extensions and adjustments, and works on all desktop and mobile devices that support WebGL 2.0 with extensions for floating point frame buffers.

2 RELATED WORK

A comprehensive study by Jönsson et al. [2014] analyzes many illumination models used in modern volume rendering applications. A more recent publication by Fong et al. [2017] has been used as a reference for the most important and promising methods used in the industry nowadays. Recent state-of-the-art reviews on GPU-based direct volume rendering (by Balsa Rodríguez et al. [2014]) and GPU-based large-scale volume visualization (by Beyer et al. [2015]) give a nice insight in recent research in the volume rendering field. These sources have served us as primary sources for the work presented in this paper.

Some of the earliest examples of volume rendering methods that form the basis for more recent contributions are described by Max [1995], and are still in widespread use today, mainly due to their simplicity and ease of implementation. However, these methods have various drawbacks, both methodological, such as bias, and perceptual, i.e. providing little or no visual cues to aid in shape or depth perception. Visual cues provided by illumination effects, such as shadows and ambient occlusion, greatly improve the user's perception (as noted by Lindemann and Ropinski [2011]), and were the main area of research for many years. Lesar et al. have considered the same issue in the context of angiogram visualization in [Lesar et al. 2015].

The simplest attempt to introduce global illumination effects into volume rendering is to precompute a shadow volume and sample it during ray propagation along with volume data, as described by Behrens and Ratering [1998]. This technique is both time- and memory-consuming, but it laid groundwork for a number of advanced techniques. A very popular technique, deep shadow maps by Lokovic and Veach [2000], has been improved and adapted for many different scenarios, including GPU ray casting, as shown by Hadwiger et al. [2006]. Although deep shadow maps are a fast and convenient way to display shadows in a volume, their main weakness is that it is only possible to simulate direct illumination from a single point light source.

It is often more useful not to consider any single light source by itself, but rather a point's shading by its immediate surroundings,

which is effectively global illumination on a more local scale. The term *ambient occlusion* (AO) is used for methods that approximate global illumination by taking into account the local neighborhood of a point on a surface, as described in the original paper by Zhukov et al. [1998]. The method has many variants, including screen-space approximations and GPU-based adaptations.

Schott et al. [2009] introduce the directional occlusion model, which is later refined by Šoltészová et al. [2010] by removing the constraint that requires the view and light directions to coincide. Ruiz et al. [2010] use separable filtering on the GPU to efficiently evaluate occlusion at interactive rates. To enhance spatial reasoning they use the gradient of the volumetric occlusion to modulate the transfer function. Ancel et al. [2010] point out a problem with AO methods that tend to over-darken important inter-occluded features and propose a feature-driven method implemented on a GPU to overcome these issues. A similar contribution by Hernell et al. [2010] uses progressive refinement to deliver an interactive experience and quality visual appearance.

All the above methods, as well as many methods presented in [Cerezo et al. 2005; Pharr et al. 2016], enhance spatial reasoning by providing visual cues based on global illumination, though only being rough approximations. In this paper we emphasize the importance of photo-realistic rendering based on the paper by Banks and Beason [2009], which states that adoption of advanced volume visualization methods in practice is near zero. The work of Kroes et al. [2012] recognized this as a problem and proposed a general solution that would be easy to integrate into the workflow, stimulating improvement of their work. Their solution allowed for an arbitrary lighting setting and worked entirely with volumes, without any special treatment for isosurfaces. We believe that this helped to facilitate the ray casting process and thus create an interactive experience. The MCRT process in their work and in our application is using a single scattering model with Woodcock tracking to determine the scattering point, contributing to the speed of the simulation. The basic Woodcock tracking method was further improved by Szirmay-Kalos et al. [2011], and later by Novák et al. with a method called residual ratio tracking [Novák et al. 2014].

The methods presented in this paper have been implemented in JavaScript and WebGL 2.0. Some research on implementing Monte Carlo rendering methods and volume rendering methods with WebGL has already been done. Congote et al. [2011] demonstrated that using web technologies is a viable option for real-time interactive rendering of volumetric data. Their application is platform-agnostic and runs both on desktop and mobile devices. There were several attempts of volume rendering implementation on mobile devices exploiting local [Mobeen and Feng 2012; Schiewe et al. 2015] as well as cloud processing power [Lee and Nam 2014]. The solution uses server-side processing power for rendering. The implementation is based on WebGL 1.0 and is using an OpenGL extension for multiple render targets to render the parametric ranges for the rays. The visualization methods they use are based on the emission-absorption model described by Max [1995], providing little to no visual cues to enhance spatial reasoning. Different implementation strategies were proposed for volume rendering implementation on mobile devices by Noguera and Jiménez [2012] who also presents an overview of mobile volume rendering approaches in [Noguera and Jimenez 2016]. Hachisuka [2015] implement a Monte Carlo method using OpenGL shading language (GLSL). Their solution, which they describe as being able to "potentially run on web browsers via WebGL", is intended to be platform-independent, but not platform-agnostic. While being a good reference on stochastic methods in GLSL, their renderer is only capable of rendering solid surfaces, but not participating media. A working WebGL path tracing implementation by Evan Wallace is available at: madebyevan.com/webgl-path-tracing/.

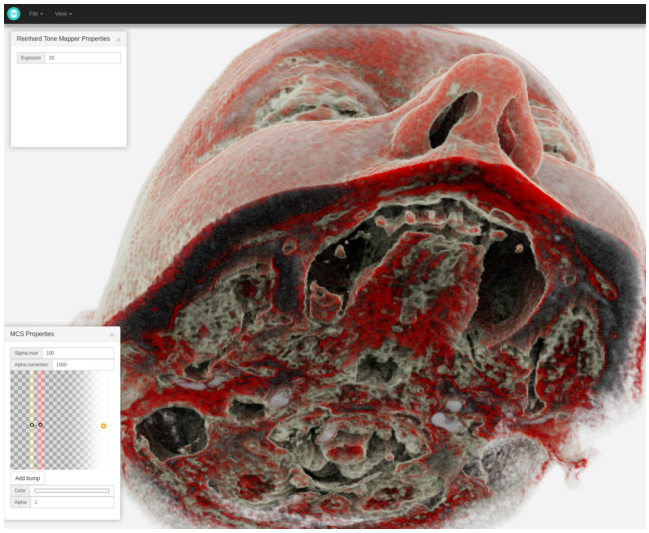

Figure 2: Our application running on a desktop device in Google Chrome.

3 METHODS

In this paper we combine a state-of-the-art Monte Carlo volume rendering method with modern web technologies to deliver an application capable of interactive exploration of volumetric data. By using a stochastic approach our application is able to render progressively refined photo-realistic images in interactive frame rates. The application is developed in JavaScript and HTML5, and is accessing the graphics hardware through the `<canvas>` element and the WebGL 2.0 API. Bootstrap and jQuery libraries are used for the graphical user interface, and no libraries are used for WebGL, to have total control over the performance of our application. Figure 2 shows a screen shot of the application running on a desktop computer.

The rendering part of our application is handled by a rendering context, which holds the WebGL context along with its data that is independent of any concrete rendering specifics, such as volume data and transfer functions. These data can be supplied to the various stages of the rendering pipeline, as described below. The rendering logic is divided into two stages: *volume rendering* and *post-processing*, producing HDR and LDR images, respectively. This makes it easy to separate any post-processing into several steps, leaving the volume rendering stage to work with HDR data only. It is important to note that WebGL 1.0 did not support floating point textures in its core profile, but rather as an extension

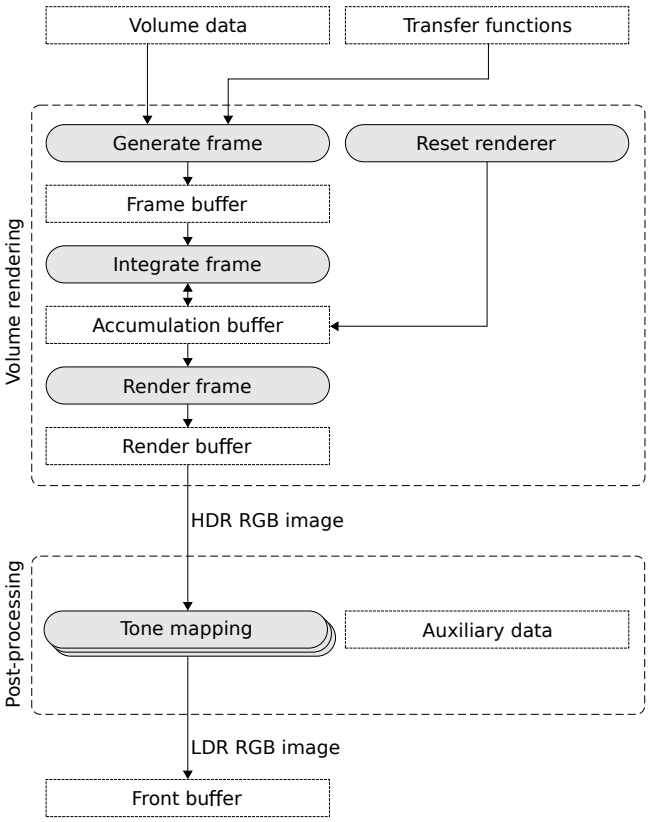

Figure 3: The rendering pipeline. Volume data and transfer functions are passed into the volume rendering stage along with other scene parameters, and the resulting HDR RGB image is sent to the post-processing stage to produce a LDR RGB image.

`OES_texture_float`, which had to be enabled during runtime. WebGL 2.0 provides support for floating textures in its core profile, although rendering to a floating point frame buffer is still available only as an extension `EXT_color_buffer_float`, upon which our application relies heavily. Both volume data and transfer functions are stored in floating point format on the GPU.

3.1 The rendering pipeline

The rendering pipeline is shown schematically in Figure 3. The first stage of the rendering pipeline is the volume rendering stage, which is represented by an abstract volume renderer. The steps involved in the rendering process are designed with interactivity in mind, so they are especially convenient to use with a MCRT renderer. The rendering process is expected to be progressive, iteratively improving the rendered frame. The methods involved are as follows:

(1) Generate frame
(2) Integrate frame
(3) Render frame
(4) (Reset renderer)

In the beginning of each iteration a frame is generated using the supplied volume data, transfer functions, camera and scene parameters. This is where the rays are generated and propagated through the volume, so it is usually the most computationally expensive step and has to be carefully implemented to allow for real-time execution. The generated frame is sent to the integrator, where it is used to update the accumulated data. The final step is to render the accumulated data into the render buffer, which is the output of the renderer. The reset step in the parentheses is used after changing the parameters that are passed into the generation step, to reset the accumulated data and prepare the renderer for a fresh run of the pipeline.

Each implementation of a volume renderer has to implement all of the above steps and provide parameters for the creation of the frame buffers used within the WebGL context. These buffers hold the generated frame, the accumulated data, and the rendered frame, corresponding to the first three of the above steps. It is important to note that the developer implementing a renderer may decide on what data to store in the frame and accumulation buffers, but the final render buffer has to be a HDR RGB rendering, which is then passed on to the post-processing stage.

This abstract notion of a volume renderer can be used without modification to render MIP images, isosurfaces, or full-featured MCRT images. For example, a MIP renderer implementation may store a maximum value for each pixel in the accumulation buffer, and only sample the volume once per pixel for an estimate, that is used to update the accumulation buffer in the integration step. An isosurface renderer implementation may store the nearest intersection and the volume gradient at that point in the accumulation buffer and use it in the rendering step along with other scene parameters, thus acting as a deferred renderer. A MCRT renderer implementation may store HDR RGB estimates in the accumulation buffer and use the rendering step for noise reduction or other filtering. All of these renderers are included in our framework.

The HDR RGB rendering that is the output of the volume rendering stage is sent into the post-processing stage. This is essentially a sequence of any number of post-processing steps, including tone mapping and gamma correction, which may produce arbitrary intermediate images. The final rendering of this stage has to be a LDR RGB image, which is ultimately presented to the user on the screen through an HTML `<canvas>` element. For example, in our application we use this stage to perform tone mapping with a simple Reinhard tone mapper with adjustable exposure.

3.2 Monte Carlo renderer implementation

In our application we have implemented a MIP renderer and an isosurface renderer to be able to compare their performance and output to the MCRT renderer. Both MIP and isosurface renderers have been implemented in a Monte Carlo fashion as described in the previous section. MCRT renderer follows the same pattern. Our implementation of the MCRT renderer works similarly to the implementation of Kroes et al. [Kroes et al. 2012], in the sense that it uses *Woodcock tracking* [Szirmay-Kalos et al. 2011; Woodcock et al. 1965] to yield a single scattering point to facilitate lighting calculations. Compared to *ray marching* it is also unbiased. *Multiple*

Figure 4: Different transfer functions used on the same dataset. The rendering pipeline is designed to be completely interactive, requiring only a reset of the buffers to take effect. The whole system is designed to encourage dynamic data exploration.

importance sampling (MIS) can be used to direct the scattering paths towards the light sources.

First, the camera parameters are used to calculate the ray parametrization for each pixel. This parametrization is then used along with the transformation of the volume bounding box to produce a parametrization range for the ray. Rays that do not intersect the volume bounding box are only used to sample the environment map. Other rays are then propagated through the volume with Woodcock tracking to find a scattering point and a light source. If the scattering point is not found, the environment map is sampled, otherwise the phase function is sampled to produce a scattering path. Woodcock tracking is used again to estimate the transmittance of the volume on this path. This method converges over time to a solution with single scattering. Furthermore, all scene parameters, including camera position, lighting setup and transfer functions, are completely interactive, requiring only a restart of the rendering pipeline to take effect. Figure 4 shows different transfer functions used on the same dataset.

All light interactions with the volume are calculated in floating point format and stored in WebGL frame buffers with the extension `EXT_color_buffer_float` enabled. This enables us to use the equations for radiative transfer directly, without any modifications and significant loss of precision. The transfer function and environment map are also stored as floating point textures on the GPU, but this requires no special treatment in WebGL 2.0, where floating point texture formats are supported by the core specification, as opposed to the extension `OES_texture_float` needed in WebGL 1.0.

The HDR RGB rendering of the volume renderer's render step is passed on by the rendering context to the post-processing step, where it is further processed and adjusted for better perception.

3.3 Post-processing

Our implementation allows for the post-processing stage to consist of an arbitrary number of steps, each transforming the image in a certain way to enhance perception. The input to this stage comes from the volme renderer as a HDR RGB image. The steps involved in this stage store the intermediate images in various formats, but the last step must always output a LDR RGB image, so that it can be displayed on the screen. This stage may also contain various filtering or noise-reduction steps to aid in the convergence of the MCRT renderer.

In our current rendering pipeline we use a single step in this stage, which is a tone mapper described by Reinhard et al. [2002] to compress the dynamic range of the output of the volume rendering stage. The method described in [Kroes et al. 2012] uses an additional gamma correction step after the tone mapping step, which we don't use in our current implementation.

4 EVALUATION AND RESULTS

The developed application (https://github.com/terier/vpt) was evaluated regarding browser support, convergence rate and speed of execution. It should be noted that we have only evaluated the rendering performance with all the required data already stored on the graphics device, since this work is not concerned with data streaming over a network or from a secondary storage. The evaluation was run on two different devices, a laptop with an Intel HD graphics 530 integrated graphics card, and a smartphone with Adreno 430 chipset.

4.1 Browser support

Browser support was analyzed using the statistical data acquired from the website https://webglstats.com, which gathers the data about WebGL implementations on users' devices from a few popular websites. While WebGL 1.0 was supported on 97% of devices, WebGL 2.0 was supported on 66% of devices. Additionally, we checked the support for two extensions used in our implementation: `EXT_color_buffer_float` for storing HDR images, which was supported on 91% of the devices, and `WEBGL_lose_context` for detecting the loss or termination of the context by the browser, which was supported on almost all devices (~100%). For evaluating the speed of execution of a render frame WebGL 2.0 provides an extension `EXT_disjoint_timer_query_webgl2`, which is available on 71% of devices, but since it's not supported on our test devices, we had to use a combination of `performance.now()` and `gl.finish()`.

4.2 Convergence rate

We evaluated the convergence rate of our implementation of the radiative transfer equation with single scattering by calculating the peak signal-to-noise ratio (PSNR) for frames of different sizes

depending on the number of method iterations. Three different framebuffer sizes were used: 256 × 256, 512 × 512 and 1024 × 1024 pixels. For the purpose of PSNR calculation we took the image after 1000 iterations as the gold standard. The results are presented in Figure 5. Due to the higher number of pixels, the PSNR values are comparable between different image sizes. The size of the image frame directly influences the speed of execution of an individual iteration. However, we cannot use PSNR to evaluate spatial image precision, since we calculate it for each pixel independently. The implemented method converges fast and yields a good result image in 20–30 iterations, but due to the nature of the Monte Carlo method our renderer suffers from diminishing returns over time, theoretically converging at the rate of $O(n^{-\frac{1}{2}})$.

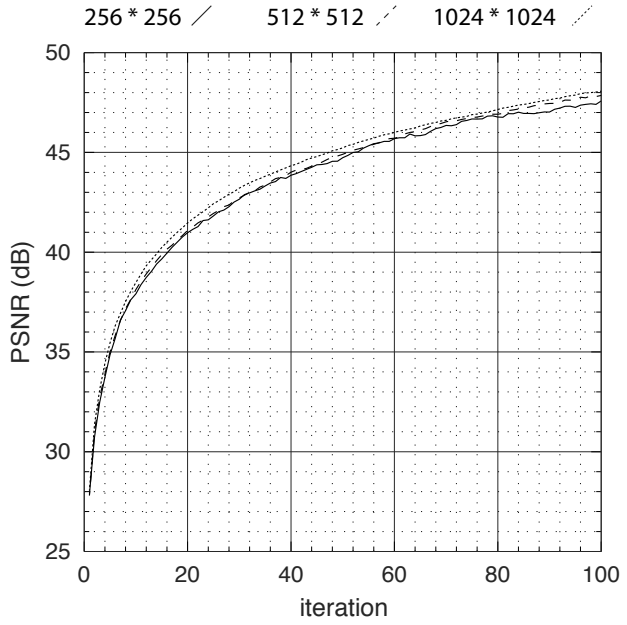

Figure 5: PSNR values depending on the number of iterations for different framebuffer sizes: 256 × 256 (left), 512 × 512 (center) and 1024 × 1024 (right) for a volume of size 128 × 128 × 128.

4.3 Speed

Iterative pipeline execution is based on `requestAnimationFrame`, which is synchronized to the refresh rate of the display, meaning we can only achieve 60 iterations per second with this method. We improved this by allowing more than one iteration inside a single render pass. On the other hand, when using a slow device we cannot avoid stalling the main thread on long running iterations, since there currently exists no generally available method for creating a WebGL context in a separate thread. We measured the average execution time of a single pipeline iteration with varying framebuffer and volume sizes. The measurements, performed in Google Chrome for Linux 56 (Desktop) and Google Chrome for Android 58 (Smartphone), are shown in table 1. It is evident that our implementation is fast enough for interactive use even on mobile devices. For a small enough framebuffer and volume we hit the limit of `requestAnimationFrame` execution rate, so the numbers are bottom-clamped at 16 ms.

Table 1: Average execution time of a single pipeline iteration with respect to the volume size at a fixed framebuffer size of 512 × 512 (left), and with respect to the framebuffer size at a fixed volume size of 128 × 128 × 128 (right).

Device	64^3	128^3	256^3	64^2	128^2	256^2
Smartphone	28 ms	65 ms	92 ms	35 ms	65 ms	98 ms
Laptop	16 ms	28 ms	45 ms	16 ms	28 ms	55 ms

5 CONCLUSIONS

In this paper we present our work on a real-time interactive platform-agnostic DVR application, capable of producing photo-realistic images. It is based on solving the radiance transport equations with MCRT and physically based shading. The application runs in any web browser supporting WebGL 2.0 with float buffer extensions, which means it can run on both desktop and mobile devices. To our knowledge, no other attempts have been made yet to deliver all of the above in one web application. We present an extensible JavaScript and WebGL 2.0 framework for developing various rendering methods in a modular fashion. Our proposed solution is capable of rendering progressively and interactively, with no preprocessing needed when changing the lighting scenario or the transfer functions.

Future improvements may include further optimizations of the framework itself and the renderers build on top of it. Fong et al. [2017] provides an exhaustive and up-to-date source for the current state of the art in production volume rendering, which will serve as a valuable reference for further development to bring these ideas into a web-based implementation. Initial improvements may range from multiple importance sampling, as described by Kroes et al. [2012], to ray propagation improvements with residual ratio tracking, presented by Novák et al. [2014]. Better camera models with optimizations in the ray generation step would be next in queue. GUI and UX improvements with other production-level necessities should also be in place. Currently our framework does not support bigger volumetric assets, which require dynamic streaming from disk or network.

We believe that our application represents a step in the right direction to bring platform-agnostic tools for volumetric data exploration to end users without any difficult installation procedures or platform restrictions. We hope that the industry sees this contribution as a viable alternative to native applications.

REFERENCES

Alexandre Ancel, Jean-Michel Dischler, and Catherine Mongenet. 2010. Feature-driven ambient occlusion for direct volume rendering. In *Proceedings of the 8th IEEE/EG international conference on Volume Graphics*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 85–92. https://doi.org/10.2312/VG/VG10/085-092

M. Balsa Rodríguez, E. Gobbetti, J. A. Iglesias Guitián, M. Makhinya, F. Marton, R. Pajarola, and S. K. Suter. 2014. State-of-the-art in compressed GPU-based direct volume rendering. *Computer Graphics Forum* 33, 6 (2014), 77–100. https://doi.org/10.1111/cgf.12280

D.C. Banks and K. Beason. 2009. Decoupling Illumination from Isosurface Generation Using 4D Light Transport. *IEEE Transactions on Visualization and Computer Graphics* 15, 6 (11 2009), 1595–1602. https://doi.org/10.1109/TVCG.2009.137

Brian A. Barsky, Daniel R. Horn, Stanley A. Klein, Jeffrey A. Pang, and Meng Yu. 2003. Camera Models and Optical Systems Used in Computer Graphics: Part II, Image-Based Techniques. In *Computational Science and Its Applications—ICCSA 2003*. Springer, Berlin, Heidelberg, 256–265. https://doi.org/10.1007/3-540-44842-X{_}27

Uwe Behrens and Ralf Ratering. 1998. Adding shadows to a texture-based volume renderer. In *Proceedings of the 1998 IEEE symposium on Volume visualization - VVS '98*. ACM Press, New York, New York, USA, 39–46. https://doi.org/10.1145/288126.288149

Johanna Beyer, Markus Hadwiger, and Hanspeter Pfister. 2015. State-of-the-Art in GPU-Based Large-Scale Volume Visualization. *Computer Graphics Forum* 34, 8 (2015), 13–37. https://doi.org/10.1111/cgf.12605

Eva Cerezo, Frederic Pérez, Xavier Pueyo, Francisco J. Seron, and François X. Sillion. 2005. A survey on participating media rendering techniques. *The Visual Computer* 21, 5 (Jun 2005), 303–328. https://doi.org/10.1007/s00371-005-0287-1

John Congote, Alvaro Segura, Luis Kabongo, Aitor Moreno, Jorge Posada, and Oscar Ruiz. 2011. Interactive visualization of volumetric data with WebGL in real-time. In *Proceedings of the 16th International Conference on 3D Web Technology - Web3D '11*. ACM Press, New York, New York, USA, 137. https://doi.org/10.1145/2010425.2010449

Tomáš Davidovič, Jaroslav Křivánek, Miloš Hašan, and Philipp Slusallek. 2014. Progressive Light Transport Simulation on the GPU. *ACM Transactions on Graphics* 33, 3 (6 2014), 1–19. https://doi.org/10.1145/2602144

Julian Fong, Magnus Wrenninge, Christopher Kulla, and Ralf Habel. 2017. Production volume rendering. In *ACM SIGGRAPH 2017 Courses on - SIGGRAPH '17*. ACM Press, New York, New York, USA, 1–79. https://doi.org/10.1145/3084873.3084907

Toshiya Hachisuka. 2015. Implementing a Photorealistic Rendering System using GLSL. *arXiv* abs/1505.06022 (5 2015), 1–4. http://arxiv.org/abs/1505.06022

Markus Hadwiger, Andrea Kratz, Christian Sigg, and Katja Bühler. 2006. GPU-accelerated deep shadow maps for direct volume rendering. In *Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware - GH '06*. ACM Press, New York, New York, USA, 49. https://doi.org/10.1145/1283900.1283908

Markus Hadwiger, Patric Ljung, Christof Rezk Salama, and Timo Ropinski. 2009. Advanced illumination techniques for GPU-based volume raycasting. In *ACM SIGGRAPH 2009 Courses on - SIGGRAPH '09*. ACM Press, New York, New York, USA, 1–166. https://doi.org/10.1145/1667239.1667241

Frida Hernell, Patric Ljung, and Anders Ynnerman. 2010. Local ambient occlusion in direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics* 16, 4 (2010), 548–559.

Daniel Jönsson, Erik Sundén, Anders Ynnerman, and Timo Ropinski. 2014. A Survey of Volumetric Illumination Techniques for Interactive Volume Rendering. *Computer Graphics Forum* 33, 1 (2 2014), 27–51. https://doi.org/10.1111/cgf.12252

James T. Kajiya. 1986. The rendering equation. *ACM SIGGRAPH Computer Graphics* 20, 4 (aug 1986), 143–150.

Craig Kolb, Don Mitchell, and Pat Hanrahan. 1995. A Realistic Camera Model for Computer Graphics. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. ACM, New York, NY, USA, 317–324. https://doi.org/10.1145218380.218463

Thomas Kroes, Frits H. Post, and Charl P. Botha. 2012. Exposure Render: An Interactive Photo-Realistic Volume Rendering Framework. *PLoS ONE* 7, 7 (7 2012), e38586. https://doi.org/10.1371/journal.pone.0038586

Eric P. Lafortune and Yves D. Willems. 1996. Rendering Participating Media with Bidirectional Path Tracing. In *Proceedings of the Eurographics Workshop on Rendering Techniques '96*. Springer-Verlag, London, UK, UK, 91–100.

Woongkyu Lee and Doohee Nam. 2014. Volume Rendering Architecture of Mobile Medical Image using Cloud Computing. *The Journal of The Institute of Internet, Broadcasting and Communication* 14, 4 (2014), 101–106.

Žiga Lesar, Ciril Bohak, and Matija Marolt. 2015. Evaluation of angiogram visualization methods for fast and reliable aneurysm diagnosis. In *Progress in Biomedical Optics and Imaging - Proceedings of SPIE*, Claudia R. Mello-Thoms and Matthew A. Kupinski (Eds.), Vol. 9416. SPIE, Orlando, Florida, United States, 9416. https://doi.org/10.1117/12.2082179

Marc Levoy. 1990. A hybrid ray tracer for rendering polygon and volume data. *IEEE Computer Graphics and Applications* 10, 2 (1990), 33–40.

Florian Lindemann and Timo Ropinski. 2011. About the Influence of Illumination Models on Image Comprehension in Direct Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (12 2011), 1922–1931. https://doi.org/10.1109/TVCG.2011.161

Tom Lokovic and Eric Veach. 2000. Deep shadow maps. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press, New York, New York, USA, 385–392. https://doi.org/10.1145/344779.344958

Nelson Max. 1995. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics* 1, 2 (6 1995), 99–108. https://doi.org/10.1109/2945.468400

Movania Muhammad Mobeen and Lin Feng. 2012. Ubiquitous Medical Volume Rendering on Mobile Devices. In *International Conference on Information Society (i-Society 2012)*. IEEE, London, UK, 93–98.

José M. Noguera and Juan-roberto Jiménez. 2012. Visualization of very large 3D volumes on mobile devices and WebGL. In *WSCG Communication Proceedings*. Václav Skala-UNION Agency, Plzen, Czech Republic, 105–112.

Jose M. Noguera and J. Roberto Jimenez. 2016. Mobile Volume Rendering: Past, Present and Future. *IEEE Transactions on Visualization and Computer Graphics* 22, 2 (feb 2016), 1164–1178.

Jan Novák, Andrew Selle, and Wojciech Jarosz. 2014. Residual ratio tracking for estimating attenuation in participating media. *ACM Transactions on Graphics* 33, 6 (11 2014), 1–11. https://doi.org/10.1145/2661229.2661292

Matt Pharr, Jakob Wenzel, and Greg Humphreys. 2016. *Physically based rendering* (3 ed.). Morgan Kaufmann, San Francisco, CA, USA. 1266 pages.

Erik Reinhard, Michael Stark, Peter Shirley, and James Ferwerda. 2002. Photographic tone reproduction for digital images. *ACM Transactions on Graphics* 21, 3 (7 2002), 267–276. https://doi.org/10.1145/566654.566575

Marc Ruiz, Lázló Szirmay-Kalos, Tamás Umenhoffer, Imma Boada, Miquel Feixas, and Mateu Sbert. 2010. Volumetric ambient occlusion for volumetric models. *The Visual Computer* 26, 6-8 (6 2010), 687–695. https://doi.org/10.1007/s00371-010-0497-z

Alexander Schiewe, Mario Anstoots, and Jens Krüger. 2015. State of the Art in Mobile Volume Rendering on iOS Devices. In *Eurographics Conference on Visualization (EuroVis) - Short Papers*, E. Bertini, J. Kennedy, and E. Puppo (Eds.). The Eurographics Association, Cagliari, Italia, 139–143.

Mathias Schott, Vincent Pegoraro, Charles Hansen, Kévin Boulanger, and Kadi Bouatouch. 2009. A Directional Occlusion Shading Model for Interactive Direct Volume Rendering. *Computer Graphics Forum* 28, 3 (6 2009), 855–862. https://doi.org/10.1111/j.1467-8659.2009.01464.x

Veronika Šoltészová, Daniel Patel, Stefan Bruckner, and Ivan Viola. 2010. A multidirectional occlusion shading model for direct volume rendering. *Computer Graphics Forum* 29, 3 (8 2010), 883–891. https://doi.org/10.1111/j.1467-8659.2009.01695.x

László Szirmay-Kalos, Balázs Tóth, and Milán Magdics. 2011. Free Path Sampling in High Resolution Inhomogeneous Participating Media. *Computer Graphics Forum* 30, 1 (3 2011), 85–97. https://doi.org/10.1111/j.1467-8659.2010.01831.x

E Woodcock, T Murphy, P Hemmings, and S Longworth. 1965. Techniques used in the GEM code for Monte Carlo neutronics calculations in reactors and other systems of complex geometry. In *Proc. Conf. Applications of Computing Methods to Reactor Problems*. Argonne, Ill. : Argonne National Laboratory, Physics Division, Springfield, Virginia, USA, 557–579.

S. Zhukov, A. Iones, and G. Kronin. 1998. An ambient light illumination model. In *Rendering Techniques '98*, George Drettakis and Nelson Max (Eds.). Springer Vienna, Vienna, 45–55. https://doi.org/10.1007/978-3-7091-6453-2{_}5