Texture-based Global Illumination for Physics-Based Light Propagation in Interactive Web Applications

Adrian Roth
University of Applied Sciences and Arts
Northwestern Switzerland
Hochschule für Technik
Bahnhofstrasse 6
5210 Windisch
adrian.roth.ar@proton.me

Hilko Cords
University of Applied Sciences and Arts
Northwestern Switzerland
Hochschule für Technik
Bahnhofstrasse 6
5210 Windisch
hilko.cords@fhnw.ch

ABSTRACT

Lamps are difficult to market via websites. Renderings and photographs of showrooms rarely fit consumer preferences and demands and the technical specifications for lighting are unintuitive and difficult to understand for non-experts. This poses a challenge for lighting system manufacturers and suppliers, as customers increasingly make purchases online. Giving customers the option to see how a lamp affects the environment would provide them with an intuitive way to find a suitable product. Unfortunately, expensive global illumination rendering methods are required to accurately simulate the effect of a specific lamp on the environment. However, contemporary methods are not suited for low-end consumer hardware or interactive web-applications. Therefore, we present a new texture-based global illumination method that simulates the effect of specific lamps on complex, polygonal 3D geometries on the GPU. Our method employs iterative light propagation to pre-compute the illumination in texture space, leveraging GPU-based bounding volume hierarchies. Our method can provide pre-calculated physics-based lighting on demand in under a second for interactive scenarios in browsers using WebGl - outperforming state-of-the-art offline renderers significantly.

Keywords

physics-based simulation, real-world lights, pre-calculated global illumination, light propagation, texture-based, real-time web-applications, polygonal objects

1 INTRODUCTION

The characteristics of lights are abstract and difficult for non-experts to grasp intuitively. Thus, manufacturers and retailers of lamps and lighting systems find it challenging to market their products through a website. Images of showrooms, such as seen in Figure 1, are commonly used to showcase lighting systems in the real-world. However, this may not meet consumer preferences and needs and cannot demonstrate the lighting system in the user's actual environment. Furthermore, the visual perception of lighting systems is heavily determined by the concrete room and furniture in which they are installed. Especially if there is a large amount of indirect lighting.

Figure 1: *Example of a lamp showcase from the company Ribag AG [Rib24].*

Previously, we developed a web configuration tool that determines physical light distribution properties based on physical lamp data (IES-data [Ill91]) using global illumination (GI) [Rie+23]. By applying an analytical solution for concave, rectangular, cuboidal rooms to solve the rendering equation, we achieved high performance on GPUs and mobile devices and determined accurate physical light distributions within milliseconds (Figure 2). This lets us provide pre-calculated GI on

demand in under a second, allowing users to re-render scenes for a quick and iterative design process in a web browser.

However, without furniture, the rooms are lifeless and cannot accurately represent a customer's unique environment. Furthermore, the empty rooms struggle to convey their size or scale, making it difficult to understand the effect of lighting systems in real-world settings.

Figure 2: *An empty room with several lamps rendered physically accurate with analytical GI [Rie+23].*

To address this issue, we propose a GI approach for creating physically accurate illumination of IES-Data in furnitured scenes. As a result, we suggest implementing spatial acceleration methods on the GPU while facilitating only the most basic OpenGL features in order to significantly increase light calculation and path tracing performance and allow for an implementation on the web - with WebGL. Thereby, all static lighting information is pre-calculated and stored in textures, enabling interactive rendering performance during exploration. Our method is suitable for determining accurate illumination and interactive exploration on the Web using WebGL (Figure 3). Our technique accurately replicates shadows, indirect lighting, and diffuse surface reflections while executing extensive ray casting operations across the entire geometry for the path tracing algorithm.

Figure 3: *A furnished scene of a polygonal living room rendered physically accurate on the GPU using our proposed rendering method.*

2 RELATED WORK

Real time global illumination is an active topic of research. Recent advances in hardware [Bur20] and rendering techniques [WP22; Lin+22] enable real time path tracing in complex scenarios. While these techniques still rely on sophisticated hardware, they reflect a broader shift in consumer expectations toward realistic rendering.

The number of ways to achieve global illumination is considerable. [DBB18] provides a comprehensive overview of global illumination algorithms. Of these, path tracing [Kaj86; LW93] and radiosity [Gor+84] are the most relevant to our work. Path tracing determines global lighting using stochastic ray casting, whereas radiosity uses an iterative radiance transfer technique to exchange light between a scene's surfaces.

The development of real time global illumination for web applications has additional hurdles as it must cater to a wide range of platforms. E.g., WebGL enables web applications to access graphics hardware from a browser while being platform-independent. However, this platform-agnostic design comes at the cost of lower performance and a limited range of features compared to desktop solutions like OpenGL, Vulkan or DirectX.

Early attempts to perform global illumination with WebGL 1.0 were highly effective, but limited in scale and ability [Con+11; Hac15]. Recent approaches using the upgraded framework, WebGL 2.0, can display complicated scenes with excellent precision and detail. Lesar et al. produced real-time representations of volumetric medical data utilizing path tracing in web environments [LBM18]. They achieve accurate renderings by iteratively and progressively using Monte Carlo ray tracing. However, their approach is limited to volumetric data. In contrast, Nilsson et al. obtained relevant results for mesh-based scenes [NO18]. Using path tracing, they were able to render scenes with thousands of triangles at about 10Hz. By accumulating samples they are able to render high quality images of simple scenes. Most recently Vitsas et al. [Vit+21] have developed a general purpose ray tracing framework on top of WebGL able to produce photorealistic 3D graphics on the web.

Light maps, on the other hand, have long been the norm for global illumination effects in real-time applications [ÖA17; Ras+10]. Because of the static nature of light maps, recent advances in real-time global illumination make them increasingly insignificant. However, so far, they still remain the standard for games, web, and mobile applications due to their high visual quality for static scenes and their low impact on runtime performance. Nevertheless, the above-mentioned, recent advances in real-time global illumination have also benefited the fast construction of light maps [Luk+13; CL21].

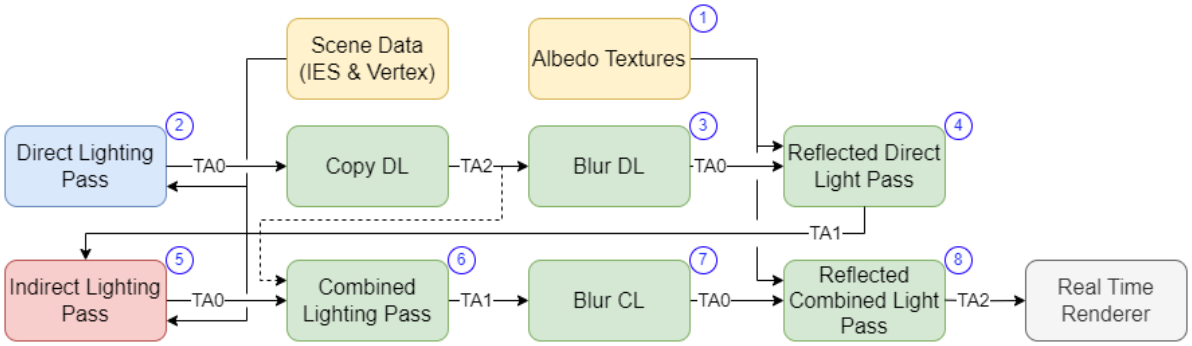

Figure 4: *Our global illumination render pipeline: Direct light shader executions (blue), indirect light shader executions (red) and effects shader executions (green). Yellow indicates static resources. Each step's output indicates the texture array (TAs) that are rendered into. The numbers at the top right of each step correspond to the numbered textures in Figure 5.*

Typically, efficient path-tracing and hence, efficient light map creation, necessitate specific data structures for accelerating ray casting. Bounding volume hierarchies (BVHs) are well established methods that considerably reduce the number of ray-triangle intersection tests [LBM18; STØ05]. However, implementing BVHs on the GPU using WebGL remains challenging. WebGL does not support stacks, hence stack-less algorithms are required. Threaded bounding volume hierarchy (TBVH) uses a fixed traversal order with pre-computed hit and miss pointers [STØ05]. Hachisuka proposed a variation to TBVHs as Multiple-threaded bounding volume Hierarchy (MTBVH), to lessen the negative performance impact from the fixed traversal [Hac15]. Thereby, MTBVH pre-computes several versions of the hit and miss pointers that are optimised based on the general direction of a ray.

3 OUR APPROACH

Our rendering method is related to path tracing and radiosity, and it operates with similar concepts. For each object we generate a series of light maps in a pre-process we call the global illumination pipeline. To enable interactive framerates, the results are given to a real-time rasterisation renderer.

As is often the case, our light maps are generated using an iterative Monte Carlo ray-tracing algorithm. In the first iteration, the direct light from each light source is determined and stored in a light map for each object. In a second iteration, the indirect light is calculated for each surface using a reverse path tracing algorithm. To determine possible sources of reflected light, we cast a number of rays in random directions for each pixel in the light map. Therefore, we transform the uv-coordinate into the world space. The amount of incoming light is then determined using the direct light maps.

Given enough rays, we can expect the results to converge to a reasonable estimation of the reflected light based on the idea of Monte Carlo global illumination by Lafortune [Laf96]. The indirect lighting pass can be repeated for as many iterations as desired, but in practice, the cost of a second reflection outweighs it's contribution to the overall illumination effect. Finally, the computed direct and indirect light maps are accumulated and fed into the real-time renderer.

In the following, we will elaborate on the details of the global illumination pipeline, specifically in terms of the improvements we implemented to render polygonal models and apply the BVH to accelerate the ray casting procedure. We will also briefly present our solution to circumvent the texture limitation of WebGL 2.0.

3.1 Global Illumination Pipeline

The global illumination pipeline assumes a static scene. There are no moving objects or lights that could cause the lighting to change. This fits in very well with our use case in the configuration of professional lighting systems.

Our global illumination pipeline consists of two basic steps: direct and indirect lighting passes. Figure 4 depicts the complete process, which we shall detail below. In Figure 5, the resulting textures of the model of a chair for each render call in the process are shown. Whereas Figure 6 shows the final result.

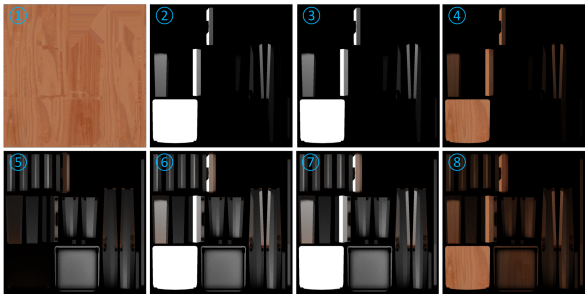

Figure 5: *Evolution of the texture of a chair: (1) Albedo Texture, (2) Direct Light (3) Blurred Direct Light, (4) Reflected Direct Light, (5) Indirect Light, (6) Combined Light, (7) Blurred Combined Light, (8) Reflected Combined Light.*

To compute the direct light (2), we walk over each point on each surface and determine how much light we re-

ceive from each light source. This requires both the scene geometry to check for obstructions and the determination of the incoming light intensities in the IES data format. The results are stored in textures for each object. For our purposes, the texture mapping for the polygon models is expected to be unique, ensuring that no two surfaces share pixels. If they did, the light data from one surface would override another's.

Afterwards, the results of direct light texture are copied for later use. To reduce artefacts from sampling the IES file, we apply a Gaussian blur to the direct light texture (3). The direct light is then multiplied by the object's albedo texture to get the reflected direct light (4). The Gaussian blur must be applied prior to multiplying with the albedo texture. If we would apply the blur after multiplying the albedo texture and direct light, we would lose the albedo texture's details.

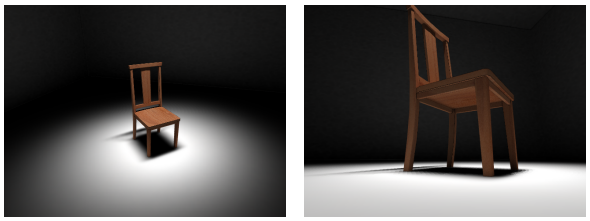

Figure 6: *A chair, rendered with our render method.*

To determine the indirect light (5), we step through each point on each surface and cast numerous rays in random directions. Wherever the rays hit a surface, we extract the associated light intensity from the reflected direct light texture. We then average the received light intensities from all rays to determine how much light is received at the sampling spot. Thus, we gather global light information from the scene onto the sampled point (see Figure 7).

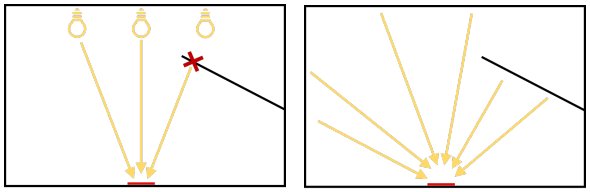

Figure 7: *Diagrams of the direct (left) and indirect (right) lighting passes. The sampled surface is in red. The cast rays are indicated in yellow and point in the direction the light travels, and therefore against the direction the ray is cast.*

Afterwards, the results of the indirect lighting pass and the original texture of the direct light are summed up (6). The end result is the combined light intensity that reaches each surface of the object. We blur it again to remove noise from the direct and indirect passes (7), then multiply it by the albedo texture of the object (8). Finally, we are left with a texture that depicts the total amount of light reflected on the surface of the object.

3.2 BVH

During the direct lighting and indirect lighting passes, we perform millions of ray-triangle intersections. To accelerate this procedure, we employ BVHs. We create the BVH using a simple top-down algorithm and the cost function surface area heuristic (SAH) as described by Jefferey Goldsmith and John Salmon [GS87]. The traversal algorithm on the GPU is based on the MT-BVH algorithm described by Hachisuka et al. [Hac15]. Wherein only the traversal order of the BVH is required, represented as hit and miss links for each node, and the axis-aligned bounding box (AABB). Leaf nodes also store a reference to the set of vertices they contain.

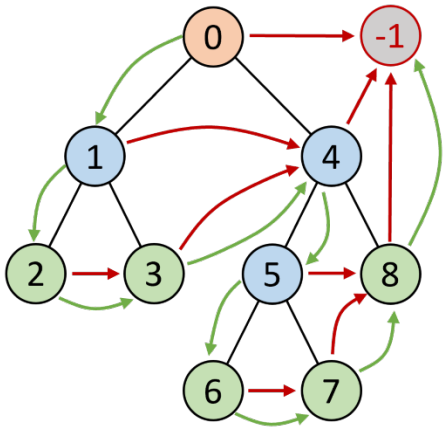

Figure 8: *Example of TBVH: the red node is the root node, the blue nodes are internal nodes and the green nodes are leaf nodes. The nodes are numbered in depth-first order. Green arrows show the hit links and red arrows show the miss links.*

The hit and miss links are shown in Figure 8. The green hit links follow the depth-first numbering of the nodes. The miss links always point to the next unvisited, unrejected node. For leaf nodes, they are always equal to the hit links. The miss link of internal left nodes (e.g. 1 and 5) points to their sibling node. Internal right nodes (e.g. 4) point to the closest unvisited right sibling of a preceding node. If there are no more viable nodes, the hit and miss links point to -1 as an indication that the traversal has ended.

With TBVH the traversal algorithm remains straightforward. It merely needs to track the current minimal distance when stepping through the BVH. At each level, we check whether the bounding volume is hit or missed, and then proceed to the node provided in the hit or miss link respectively. Extending the traversal to MTBVH merely requires more hit and miss link lists for the optimised traversal in each major direction.

Ordering the hit and miss links, AABBs, and vertex reference indices in the same order in respective lists eliminates the need to construct a data structure on the GPU. The node is represented by an index that reads

relevant data from one of four lists (hit links, miss links, AABBs, and vertex indices).

To accommodate the transfer of the data from the CPU to the GPU, the traversal data of the BVH has to be encoded into a texture. The four aforementioned lists are therefore wrapped into a two dimensional array. A node is then no longer represented by an index but a 2D coordinate (see Figure 9). The hit and miss links can therefore also be stored as 2D coordinates pointing to the next node in the traversal order. The AABBs are represented through two three dimensional vectors for the lower and upper bound of the AABB. The vertex indices can remain as they are. This results in a total of 30 floating point values and two integer values per node.

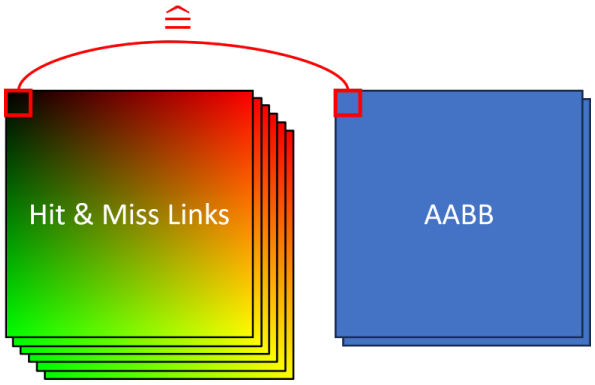

Figure 9: *Encoding of BVH traversal data in two data texture arrays. The data for a node in the BVH is dispersed across all textures but always at the same position in each texture. Each node can be represented through the 2D coordinate of that position.*

We can store all 32 values in 8 four channel floating point pixels. The hit and miss links per direction can be stored in one 4 channel pixel, each link has 2 components. The lower bound of the AABB and the starting index of the vertices can be stored as one pixel as well as the upper bound and the end index (Figure 10). To maintain the simplicity of representing a node as a 2D coordinate, each of these eight pixels are stored in separate textures at the appropriate position.

For our implementation we've opted to store the hit and miss links in a six layer texture array and the AABBs in a two layer texture array (Figure 9). We've separated the two out of convenience and clarity in the code base and for easier debugging. However, there is no technical reason to separate the two and a single texture array could hold all eight textures.

3.3 Texture Limitations

WebGL 2.0 imposes a hard limit for active textures during a single pass. The limit depends on the hardware and currently is usually either 16 or 32 textures. WebGL is designed for forward rendering applications

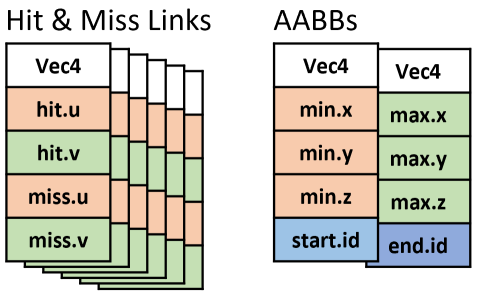

Figure 10: *Encoding of a single BVH node as eight 4 channel pixels (vec4). 6 pixels to hold the hit and miss links for each direction in the MTBVH and 2 pixels to define the AABB and the start and end indices of the associated vertices.*

which rarely exceed this limitation. However, our method needs the direct light maps of every model in the scene during indirect lighting passes. Additionally, the scene geometry and acceleration structures are also stored and passed to the GPU through data textures. As the number of models in a scene grows, it becomes increasingly difficult to accommodate all of its textures with the limit imposed by WebGL.

To circumvent the texture limit, we can use texture arrays. They count as one texture against the limit but are able to hold several textures in a three dimensional space. All textures in a texture array have the same resolution, which might pose problems when reading or writing to them on the GPU. There are two options for creating a texture array for textures with different resolutions. Either the textures are scaled so that they all have the same resolution, or the texture array is dimensioned for the largest texture, with smaller textures occupying only a portion of the layer.

Both options have substantial drawbacks. Scaling textures changes the distribution of pixels per surface. Because we use the texture as our sampling raster, the distribution of sampled points also changes. Alternatively, if the textures don't fill the full resolution, built-in WebGL functions such as texture wrapping and pixel interpolation will fail for edge cases.

In our implementation, we opted for the latter option. Achieving a uniform sample density was more critical than avoiding texture wrapping or pixel interpolation issues, which can be explicitly handled in shaders. Additionally, edge cases are infrequent when the texture mapping is designed to stay within boundaries.

4 DISCUSSION

We implemented our proposed rendering method and different BVHs within a web environment. Thereby, our prototype uses WebGL and all performance and quality metrics were measured in the Microsoft Edge browser. We performed tests on a laptop NVIDIA GTX 1650 with an Intel Core i7-9750H and a desktop NVIDIA GTX 1080 Ti with an AMD Ryzen 9

3950X. In the following, we will examine how introducing GPU-based TBVH and MTBVH improves performance and discuss and compare visual quality with a reference implementation in Unity.

4.1 Performance

To evaluate the performance of our prototype, we measured the time needed to pre-calculate the global illumination and render the light maps. In particular, we ignore the time needed to load all assets and to generate the BVH, as these procedures are not related to the actual rendering method itself.

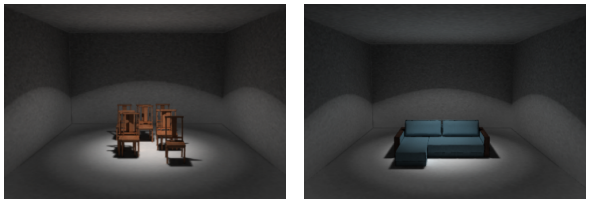

Figure 11: *Test cases chairs_05 (left), with 5 chairs, and sofa_01 (right), with one sofa, rendered with our render method under a single point light.*

We defined a set of scenes and settings to measure the performance improvements of BVHs for different numbers of polygons and rendered pixels (Table 1). The test cases contained a number of chairs (1, 3, 5, or 10) or a single sofa (Figure 11). Each scene was tested on the two aforementioned GPUs with no BVH, TBVH or MTBVH, resulting in 30 test cases. For each test case we recorded ten measurements of the render times.

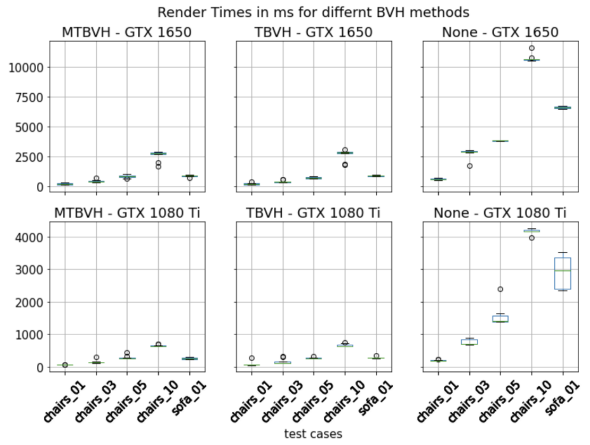

Figure 12: *Render Times in ms for different BVH methods. The number of chair models between each test case is increased, from 1 to 3 to 5 to 10. The sofa_01 test case is the exception, it contains only a sofa (see also Figure 11).*

The box-plot in Figure 12 shows the render times for the different BVH methods across our test cases. BVHs reduce the render times substantially. The increase in render times across test cases cannot be attributed to the increase in polygons alone. Each additional object requires an additional set of light maps to be rendered, increasing the total number of rendered pixels (Table 1). For example, *chairs_5* and *sofa_01* have a similar number of rendered pixels but *sofa_01* consists of twice the number of polygons. Comparing these two test cases provides the best estimation of the effect BVHs have on their own.

Our measurements reveal that the difference between TBVH and MTBVH is minor in our test cases (Table 1). MTBVH frequently leads by a few *ms*, especially for larger scenes, but the difference is negligible. The determination of the overall direction of the ray in our implementation of MTBVH probably causes branching issues on the GPU execution, leading to a longer render time.

4.2 Graphics

In order to evaluate the correctness of our rendering method, we created an equivalent scene in Unity. We then used Unity's built-in light baking pipeline to generate light maps with similar constraints as our own rendering method. Specifically, we allowed only one single reflection per sampled ray. Notably, we used Unity in our case as an offline renderer - the baking of the light maps took Unity's engine between one and three minutes for each of these scenes. The results are shown in 13, 14 and 15.

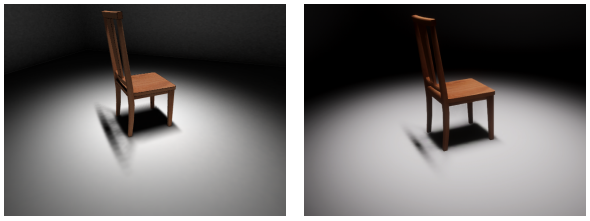

Figure 13: *Visual comparison of shadows between our render method (left) and Unity (right).*

We can confirm the presence of many of the expected global illumination effects. The walls and ceiling are indirectly lit by the area on the floor that is illuminated by the lights. The shadow of the chair is soft due to the influence of multiple lights (Figure 14). Also, there are coloured reflections based on the surface colour of the cube in Figure 15.

Our method produces more detailed soft shadows in Figure 13 than Unity's light maps, which have lesser resolution in principle. Thus, our renderer can represent the shadow gradient at a more granular level, resulting in improved quality.

The employment of a Gaussian filter causes colour bleeding on the edges. In Figure 14, the edges of the chair have small black seams. During both, direct and indirect render passes, the gaps between the texture's faces remain blank. During the blur pass, these gaps bleed into the visible area of the texture. However, these artefacts can easily be prevented by using a more

			Average Render Times per Test Case in ms					
			GTX 1650			GTX 1080 TI		
Test Case	# Pixels	# Polygons	MTBVH	TBVH	No BVH	MTBVH	TBVH	No BVH
chairs_01	459k	252	226	**222**	618	**61**	81	206
chairs_03	590k	732	446	**412**	2810	**154**	168	760
chairs_05	721k	1212	835	**732**	3833	289	**272**	1553
chairs_10	1049k	2412	**2616**	2675	10708	**644**	662	4172
sofa_01	655k	2036	**887**	908	6617	**263**	279	2909

Table 1: *Performance and metrics for each of our test cases in Figure 12. The fastest render times for each test case are indicated in bold.*

Figure 14: *Visual comparison of indirect lighting between our render method (left) and Unity (right).*

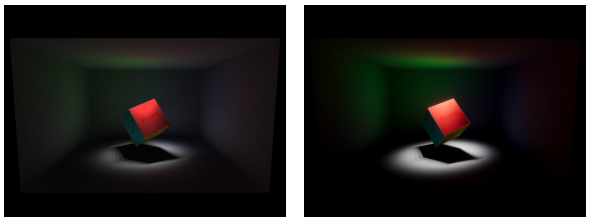

Figure 15: *Visual comparison of coloured reflections between our render method (left) and Unity (right).*

Figure 14 demonstrates the effects of the indirect pass on the underside of the chairs. The chair rendered with our method is more matted and less colourful, but the overall shading is fairly similar to the chair generated with Unity. In Figure 15, we rendered a separate scene containing a test cube only. This image depicts the impact of diffuse coloured surface reflections. Again, the version rendered with Unity is more colourful and less noisy, but the effect is similar in both versions.

One flaw that is visible in Figure 16 is that the floor is completely dark outside the directly lit area. As the walls and the ceiling are not directly lit, there is no lit surface from the direct light pass that could reflect onto the floor. By allowing up to two bounces, the floor is also illuminated. However, with our rendering method, this results in nearly twice the render time for the same scene.

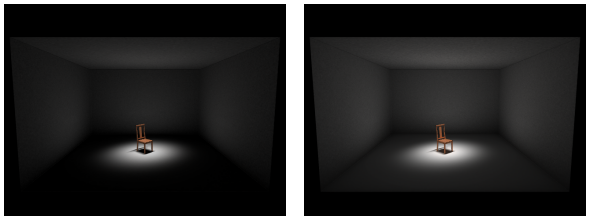

Figure 16: *Comparison of our results rendered with a single indirect light pass (left) and two indirect light passes (right).*

Overall, we have shown that our rendering method delivers similar results to a modern state-of-the-art renderer, while our performance is significantly higher. The baking of the light map took Unity's engine between one and three minutes for each of these scenes. Nevertheless, our method still suffers some minor visual artefacts inherent to our approach, which are the price for the higher performance.

5 CONCLUSION & FUTURE WORK

We presented a rendering approach for fast global illumination on the GPU on the web using WebGL. Our approach enables the simulation of physically accurate light propagation of real lamp data (i.e., IES-data). We achieved high performance by implementing BVHs on the GPU to quickly find ray-triangle intersections. Thereby, static light information is stored in light maps. In this way, a scene can be adjusted and modified before being re-rendered within a second to enable easy design and interactive navigation in web environments, such as configuration tools in the light manufacturing industry.

In this way, we achieve visual results comparable to state-of-the-art offline-renderers with only minor visual shortcomings. However, our approach is significantly faster (seconds vs. minutes).

Future work includes the investigation of more advanced denoising methods such as Non-Local Means with Joint Filtering (JNLM), RadeonPro, Open Image

Denoiser (OIDN) or OptiX [Uni23; God23] to improve visual results. This would probably also allow a reduction in the resolution of light maps, which would improve performance even further. Adapting the rendering method to progressive rendering of light maps, similarly to the works by Lesar et al. and Nilsson et al. [LBM18; NO18], would also allow for intermediate results while further details are added successively as the user navigates the scene. Finally, we plan to incorporate technologies such as Deep-Learning Super Sampling (DLSS) for improved anti-aliasing and importance sampling to increase quality and performance even further.

6 ACKNOWLEDGEMENTS

This work was supported by the project LiSi, funded by the Forschungsfonds Aargau and the company Ribag AG. We would like to thank Dominik Hausherr and Andreas Richner from Ribag AG for the valuable collaboration.

REFERENCES

[Bur20] Andrew Burnes. *Whitepaper: NVIDIA Ampere GA102 GPU Architecture*. Tech. rep. NVIDIA, 2020.

[CL21] Ma Cheng and Wang Lu. "A Fast Light Baking System for Mobile VR Game Based on Edge Computing Framework". In: *Proceedings of the 2021 ACM International Conference on Intelligent Computing and its Emerging Applications*. 2021, pp. 176–181.

[Con+11] John Congote, Alvaro Segura, Luis Kabongo, Aitor Moreno, Jorge Posada, and Oscar Ruiz. "Interactive visualization of volumetric data with WebGL in real-time". In: *Proceedings of the 16th International Conference on 3D Web Technology*. Web3D '11. Paris, France: Association for Computing Machinery, 2011, pp. 137–146. ISBN: 9781450307741. DOI: 10.1145/2010425.2010449.

[DBB18] Philip Dutre, Philippe Bekaert, and Kavita Bala. *Advanced global illumination*. CRC Press, 2018.

[God23] Godot. *Using Lightmap global illumination*. Nov. 30, 2023. URL: https://docs.godotengine.org/en/stable/tutorials/3d/global_illumination/using_lightmap_gi.html (visited on 02/08/2024).

[Gor+84] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. "Modeling the Interaction of Light between Diffuse Surfaces". In: *SIGGRAPH Comput. Graph.* 18.3 (Jan. 1984), pp. 213–222. ISSN: 0097-8930. DOI: 10.1145/964965.808601.

[GS87] Jeffrey Goldsmith and John Salmon. "Automatic Creation of Object Hierarchies for Ray Tracing". In: *IEEE Computer Graphics and Applications* 7.5 (1987), pp. 14–20. DOI: 10.1109/MCG.1987.276983.

[Hac15] Toshiya Hachisuka. *Implementing a Photorealistic Rendering System using GLSL*. 2015. arXiv: 1505.06022 [cs.GR].

[Ill91] Illuminating Engineering Society. *IES Standard File Format for Electronic Transfer of Photometric Data and Related Information*. Tech. rep. IES Computer Committee and others, 1991.

[Kaj86] James T. Kajiya. "The rendering equation". eng. In: *Computer graphics (New York, N.Y.)* 20.4 (1986), pp. 143–150. ISSN: 0097-8930.

[Laf96] Eric Lafortune. "Mathematical models and Monte Carlo algorithms for physically based rendering". In: *Department of Computer Science, Faculty of Engineering, Katholieke Universiteit Leuven* 20.74-79 (1996), p. 4.

[LBM18] Žiga Lesar, Ciril Bohak, and Matija Marolt. "Real-time interactive platform-agnostic volumetric path tracing in webGL 2.0". In: *Proceedings of the 23rd International ACM Conference on 3D Web Technology*. Web3D '18. Poznań, Poland: Association for Computing Machinery, 2018. ISBN: 9781450358002. DOI: 10.1145/3208806.3208814.

[Lin+22] Daqi Lin, Markus Kettunen, Benedikt Bitterli, Jacopo Pantaleoni, Cem Yuksel, and Chris Wyman. "Generalized resampled importance sampling: foundations of ReSTIR". In: *ACM Trans. Graph.* 41.4 (July 2022). ISSN: 0730-0301. DOI: 10.1145/3528223.3530158.

[Luk+13] Christian Luksch, Robert F Tobler, Ralf Habel, Michael Schwärzler, and Michael Wimmer. "Fast light-map computation with virtual polygon lights". In: *Proceedings of the ACM SIGGRAPH symposium on interactive 3D graphics and games*. 2013, pp. 87–94.

[LW93] Eric P Lafortune and Yves D Willems. "Bi-directional path tracing". In: (1993).

[NO18] Martin Nilsson and Alma Ottedag. "Real-time path tracing of small scenes using WebGL". In: (2018).

[ÖA17] Bekir Öztürk and Ahmet Oğuz Akyüz. "Semi-dynamic light maps". In: *ACM SIGGRAPH 2017 Posters*. 2017, pp. 1–2.

[Ras+10] Jim Rasmusson, Jacob Ström, Per Wennersten, Michael Doggett, and Tomas Akenine-Möller. "Texture compression of light maps using smooth profile functions". In: *Proceedings of the Conference on High Performance Graphics*. 2010, pp. 143–152.

[Rib24] Ribag AG. *Collections*. Feb. 5, 2024. URL: https://www.ribag.ch/en/products/collections (visited on 02/05/2024).

[Rie+23] Manuel Riedi, Alexander Legath, Luca Fluri, and Hilko Cords. *Configuration and Simulation Tool for Lighting Systems*. Online-Tool. Aug. 2023. URL: https://iit.cs.technik.fhnw.ch/lichtsystem-konfigurator/ (visited on 02/29/2024).

[STØ05] Lars Ole Simonsen, Niels Thrane, and P Ørbæk. "A comparison of acceleration structures for GPU assisted ray tracing". In: *Master's thesis, University of Aarhus* (2005).

[Uni23] Unity. *The Progressive Lightmapper - Unity Documentation*. 2023. URL: https://docs.unity3d.com/Manual/progressive-lightmapper.html (visited on 02/08/2024).

[Vit+21] Nick Vitsas, Anastasios Gkaravelis, Andreas A Vasilakis, and Georgios Papaioannou. "WebRays: Ray tracing on the web". In: *Ray Tracing Gems II: Next Generation Real-Time Rendering with DXR, Vulkan, and OptiX* (2021), pp. 281–299.

[WP22] Chris Wyman and Alexey Panteleev. "Rearchitecting spatiotemporal resampling for production". In: *Proceedings of the Conference on High-Performance Graphics*. HPG '21. Goslar, DEU: Eurographics Association, 2022, pp. 23–41. DOI: 10.2312/hpg.20211281.