

# Physically-based Path Tracer using WebGPU and OpenPBR

Simon Stucki

ZHAW Zurich University of Applied Sciences  
Winterthur, Switzerland  
stucks1@students.zhaw.ch

Philipp Ackermann

ZHAW Zurich University of Applied Sciences  
Winterthur, Switzerland  
philipp.ackermann@zhaw.ch



**Figure 1:** Path traced renderings of pushbutton CAD models. From left to right: side view with metallic reflection, rear view exhibiting red color bleeding, rear view with ambient occlusion, and pushbutton mounted on a metallic surface, including the reflection of the Stanford bunny model [Turk and Levoy 1994] positioned in the scene.

## ABSTRACT

This work presents a web-based, open-source path tracer for rendering physically-based 3D scenes using WebGPU and the OpenPBR surface shading model. While rasterization has been the dominant real-time rendering technique on the web since WebGL's introduction in 2011, it struggles with global illumination. This necessitates more complex techniques, often relying on pregenerated artifacts to attain the desired level of visual fidelity. Path tracing inherently addresses these limitations but at the cost of increased rendering time. Our work focuses on industrial applications where highly customizable products are common and real-time performance is not critical. We leverage WebGPU to implement path tracing on the web, integrating the OpenPBR standard for physically-based material representation. The result is a near real-time path tracer capable of rendering high-fidelity 3D scenes directly in web browsers, eliminating the need for pregenerated assets. Our implementation demonstrates the potential of WebGPU for advanced rendering techniques and opens new possibilities for web-based 3D visualization in industrial applications.

## CCS CONCEPTS

- Information systems → Web applications; • Computing methodologies → Ray tracing; Reflectance modeling; • Software and its engineering → Software creation and management.

## KEYWORDS

web path tracer, physically-based rendering, WebGPU, OpenPBR

## 1 INTRODUCTION

Rasterization is a rendering technique that projects 3D scene geometry onto a 2D plane. Historically, the technique has been widely adopted in real-time rendering due to its efficiency. However, rasterization has limitations in achieving photorealism. One of the main limitations is the lack of support for global illumination. These phenomena can be observed in objects like mirrors or metallic surfaces. Effects such as refraction, reflection, and ambient occlusion require additional techniques.

Various methods have been developed to address these limitations, including pre-baked shadow, environment [Greene 1986] and light maps; screen space reflections (SSR) [Stachowiak 2015]; screen space ambient occlusion (SSAO) [Bavoil and Sainz 2008]; and screen space directional occlusion (SSDO) [Ritschel et al. 2009].

These approaches induce complexity, may need to be computed at the assembly level, and can be computationally expensive. An alternative rendering technique resembles reality more closely and inherently alleviates these limitations: ray tracing.

### 1.1 Ray Tracing

Ray tracing is a powerful technique for rendering photorealistic scenes including global illumination. Historically, ray tracing has been mainly used in offline rendering due to its computational complexity. The theory behind ray tracing techniques is well established [Whitted 2020].

Kajiya defined the rendering equation in 1986, coining the term path tracing. Generally, path tracing can be seen as an integration of the rendering equation, which is defined as:

$$I(x, x') = g(x, x')[\epsilon(x, x') + \int_S p(x, x', x'') I(x', x'') dx'']$$

where  $I(x, x')$  is the radiance from point  $x$  to point  $x'$ .  $x$  for example being the camera position and  $x'$  the intersected object.  $g(x, x')$  is a geometry term determining how much light is transmitted. This depends on the distance and possibly occlusions such as transparent surfaces.  $\epsilon(x, x')$  is the emitted radiance, generally used for light sources. The integral term is taken over  $S = \cup S_i$  which is the union of all surfaces.  $p(x, x', x'')$  is the bidirectional reflection function, which describes how light from all possible directions  $x''$  is reflected at the surface. [Kajiya 1986]

Ray tracing techniques focus on optimizing the rendering equation. Research in the 1990s focused on efficient light transport techniques such as bidirectional light transport and Metropolis light transport [Veach 1998]. Concurrently, production offline renderers like Blue Moon Rendering Tools (BMRT) were developed [Gritz and Hahn 1996].

Recent hardware advancements, including dedicated ray tracing acceleration [Dally et al. 2021], have made real-time ray tracing feasible, sparking increased research. Notable developments include reservoir-based spatio-temporal importance resampling (ReSTIR) [Bitterli et al. 2020] and subsequent improvements [Lin et al. 2022; Ouyang et al. 2021].

## 2 USE CASES

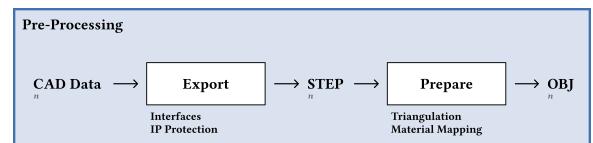
E-commerce represents a key use case for product renderings. Traditional catalogues struggle with highly configurable products. Computer graphics addresses this challenge through product configurators for virtual assembly and visualization.

Leveraging existing CAD models, prevalent in mechanical engineering and product design, for end-user applications offers a significant advantage. These models contain geometric and material information, which eliminates the need for redundant 3D models for marketing purposes. One example is EAO, which manufactures highly customizable industrial pushbuttons and operator panels as visualized in Figure 1. Due to the nature of the product, the number of possible assemblies grows almost exponentially with the number of components. To facilitate the configuration process, a web-based configurator is optimal because it can be used on a large variety of devices without having to install additional software. Given the company's priority for photorealistic renderings of the assemblies, using ray tracing techniques is a compelling choice.

Pre-rendering all product configurations is theoretically possible for a finite number of configurations, but computationally expensive. If this is not feasible, real-time rendering is an alternative. For real-time rendering, one option is remote rendering [Shi and Hsu 2015], which employs a server to render the scene and stream the visualization to the browser. The main drawbacks of this approach are network latency, reliance on network stability, as well as operational cost for the server infrastructure, which frequently requires dedicated GPUs for rendering. Another option is client-side rendering. Most frequently, rasterization approaches are used

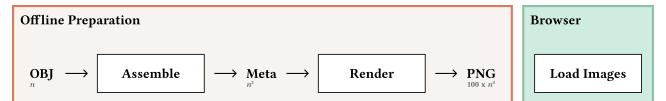
for web-based renderings. However, due to the limitations in global illumination effects, the need for a real-time, client-side ray tracing solution for the web becomes apparent when considering the use case.

CAD models require pre-processing as described in Figure 2. This step involves surface triangulation, potential further refinement using algorithms used to generate level of detail (LOD) artifacts [Luebke 2003], and measures to protect intellectual property (IP) rights by removing proprietary data. While numerous CAD formats include material information, it is often unsuitable for rendering. To address this, a material mapping can be defined, translating CAD materials to a suitable representation for the rendering pipeline.



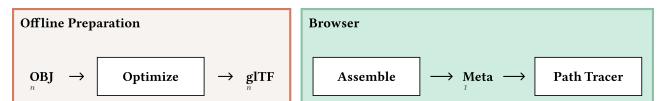
**Figure 2: A two-step pre-processing stage is employed for offline as well as real-time rendering pipelines.**

An offline rendering pipeline generates static images of the assembly, which are then displayed in the browser. This means that all possible assemblies need to be rendered and stored upfront. As the number of components increases, the number of possible combinations grows exponentially, which can lead to large amounts of storage and processing power being required, as shown in Figure 3.



**Figure 3: In an offline rendering setup, the number of images grows exponentially. The number of images increases further when offering 360° viewing.**

This is the main benefit of using a real-time rendering pipeline, as illustrated in Figure 4. The rendering is done in the browser, which means that the server only needs to provide the geometry and material information in an exchange format such as glTF. This approach is more flexible and can be used for a larger number of configurations. The amount of data to be stored and processed offline grows linearly with the number of components, independent of the number of possible configurations.



**Figure 4: A real-time rendering pipeline solely relies on having an adequate model for each component of the assembly. It does not require pre-rendering every possible configuration.**

### 3 PRIOR WORK

There are a variety of path tracers available for the web, most of which are based on WebGL. The first experiments using WebGL for path tracing were implemented as early as 2010. One such example is the Google experiment demonstrating a Cornell Box [Goral et al. 1984] with basic primitive shapes such as spheres and planes [Wallace 2011]. Since then, multiple open-source implementations for the web have been created. Some of the most widely known open-source web-based path tracers include:

- `three-gpu-pathtracer` [Johnson 2024].
- `Three.js PathTracer` [Loftis 2024].
- `dspbr-pt` [Sdorra and Häußler 2022].

WebGPU is a new web standard which, unlike WebGL, is no longer based on OpenGL. It is designed to be more efficient and flexible than its predecessor. Flexibility is mainly exhibited through its support for general-purpose GPU (GPGPU) computations by design. Common 3D engines based on rasterization, including Babylon.js [Babylon.js 2022], PlayCanvas [Valigursky 2023], Three.js [Three.js 2024], and Unity [Craven et al. 2023], have officially announced support for or started working on WebGPU.

Various applications of WebGPU have been investigated in recent years. Examples include DynamicalJS, a framework for visualizing graphs [Dotson 2022]; RenderCore, a research-oriented rendering engine [Bohak et al. 2024]; and demonstrations of how to use WebGPU for client-side data aggregation [Kimmersdorfer et al. 2023].

Research on the performance comparison between existing 3D engines and WebGPU engines has been conducted as part of the development of FusionRender, which concluded that measurable performance gains can be achieved by using WebGPU, but only when effectively leveraging its novel design principles [Bi et al. 2024]. Similar findings have emerged from other independent studies, demonstrating that WebGPU can be faster than WebGL [Chickerur et al. 2024; Fransson and Hermansson 2023; Usta 2024].

In light of these findings, WebGPU presents a transformative opportunity. It is particularly well-suited for the development of a new real-time path tracing library for the web.

### 4 PHYSICALLY BASED RENDERING

Defining the geometry is one part of the equation. Another major part is defining materials. Materials determine how a surface interacts with light. The core idea of physically based rendering (PBR) is to simulate the interaction of light using physical models. Instead of fine-tuning parameters for a specific look and feel and having to adjust based on desired lighting situations, PBR aims to define a material that behaves consistently in different lighting situations. One prominent example of a PBR system is pbrt, an open-source renderer with an associated literate programming book [Pharr et al. 2023].

Based on these principles, different material shading models have been developed. MaterialX is an open standard for representing materials using a node-based system. Originally developed by Lucasfilm in 2012, it has since been adopted by the Academy Software Foundation as an open standard. While not strictly limited to PBR, the standard provides a wide range of features for describing physically based materials [Harrysson et al. 2019].

OpenPBR, also hosted by the Academy Software Foundation as an open standard, differs from MaterialX by providing an uber shader approach rather than a node-based system. The uber shader approach is defined by a fixed set of inputs which can be adjusted but does not support custom node graphs like those in MaterialX. OpenPBR combines aspects of Autodesk Standard Surface and Adobe Standard Material. Its parameters allow for configuring metalness, glossy-diffuse, subsurface, coat, fuzz, emission, and more. To date, the standard has a reference implementation in MaterialX. [Andersson et al. 2024]

The emphasis on high-quality PBR materials is advantageous for the industrial use cases described in this paper. Furthermore, the goal of interoperability within the broader ecosystem strengthens the case for using OpenPBR in our implementation.

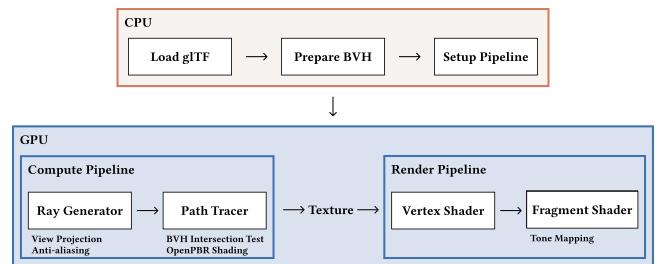
### 5 RESULTS

The work presented in this paper introduces a web-based path tracer that utilizes WebGPU for near real-time rendering. The renderer supports loading glTF models and allows for the configuration of materials using OpenPBR parameters.

The renderer is available as an open-source project with extensive documentation at <https://www.github.com/StuckiSimon/strahl>. The library is published on the npm registry as `strahl`.

#### 5.1 Implementation

Figure 5 illustrates the procedure of the path tracer. Scene preparation is performed on the CPU. This setup needs to be done once per visualization. Subsequent sampling of the scene is carried out repeatedly on the GPU, which constitutes the most computationally intensive part of the process.



**Figure 5: Path Tracer Pipeline, distinguishing GPU and CPU tasks. The stages of the compute pipeline and render pipeline are executed sequentially.**

**5.1.1 CPU Setup.** The CPU processing begins by loading the glTF model. As an acceleration structure for ray intersection tests, the renderer employs a bounding volume hierarchy (BVH) based on axis-aligned bounding boxes (AABB). The core concept is to group adjacent objects in a bounding volume and structure the hierarchy so that all child elements of a node are contained in the bounding volume of the parent node. This hierarchical structure allows for early rejection of branches not intersected by the ray, reducing the number of intersection tests required per ray cast from  $O(n)$  to  $O(\log(n))$ . Once the BVH is constructed, the scene is prepared for rendering and the pipeline is set up.

**5.1.2 Ray Generator.** The ray generator is responsible for casting rays into the scene according to the view projection. Based on the use case, the generator uses perspective projection. The path tracer employs a backward ray tracing approach, tracing rays from the camera into the scene. To mitigate aliasing effects, rays for the same pixel are slightly varied across multiple samples. Parallelization is achieved by computing adjacent pixels concurrently across multiple GPU cores.

To sample distributions, the path tracer selects random values using a random number generator (RNG). Several suitable pseudorandom number generators have been considered, including Mersenne Twister [Matsumoto and Nishimura 1998] and Xorshifts such as the method described by Marsaglia [Marsaglia 2003]. Our renderer uses a permuted congruential generator (PCG) as the RNG due to its good performance and quality [O'Neill 2014].

**5.1.3 Path Tracer.** Afterwards, the core component of the path tracer recursively traces the rays. The procedure checks for intersections using the BVH and calculates the radiance contribution of the intersected objects based on the OpenPBR material parameters. The OpenPBR standard employs an uber shader approach, which provides a balance between flexibility and performance. The OpenPBR support is focused on the most relevant features for industrial use cases, permitting realistic rendering of metallic, glossy, and diffuse surfaces.

Importance sampling is used to reduce the number of samples required for accurate results by prioritizing sample directions that contribute more significantly to the final image. For highly metallic surfaces, sampling is steered towards the reflection direction, while for diffuse surfaces, sampling is steered towards the surface normal.

The path tracer utilizes Russian roulette for probabilistic path termination, selectively discarding paths with low expected radiance contribution. The samples are then averaged to compute the final pixel color.

**5.1.4 Render Pipeline.** The output of the path tracing compute shader is a texture, which is then passed to a rasterizer. The rasterizer renders the texture to the canvas using a fullscreen quad consisting of two triangles. Additionally, it applies tone mapping based on the Khronos PBR neutral tone mapper [Lalish 2024].

After each sampling iteration, the current result is visualized, resulting in a progressive rendering approach.

## 5.2 Performance

Real-time performance is a key aspect of web-based applications. The renderer can visualize complex models in near real-time, requiring approximately 10 ms per sample on suitable hardware, taking approximately two seconds to render an image with 200 samples. The number of samples needed for a high-quality image depends on the scene complexity; effects such as reflections require more samples to converge.

As indicated by Table 1, CPU performance for BVH construction varies significantly based on the model's complexity. Table 2 shows that GPU performance for path tracing is more stable across different model complexities and constitutes the most computationally expensive part of the process.

## 6 DISCUSSION

The defined approach provides a viable solution to address challenges encountered in industrial applications. The ability to render complex assemblies in real-time without the need for pregenerated artifacts facilitates the use of CAD models with manifold assembly configurations and customer-specific materials. Product configurators using WebGPU can present product variety in high visual fidelity while enabling an interactive end-user experience with arbitrary viewing angles.

Additionally, operating costs can be reduced by eliminating the need to host a rendering pipeline or remote rendering service.

### 6.1 Real-Time Rendering

Path tracing is computationally expensive and requires multiple samples per pixel to achieve accurate results. Consequently, the sampling process is noticeable during interactions with the scene. One technique to improve perceived interaction quality is to overlay the rendering with a rasterization preview during interactions.

To reduce the number of samples required for higher quality renderings, techniques like neural radiance caching (NRC) [Müller et al. 2021] or ReSTIR [Bitterli et al. 2020] could be employed in future work. Additionally, denoising algorithms applied as post-processing steps could enhance the quality of the results. Options include blockwise multi-order feature regression (BMFR) [Koskela et al. 2019] and Open Image Denoise (OIDN) [Áfra 2024].

Further improvements can be achieved by aligning glTF PBR with OpenPBR, focusing on real-time rendering. This alignment could reduce the computational resources required for surface shading.

### 6.2 Production Readiness of WebGPU

WebGPU's production readiness remains limited due to incomplete support in Safari and Firefox. Both browsers have announced plans for implementation [Mozilla 2023; Wyrzykowski 2023]. Thanks to the extensive conformance test suite [W3C group for GPU web standards 2024], it is more likely that the different implementations will be compatible with each other.

The main browser supporting WebGPU to date is Chrome, which shipped WebGPU for general use on desktops in May 2023 [Google 2023]. Since January 2024, WebGPU has also been supported on modern Android devices [Google 2024]. This makes it straightforward to use WebGPU on most modern devices, with the notable exception of Apple iOS and iPadOS devices.

### 6.3 Future Work

Future work may include supporting additional OpenPBR features, implementing denoising techniques, integrating rendering techniques such as depth of field and volumetric path tracing, enhancing performance, and comparing the renderer with alternative web-based path tracers. The open-source nature of the project facilitates extension and serves as inspiration for other initiatives.

## ACKNOWLEDGMENTS

We thank the reviewers for their valuable feedback and suggestions. We also thank EAO and Intelliact for permitting the use of production CAD models, enabling the verification of a real-world use case.

## REFERENCES

- Attila T. Áfra. 2024. Intel® Open Image Denoise. <https://www.openimagedenoise.org>.
- Zap Andersson, Paul Edmondson, Julien Guertault, Adrien Herubel, Alan King, Peter Kutz, Andréa Machizaud, Jamie Portsmouth, Frédéric Servant, and Jonathan Stone. 2024. *OpenPBR Surface Specification*. Technical Report. Academy Software Foundation (ASWF). <https://academysoftwarefoundation.github.io/OpenPBR/Babylon.js>. 2022. *WebGPU Support*. Retrieved July 18, 2024 from <https://github.com/BabylonJS/Babylon.js/issues/6443>
- Louis Bavoil and Miguel Sainz. 2008. Screen space ambient occlusion. *NVIDIA developer information*. <http://developer.nvidia.com> 6, 2 (2008).
- Weichen Bi, Yun Ma, Yudong Han, Yifan Chen, Deyu Tian, and Jiaqi Du. 2024. FusionRender: Harnessing WebGPU™ Power for Enhanced Graphics Performance on Web Browsers. In *Proceedings of the ACM on Web Conference 2024* (Singapore, Singapore) (WWW '24). Association for Computing Machinery, New York, NY, USA, 2890–2901. <https://doi.org/10.1145/3589334.3645395>
- Benedikt Bitterli, Chris Wyman, Matt Pharr, Peter Shirley, Aaron Lefohn, and Wojciech Jarosz. 2020. Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting. *ACM Trans. Graph.* 39, 4, Article 148 (aug 2020), 17 pages. <https://doi.org/10.1145/3386569.3392481>
- Ciril Bohak, Dmytro Kovalskyi, Sergey Linev, Alja Mrak Tadel, Sébastien Strban, Matevž Tadel, and Avi Yagil. 2024. RenderCore – a new WebGPU-based rendering engine for Root-eve. *EPJ Web of Conferences* 295 (2024), 03035. <https://doi.org/10.1051/epjconf/202429503035>
- Satyadhyana Chickerur, Sankalp Balannavar, Pranali Hongekar, Aditi Prerna, and Soumya Jituri. 2024. WebGL vs. WebGPU: A Performance Analysis for Web 3.0. *Procedia Computer Science* 233 (2024), 919–928. <https://doi.org/10.1016/j.procs.2024.03.281> 5th International Conference on Innovative Data Communication Technologies and Application (ICIDCA 2024).
- Ben Craven, Matthew Buscemi, and Anthony Bowker. 2023. *Web runtime updates are here: Take your browser to the next level*. Retrieved July 18, 2024 from <https://blog.unity.com/engine-platform/web-runtime-updates-enhance-browser-experience>
- William J. Dally, Stephen W. Keckler, and David B. Kirk. 2021. Evolution of the Graphics Processing Unit (GPU). *IEEE Micro* 41, 6 (2021), 42–51. <https://doi.org/10.1109/MM.2021.3113475>
- Robert L. Dotson. 2022. *DynamicalJS: A composable framework for online exploratory visualization of arbitrarily-complex multivariate networks*. Master's thesis. Harvard University Division of Continuing Education.
- Emil Fransson and Jonatan Hermansson. 2023. Performance comparison of WebGPU and WebGL in the Godot game engine. (2023). <https://urn.kb.se/resolve?urn=urn:nbn:se:bth-24706>
- Google. 2023. *Chrome Platform Status: WebGPU*. Retrieved July 18, 2024 from <https://chromestatus.com/feature/6213121689518080>
- Google. 2024. *Chrome Platform Status: WebGPU on Android*. Retrieved July 18, 2024 from <https://chromestatus.com/feature/5119617865613312>
- Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaiola. 1984. Modeling the interaction of light between diffuse surfaces. *ACM SIGGRAPH computer graphics* 18, 3 (1984), 213–222.
- Ned Greene. 1986. Environment mapping and other applications of world projections. *IEEE computer graphics and Applications* 6, 11 (1986), 21–29.
- Larry Gritz and James K. Hahn. 1996. BMRT: A Global Illumination Implementation of the RenderMan Standard. *Journal of Graphics Tools* 1, 3 (1996), 29–47. <https://doi.org/10.1080/10867651.1996.10487462>
- Niklas Harrysson, Doug Smythe, and Jonathan Stone. 2019. *MaterialX Physically-Based Shading Nodes Introduction*. Retrieved July 18, 2024 from <https://materialx.org/assets/MaterialX.v1.37REV2.PBRSpec.pdf>
- Garrett Johnson. 2024. *three-gpu-pathtracer*. Retrieved July 18, 2024 from <https://github.com/gkjohnson/three-gpu-pathtracer>
- James T. Kajiya. 1986. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, 143–150.
- Gerald Kimmersdorfer, Dominik Wolf, and Manuela Waldner. 2023. WebGPU for Scalable Client-Side Aggregate Visualization. *Proceedings of Eurographics - The European Association for Computer Graphics* (2023), 1–3.
- Matias Koskela, Kalle Immonen, Markku Mäkitalo, Alessandro Foi, Timo Viitanen, Pekka Jääskeläinen, Heikki Kultala, and Jarmo Takala. 2019. Blockwise Multi-Order Feature Regression for Real-Time Path-Tracing Reconstruction. *ACM Trans. Graph.* 38, 5, Article 138 (jun 2019), 14 pages. <https://doi.org/10.1145/3269978>
- Emmett Lalish. 2024. *Khronos PBR Neutral Tone Mapper*. Retrieved July 18, 2024 from <https://github.com/KhronosGroup/ToneMapping>
- Daqi Lin, Markus Kettunen, Benedikt Bitterli, Jacopo Pantaleoni, Cem Yuksel, and Chris Wyman. 2022. Generalized resampled importance sampling: Foundations of restir. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–23.
- Erich Loftis. 2024. *THREE.js-PathTracing-Renderer*. Retrieved July 18, 2024 from <https://github.com/erichlof/THREE.js-PathTracing-Renderer>
- David Luebke. 2003. *Level of detail for 3D graphics*. Morgan Kaufmann.
- George Marsaglia. 2003. Xorshift rngs. *Journal of Statistical software* 8 (2003), 1–6.
- Makoto Matsumoto and Takuji Nishimura. 1998. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.* 8, 1 (jan 1998), 3–30. <https://doi.org/10.1145/272991.272995>
- Mozilla. 2023. *Mozilla Platform GFX WebGPU*. Retrieved July 18, 2024 from <https://wiki.mozilla.org/Platform/GFX/WebGPU>
- Thomas Müller, Fabrice Rousselé, Jan Novák, and Alexander Keller. 2021. Real-time neural radiance caching for path tracing. *ACM Trans. Graph.* 40, 4, Article 36 (Jul 2021), 16 pages. <https://doi.org/10.1145/3450626.3459812>
- Melissa E. O'Neill. 2014. PCG: A family of simple fast space-efficient statistically good algorithms for random number generation. *ACM Trans. Math. Software* (2014).
- Y. Ouyang, S. Liu, M. Kettunen, M. Pharr, and J. Pantaleoni. 2021. ReSTIR GI: Path Resampling for Real-Time Path Tracing. *Computer Graphics Forum* 40, 8 (2021), 17–29. <https://doi.org/10.1111/cgf.14378>
- Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2023. *Physically Based Rendering: From Theory to Implementation*. The MIT Press.
- Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. 2009. Approximating dynamic global illumination in image space. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games* (Boston, Massachusetts) (I3D '09). Association for Computing Machinery, New York, NY, USA, 75–82. <https://doi.org/10.1145/1507149.1507161>
- Bastian Sdorra and Tobias Häußler. 2022. *dspbr-pt*. Retrieved July 18, 2024 from <https://github.com/DassaultSystemes-Technology/dspbr-pt>
- Shu Shi and Cheng-Hsin Hsu. 2015. A Survey of Interactive Remote Rendering Systems. *ACM Comput. Surv.* 47, 4, Article 57 (may 2015), 29 pages. <https://doi.org/10.1145/2719921>
- Tomasz Stachowiak. 2015. Advances in real time rendering, part I, Stochastic Screen-Space Reflections. In *ACM SIGGRAPH 2015 Courses* (Los Angeles, California) (SIGGRAPH '15). Association for Computing Machinery, New York, NY, USA, Article 1. <https://doi.org/10.1145/2776880.27877701>
- Three.js. 2024. *Three.js WebGPURenderer.js*. Retrieved July 18, 2024 from <https://github.com/mrdoob/three.js/blob/fbef51075d125234079aaa494f9da1066f3d3e77/src/renderers/webgpu/WebGPURenderer.js>
- Greg Turk and Marc Levoy. 1994. Zippered polygon meshes from range images. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '94)*. Association for Computing Machinery, New York, NY, USA, 311–318. <https://doi.org/10.1145/192161.192241>
- Ziya Usta. 2024. WebGPU: A new Graphic API for 3D WebGIS Applications. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XLVIII-4/W9-2024* (2024), 377–382. <https://doi.org/10.5194/isprs-archives-XLVIII-4-W9-2024-377-2024>
- Martin Valigursky. 2023. *Initial WebGPU support lands in PlayCanvas Engine 1.62*. Retrieved July 18, 2024 from <https://blog.playcanvas.com/initial-webgpu-support-lands-in-playcanvas-engine-1-62/>
- Eric Veach. 1998. *Robust Monte Carlo Methods for Light Transport Simulation*. Stanford University, Department of Computer Science.
- W3C group for GPU web standards. 2024. *WebGPU Conformance Test Suite*. Retrieved July 18, 2024 from <https://github.com/gpuweb/cts>
- Evan Wallace. 2011. *WebGL Path Tracing*. Retrieved July 18, 2024 from <https://experiments.withgoogle.com/webgl-path-tracing>
- Turner Whitted. 2020. Origins of Global Illumination. *IEEE Computer Graphics and Applications* 40, 1 (2020), 20–27. <https://doi.org/10.1109/MCG.2019.2957688>
- Mike Wyrzykowski. 2023. *WebGPU now available for testing in Safari Technology Preview*. Retrieved July 18, 2024 from <https://webkit.org/blog/14879/webgpu-now-available-for-testing-in-safari-technology-preview/>

## A PERFORMANCE MEASUREMENTS

The measurements are split into two parts: CPU performance and compute pipeline GPU performance, as described in Figure 5. CPU performance excludes loading the glTF, which is heavily dependent on bandwidth and focuses on the BVH construction. For the GPU phase, a total of 100 samples per pixel with a ray depth of five was used. The image was rendered in Chrome 126 at a resolution of 512×512 pixels. Experiments were conducted with different model complexities. The simplified versions are decimated meshes of the original, which consists of roughly one million triangles. The LOD artifacts are shown in Figure 6. The first two levels are intended to be visually similar, while the third level is a simplified version intended to demonstrate the effect of non-manifold geometry for ray tracing.

The machine specifications used for the measurements are:

- Apple M1 Max: MacBook Pro with Apple Silicon M1 Max
- AMD/NVIDIA: AMD Ryzen 5 5600X and NVIDIA GeForce RTX 3080 (10 GB)



**Figure 6: The three LOD artifacts, from left to right: 1,068,735 triangles, 106,873 triangles, 10,687 triangles. The left and middle figures share similar visual fidelity characteristics.**

The results for CPU performance were recorded using the Performance API and are presented in Table 1. The results for GPU performance were recorded using WebGPU timestamp queries and are presented in Table 2. All measurements were calculated using the mean time of 30 benchmark samples, with a 95% confidence interval given as  $\pm$  standard deviation from the mean time in milliseconds.

**Table 1: BVH setup time based on model complexity**

Triangles	Apple M1 Max	AMD/NVIDIA
1,068,735	327.57 ms $\pm$ 0.56 ms	383.10 ms $\pm$ 4.06 ms
106,873	45.49 ms $\pm$ 0.31 ms	41.77 ms $\pm$ 0.43 ms
10,687	10.53 ms $\pm$ 0.24 ms	8.43 ms $\pm$ 0.22 ms

**Table 2: GPU path tracer time based on model complexity**

Triangles	Apple M1 Max	AMD/NVIDIA
1,068,735	2,319.45 ms $\pm$ 11.14 ms	1,058.73 ms $\pm$ 59.47 ms
106,873	1,992.11 ms $\pm$ 8.49 ms	790.20 ms $\pm$ 3.79 ms
10,687	2,031.38 ms $\pm$ 9.87 ms	790.83 ms $\pm$ 4.18 ms

The path tracer has not been optimized for performance. These measurements are intended to provide a baseline for future optimizations.