Improved Rendering for Connectome Visualization Using Real-Time Raytracing

Sebastian Kopacz
Computer Science
University of Calgary
Calgary, Canada
sebastian.kopacz@ucalgary.ca

***Abstract*—Current connectome visualizations use simple or unlit shading techniques with a set of colors depicting different neuron types. Unfortunately, the lack of lighting information causes the resulting renders to appear flat and homogeneous. This approach may be problematic, as it not only degrades visual appeal but also makes the visualization difficult to analyze. As a solution, I wrote a renderer that used lighting techniques such as shadows or ambient occlusion to improve the visual clarity of connectome map visualizations. Using ambient occlusion alone resulted in better visual quality than combining it with shadows.**

Keywords—Neuroimaging, Connectome, Visualization, Ambient Occlusion, Shading, Raytracing, Rasterization, Deferred Rendering

I. Introduction

Connectome maps aim to comprehensively map all the neural connections within a given organism's nervous system. These maps allow researchers not only to better understand the nervous system of the organism that they are studying but also to simulate it, as has been done with *Caenorhabditis elegans* [1]. Additionally, connectome maps allow for visualizations which can depict individual neurons. This makes for stunning visuals during publications as well as being able to aid in understanding and teaching neuroscience concepts.

Unfortunately, current visualization techniques tend to use basic or completely unlit shading. Due to our visual system's reliance on lighting information, the lack of proper shading techniques decreases both the visual appeal and informative value of those visualizations. For instance, without proper lighting information, depth and edge detection become harder, which essentially makes the image appear flat and homogeneous [2]. This makes it more difficult to visually trace individual neurons as can be seen in Fig. 1. Using distinct colors for different neurons can help alleviate this issue but there will still be self-overlapping problems due to the branching nature of neurons. In addition, current visualizations tend to only use distinct colors for neuron types instead of unique colors for each neuron in a scene. This will inevitably result in visualizations with multiple unique neurons depicted in the same color, further decreasing visual clarity. Thus, simply including lighting information, by using various shading techniques such as shadows and ambient occlusion (AO) should considerably improve the clarity and visual appeal of connectome visualizations. Raytracing is preferred over rasterization as global lighting requires scene data. Since raytracing is more expensive than rasterization, optimizations and sufficiently powerful hardware are necessary for an interactive renderer.

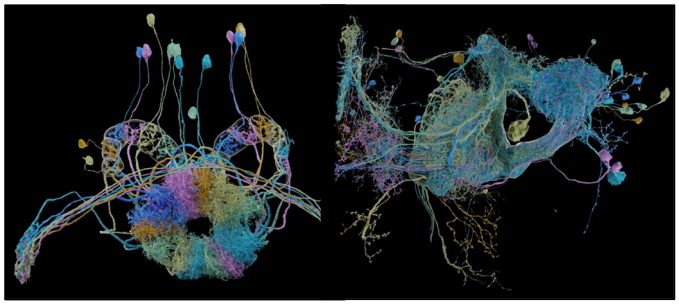

Fig. 1. Example of existing visualization techniques for connectome data as used by the hemibrain connectome project [3].

Therefore, my plan for this project was to implement and evaluate a real-time raytracer render engine with various global lighting effects for connectome visualization. I primarily wanted to explore shadows and ambient occlusion since those seemed to be the most impactful lighting effects for increasing the visual appeal and readability of a connectome visualization. Shadows help estimate depth by grounding objects within a scene. Meanwhile ambient occlusion reveals depth detail in objects by darkening cavities that are likely occluded from most light sources in the scene. As mentioned previously, raytracing is preferred over rasterization because scene information is directly accessible, which allows for better lighting effects. With rasterization, it is impossible to check for other geometry when shading a given fragment (triangle). This is a fundamental limitation of rasterization necessitating the development of various hacks to approximate effects which can be trivially done using raytracing [4]. An example of visual discrepancy between a trivial raytraced solution and an unnecessarily complicated rasterized workaround can be clearly seen in Fig. 2 with regards to ambient occlusion.

The main downside of raytracing is its performance penalty but thanks to a combination of algorithms, optimizations, and custom hardware, nontrivial real-time raytracing is possible on consumer grade hardware. Nvidia created the most accessible and well packaged solution for real-time raytracing when they made their RTX series graphics cards [5]. Their solution neatly packaged a hardware accelerated ray-triangle intersection solver with well optimized software for generating the bounding volume hierarchy (BVH) for a given scene. Therefore, allowing for the computation of around 10 billion rays per second in ideal conditions [6]. For a large connectome visualization, I expect that value to be closer to 2 billion rays per second.

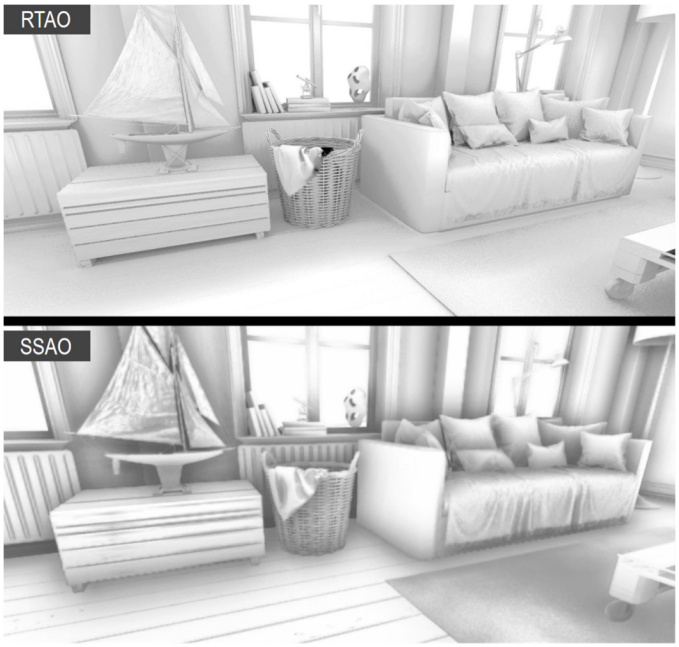

Fig. 2. Comparison between raytraced ambient occlusion (RTAO) and rasterized ambient occlusion (SSAO) [7]. As evident the raytraced version is considerably shaper and has fewer inaccuracies than the rasterized equivalent. Take specific note of the couch and basket.

II. Methodology

A. Materials

I developed the entire project using CLion on Windows 10. As such, the entire program was written in C++ and used the CMake build system. Libraries used in this project were assimp, glfw, mingw-std-threads, spdlog, glm, simpleini, stb, and vulkan-headers. The version of Vulkan used for this project was version 1.1.101.0. Additionally, a Python script was written to generate an image with light ray directions. Version control was done using Git with the repository self-hosted on my server.

In terms of hardware the project was mainly developed on my desktop using an RTX 2080 GPU. This was only necessary for using hardware accelerated raytracing and I was still able to use my laptop with a Nvidia 960m GPU when working with just the rasterizer.

B. Connectome Dataset

The hemibrain connectome by the research group at the Janelia Research Campus is currently the largest synaptic-level connectome [3], [8]. Their goal is to create a map with synaptic resolution of the adult fruit fly (*Drosophila melanogaster),* a species often used in neuroscience research [3], [8]–[10]. They also made their entire dataset publicly available. My first objective was to write or find a program to open or convert swc files into a more common model format. I began by researching how neuron skeletons were stored in a swc file format [11], [12]. This allowed me to write a simple swc loader which simply place objects at the vertices specified in the swc file. However, in the interest of time, I switched to using SWC Mesher, a Blender addon for converting swc files into obj files [13]. Therefore, refocusing the project on building a render engine rather than a neuron modeler.

C. Renderer

Last winter, I wrote a render engine which used Nvidia's hardware-accelerated raytracing technology [14]. That render engine was reused as a starting point for this project. However, some refactoring was necessary as my focus last year was on learning Vulkan and RTX which lead to a suboptimal implementation. While refactoring the renderer, I also added or fixed various features that I neglected last year. These upgrades improved the overall quality of the render engine which made it considerably easier to work with. Once those upgrades were completed, I focused on features that were necessary for the current project, such as implementing lighting effects and loading neurons.

Optionally, I hoped to combine all the lighting effects with deferred rendering as illustrated in Fig. 3. This would result in a hybrid render engine that would use rasterization for local lighting and raytracing for global lighting. The hybrid render engine would lead to better visuals at a lower performance cost by combining the best of both rendering techniques. Sadly, instead of a fancy deferred rendering pipeline, I essentially wrote a single monolithic shader that computer raytraced diffuse lighting, shadows, and ambient occlusion in a single pass. Related to that, enabling and disabling features required modifying the shader file and recompiling it. Although this was undesirable from a long-term maintainability perspective, it was acceptable for the purposes of the project.

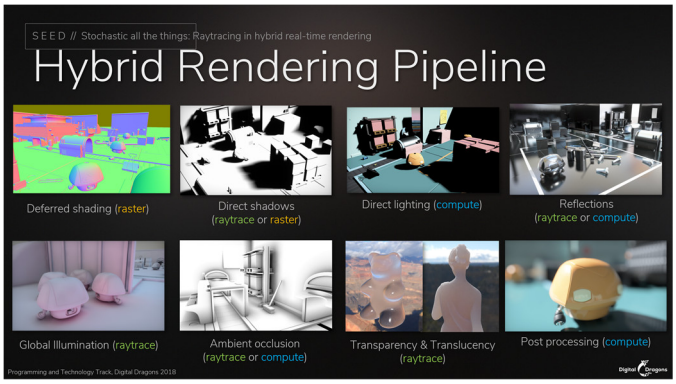

Fig. 3. Hybrid rendering pipeline in *Project PICA PICA* as presented by Tomasz Stachowiak from Electronic Arts at Digital Dragons 2018 [15].

D. Lighting

Implementing raytraced ambient occlusion was my primary focus. This was because the render engine I was reusing already supported raytraced shadows and ambient occlusion was expected to add depth detail to the scene. If I completed that with time to spare, my plan was to add as many lighting effects as possible. With the goal being a more readable and appealing visualization.

Ambient occlusion works by sending rays in every direction around the point that is being shaded and checking if they hit anything. The length of those rays determines how far away intersections with other geometry are counted as occlusions. Since checking every possible direction is impractical, only a handful of rays are cast. Those rays are uniformly distributed in every possible direction around the point and are cast in a random order as can be seen in Fig. 4.

For this, I wrote a Python script which generated an image texture that contained the directions light should travel in. Unfortunately, this process compressed the directions to only those that can be represented in an 8-bit rgb image. This is likely why the texture based on values from the Halton sequence was repetitive as can be seen in Fig. 5. Conversion from x, y, z floats that were between -1 and 1 to r, g, b integers that were between 0 and 255 was done using $(v + 1) * 128$. The reason for using an image as opposed to a single value was to vary the light direction for each ray. Otherwise, I would have a single light source for the entire scene resulting in direct shadows.

Calculating directions was based on the *Global Illumination and Path Tracing* lesson from Scrachapixel 2.0 [16]. I first generated a sequence of random values uniformly distributed in the range [0,1]. This was done using Python's random function since the Halton sequence resulted in a repeating image as can be seen in Fig. 5. Repetition was undesirable because the image was accessed linearly based on screen coordinates and sample count. Therefore, propagating obvious texture repetitions to the final render. The following equations were used to compute the directions light rays should be cast: $x = \sqrt{1 - r_1 * r_2} * \cos(2\pi r_1)$, $y = \sqrt{1 - r_1 * r_2} * \sin(2\pi r_1)$ and $z = r_1$ where r_1 and r_2 are consecutive values from the sequence of uniformly distributed values ranging from 0 to 1.

The directions from the image had to be transformed to the surface normal of the point that they were used for. To do this, I created a transformation matrix consisting of the surface normal and two other orthogonal vectors. The first of those vectors was computed using $(0, -N_z, N_y)$ where N is the normal vector. The second vector was computed using a cross product of the normal vector with the first vector. Finally, the transformation matrix was created from the first, second, and normal vectors. The direction could then be easily transformed to the surface normal of the spot it was needed for by multiplying it with the transformation matrix for that point.

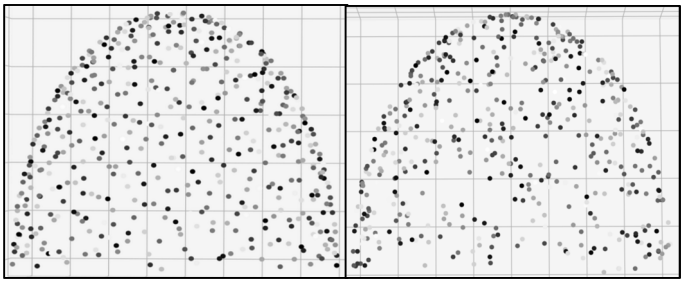

Fig. 4. Hemisphere sampling for calculating global illumination effects like ambient occlusion. The left image is based on values from the Halton sequence, while the right image is based on completely random values. As evident, using values from the Halton sequence results in more uniformly distributed points on the hemisphere. Color intensity of each point relates to the order in which it was drawn, with darker points being drawn before lighter points. This is important as it illustrates that the points are placed in a random order on the hemisphere.

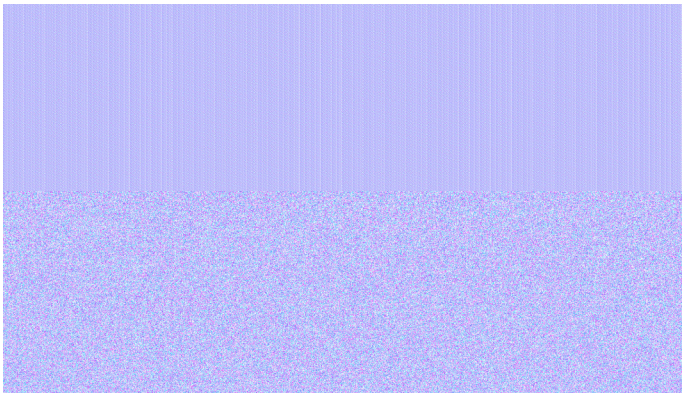

Fig. 5. Samples of the textures storing hemisphere directions for calculating ambient occlusion. The top texture is based on values from the Halton sequence while the bottom texture is based on completely random values. As evident, the top image has repetition while the bottom image is sufficiently random.

E. Performance Evaluation

My hope was to achieve real-time rendering, which I defined as around 60 frames per second on a consumer grade RTX GPU. I also planned on assessing the performance cost of each lighting effect. This feature would have helped determine which features were worth using as well as evaluate the interactivity of the render engine.

III. Results

Unfortunately, when I planned this project, I was expecting considerably fewer bugs and complications than I ended up encountering. This miscalculation meant that by the time I updated my renderer and was able to load neurons from the hemibrain connectome, I had only enough time to implement ambient occlusion. Despite running out of time to implement additional lighting effects, I was still able to complete the main shading features I had set out to implement. Fig. 6. shows all these effects in action on a satellite test model I had created [17]. This model had excellent topology with self-shadowing and detailed crevices making it ideal for showcasing shadows and ambient occlusion. Sadly, the generated neurons have noticeably worse topology with fewer crevices. As a result, neurons had more lighting artifacts than the satellite model.

Additionally, decreasing the ray distance when calculating ambient occlusion further contrasts crevices with flat or round regions of the model. This effect is visualized in Fig. 7. where I decreased the AO ray distance from 1 to 0.01. Unfortunately, this effect would not be applicable for neurons as they do not have sharp crevices like the satellite model. Decreasing the ray distance is a trade off since now there is no large-scale ambient occlusion. For instance, as seen in Fig. 6. the inside of the dish is completely white whereas it had been darker previously.

Applying ambient occlusion to a single neuron darkens the inside parts of branch attachments as seen in Fig. 8. This is consistent with intuition since as seen previously, ambient occlusion darkens crevices. Therefore, branch attachments will be darkened since they create crevices. Additionally, using Cycles (render engine in Blender) produced a similar effect as can be seen in Fig. 9. Therefore, supporting the correctness of my ambient occlusion implementation.

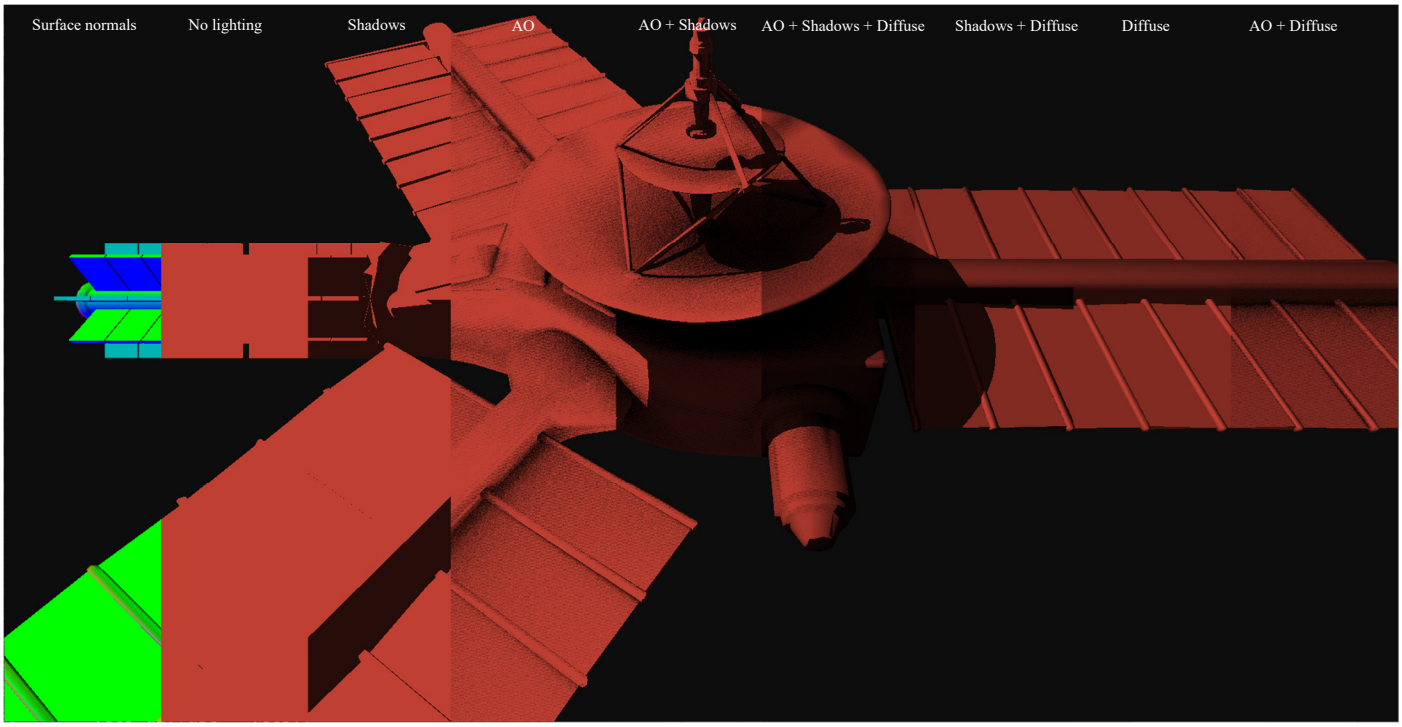

Fig. 6. Satellite model I made which has geometry ideal for showcasing the different shading modes that had been implemented.

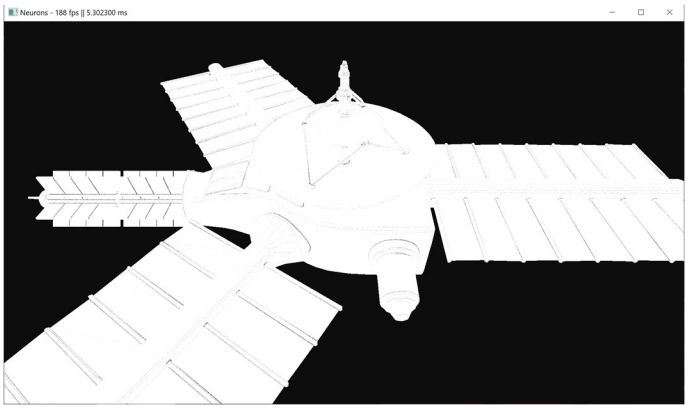

Fig. 7. Satellite model showing ambient occlusion with a short ray distance.

Fig. 8. Effect of ambient occlusion shading on a single neuron as seen in my render engine.

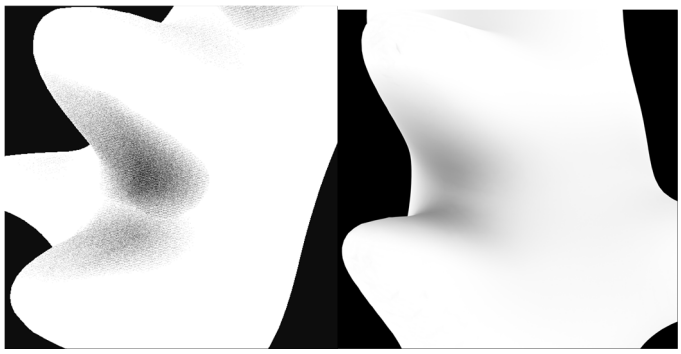

Fig. 9. Comparison between a render from my render engine (left image) and a render from Blender's Cycles render engine (right image). Relatively similar parameters were used in for both renders suggesting that my render is probably correct. Main difference is due to using fewer samples per pixel in my renderer.

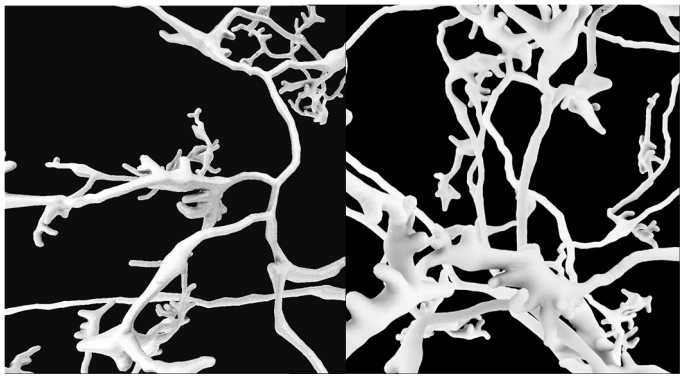

Fig. 10. Comparison between a render from my render engine (left image) and a render from Blender's Cycles render engine (right image). Relatively similar parameters were used in for both renders suggesting that my render is probably correct. Main difference is due to using fewer samples per pixel in my renderer.

As seen in Fig. 11. increasing the ambient occlusion's ray distance improves the visual quality of a neuron with extensive dendritic branching. Since rendering the neuron using Cycles resulted in similar lighting to my own render as seen in Fig. 10. it is reasonable to assume render engine's ambient occlusion implementation is correct.

Unfortunately, as seen in Fig. 12. neurons rendered with shadows resulted in degraded image quality. This was likely due to those shadows having no clear visual connection to the geometry that created them. Perhaps soft shadows could have a positive effect on visual quality.

Combining ambient occlusion with diffuse shading resulted in the best visual quality as demonstrated in Fig. 13. However, the light emanating from the left of the image likely contributes to its visual appeal. Moving the camera so that the light source is from behind the neuron or head on from the camera as can be seen in Fig. 14. degraded visual quality. A potential solution could be to lock the light source with the camera's movement or allow the user to move it around the scene. Ideally the light source would be an environment map that the user would be able to rotate any way they wished.

Fig. 11. Ambient occlusion as applied to a neuron with extensive dendritic branching. The left side of the image used an AO ray distance of 100 while the right side used an AO ray distance of 1. This meant that a larger portion of the scene counted toward a given point's ambient occlusion thereby darkening more of the model.

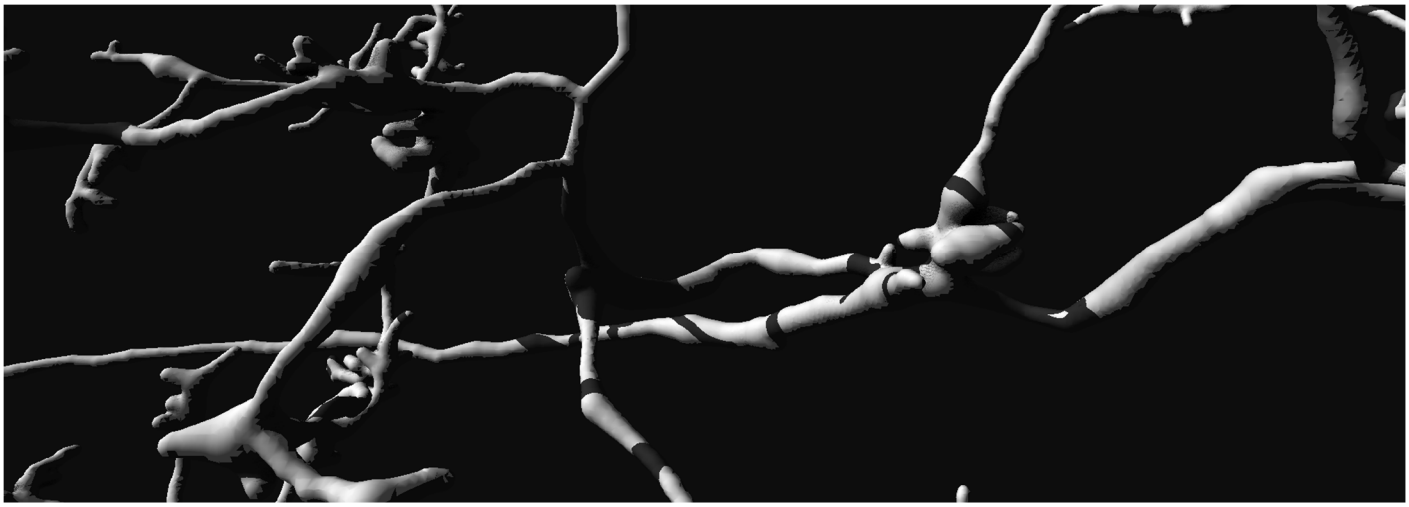

Fig. 12. Neuron render with ambient occlusion, shadows and diffuse lighting enabled. Demostrates the drawback of using shadows in a neuron visualization.

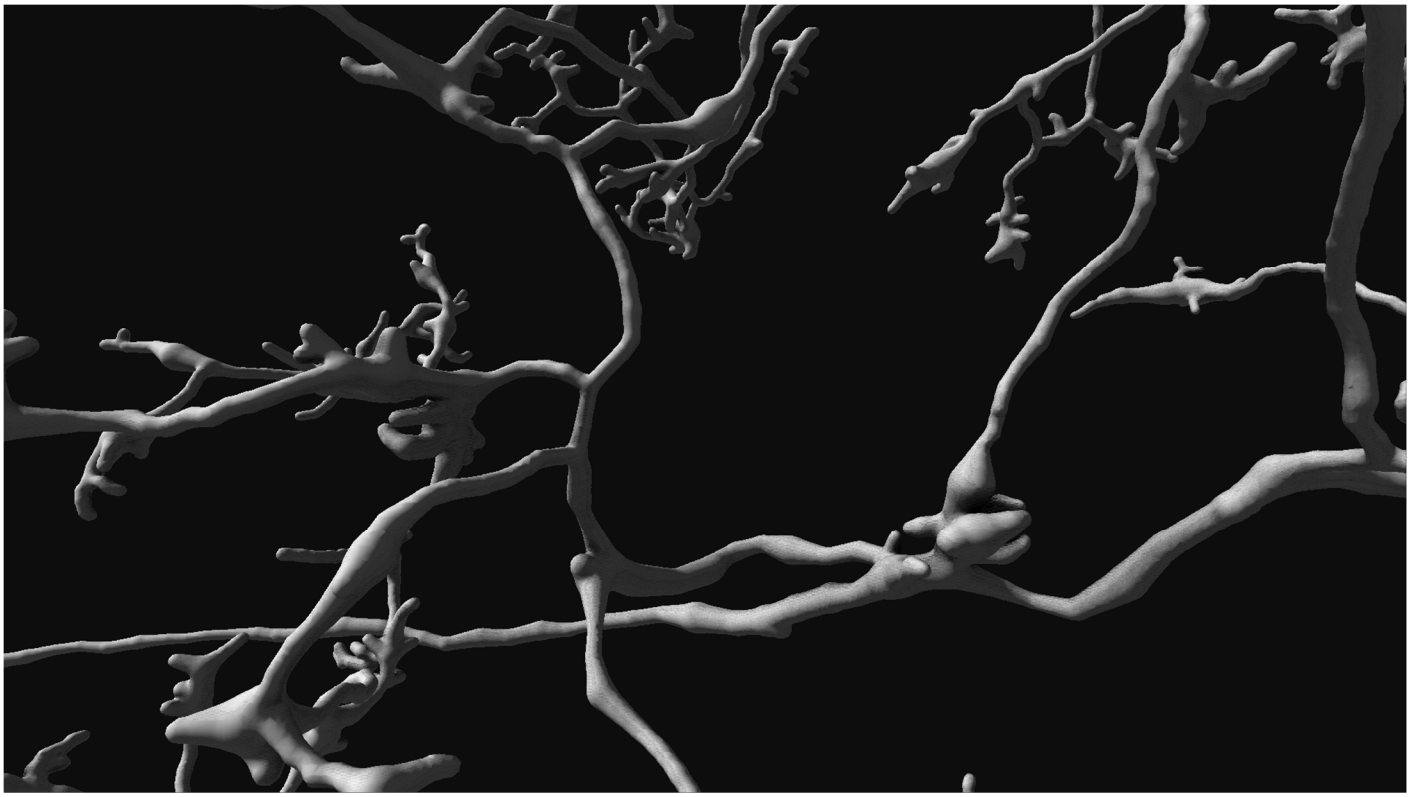

Fig. 13. Ambient occlusion and diffuse shading applied to a neuron with extensive dendritic branching.

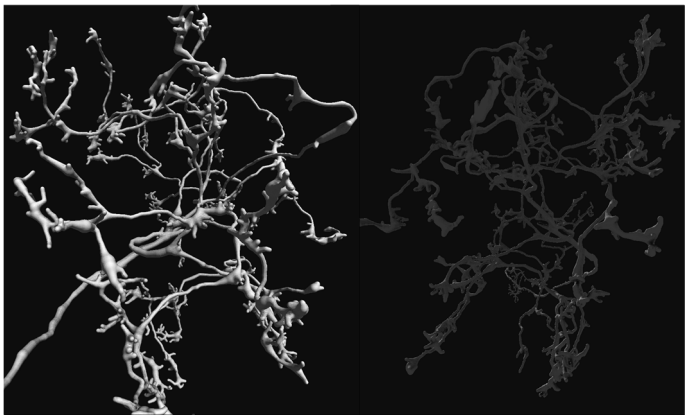

Fig. 14. Neuron renders with diffuse lighting using either a light point from behind the camera (left image) or a light behind the neuron (right image). While the left image looks somewhat okay the right image is considerably worse.

Performance was evaluated manually by looking at the program's title bar (show in Fig. 7.) while using various lighting effects. That data was graphed in Fig. 15. and Fig. 16. Analyzing those graphs revealed that raytraced ambient occlusion (RTAO), was the most expensive lighting feature. The cost of ambient occlusion scaled linearly with how many samples were being calculated. Specifically, $y = x * 0.2$ where x is the number of samples and y is the expected frame time given in milliseconds. Performance of raytraced surface normals and diffuse lighting was equivalent with both running at about 0.7 ms. As expected, rasterization was considerably faster than raytracing with nearly double the performance running at approximately 0.4 ms. raytraced shadows were also fast running at around 0.8 ms.

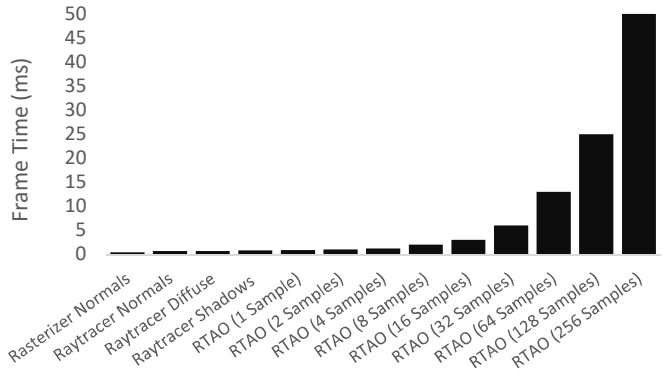

Fig. 15. Performance cost of each effect measured in milliseconds per frame.

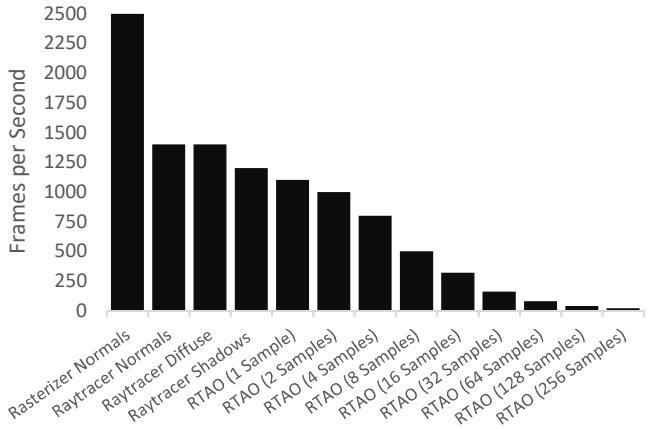

Fig. 16. Performance cost of each effect measured in frames per second.

IV. Conclusion

Overall, ambient occlusion with diffuse lighting was the most visually appealing. However, it is necessary to note that the diffuse lighting had to be from an angle for it to improve visual quality. Otherwise, it would detract from the render instead of adding to it. Unfortunately, shadows were disappointing since they were disconnected from the geometry casting them. Perhaps using soft shadows would be more visually appealing. The idea being that they would be affected by the distance between the geometry creating them and where they would end up falling in the scene. This property would encode a human readable distance from any shadow to the geometry that cast that shadow. As such, future work may wish to explore using soft shadows for encoding distance to occluding geometry.

Another effect that should be explored in future work is a Fresnel Effect. As a test I created this effect in Blender (shown in Fig. 15.) and it highlights neuron edges marvelously. This effect depends on geometry angling away from the camera. As such it works well for curved surfaces such as the side of a neuron. Unfortunately, curved surfaces can be anywhere in a model which may cause unwanted highlights as seen in Fig. 15. Therefore, a better non-photorealistic edge rendering effect might be necessary. Other lighting effects such as: global illumination, reflections, refractions, and subsurface scattering effects could also be explored for both photorealistic and non-photorealistic lighting in future connectome visualizations.

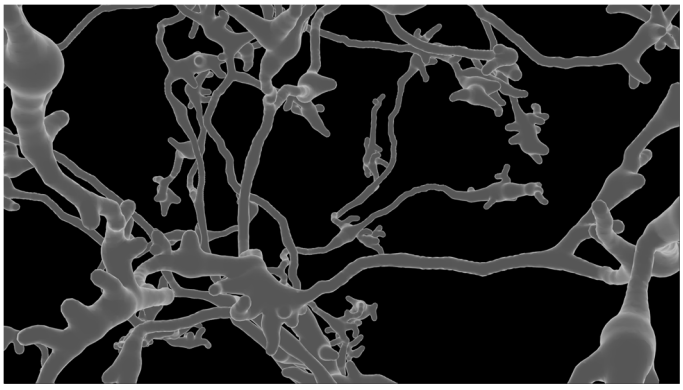

Fig. 17. Neuron rendered with Cycles using Fresnel shading node [18].

Performance was more than adequate with as many as 64 ambient occlusion samples running at 80 frames per second. However, performance degraded near a surface and it is likely that increasing geometry will require decreasing ambient occlusion samples. This should not be an issue since even 16 samples were enough for an acceptable ambient occlusion effect. A denoiser could further decrease the number of samples necessary for a nice ambient occlusion effect.

While the lighting techniques used should trivially transfer to visualizations containing multiple interconnected neurons, only singular neurons were visualized so far. This was largely due to the neuron preprocessing pipeline being mostly manual. As such preparing just a single neuron for visualization required considerable time and effort. Eliminating this limitation by replacing or automating the preprocessing pipeline would be necessary for efficiently preparing multiple neurons for visualization.

Acknowledgments

Dr. Usman Alim for providing suggestions and guidance while supervising the project.

The research group from the Janelia Research Campus that was behind the hemibrain connectome project [3], [8]. This project is enabled by their publicly available dataset.

My render engine is based on an early version of the Hazel Engine that Yan Chernikov is developing as part of a tutorial series [19]. My event system and logger were both taken from the Hazel Engine.

A considerably amount of code in my render engine came from various Vulkan tutorials. Those included Vulkan Tutorial by Alexander Overvoorde, Vulkan C++ examples and demos by Sascha Willems, NVIDIA Vulkan Ray Tracing Tutorial by Martin-Karl Lefrançois and Pascal Gautron [20]–[22].

Even though the RTAO project by Jakub Boksansky used DirectX Raytracing (DXR) instead of Vulkan it was still useful for understanding how ambient occlusion worked [23].

References

[1] G. P. Sarma *et al.*, "OpenWorm: overview and recent advances in integrative biological simulation of Caenorhabditis elegans," *Philos. Trans. R. Soc. B Biol. Sci.*, vol. 373, no. 1758, p. 20170382, Oct. 2018.

[2] P. Mamassian, D. C. Knill, and D. Kersten, "The perception of cast shadows," *Trends Cogn. Sci.*, vol. 2, no. 8, pp. 288–295, Aug. 1998.

[3] Janelia Research Campus, "Hemibrain," *Janelia Research Campus*, 2020. [Online]. Available: https://www.janelia.org/project-team/flyem/hemibrain.

[4] B. Caulfield, "What's the Difference Between Ray Tracing and Rasterization?," *Nvidia*, 2018. [Online]. Available: https://blogs.nvidia.com/blog/2018/03/19/whats-difference-between-ray-tracing-rasterization/.

[5] Nvidia, "NVIDIA GeForce RTX - Official Launch Event," *YouTube*, 2018. [Online]. Available: https://youtu.be/Mrixi27G9yM.

[6] sadtaco and Qesa, "Is the '10Gigarays per second' actually the *effective* Gigarays when you factor in tensor cores denoising?," *Reddit*, 2018. [Online]. Available: https://www.reddit.com/r/nvidia/comments/9a112w/is_the_10gigarays_per_second_actually_the/.

[7] S. Nuno, "Bringing Ray Tracing to Vulkan," *Khronos Group*, 2019. [Online]. Available: https://www.khronos.org/assets/uploads/developers/library/2019-reboot-develop-red/Bringing-Ray-Tracing-To-Vulkan_Oct19.pdf.

[8] C. S. Xu *et al.*, "A Connectome of the Adult Drosophila Central Brain," *bioRxiv*, p. 2020.01.21.911859, Jan. 2020.

[9] N. Bielopolski, H. Amin, A. A. Apostolopoulou, and E. Rozenfeld, "Inhibitory muscarinic acetylcholine receptors enhance aversive olfactory learning in adult Drosophila," pp. 1–24, 2019.

[10] M. J. Krashes, A. C. Keene, B. Leung, J. D. Armstrong, and S. Waddell, "Sequential use of mushroom body neuron subsets during Drosophila odor memory processing," vol. 53, no. 1, pp. 103–115, 2008.

[11] Neuronland, “SWC,” 2016. [Online]. Available: http://www.neuronland.org/NLMorphologyConverter/MorphologyFormats/SWC/Spec.html.

[12] Neuroinformatics.NL, “SWC plus (SWC+) format specification.”

[13] cnlbob and Smrfeld, “SWC Mesher,” *github*, 2017. [Online]. Available: https://github.com/mcellteam/swc_mesher.

[14] S. Kopacz, “RTX Rendering Project,” 2019. [Online]. Available: https://beskamir.github.io/projects/rendering-project.

[15] T. Stachwiak, “Stochastic all the things: Raytracing in hybrid real-time rendering,” in *Digital Dragons*, 2018.

[16] Scratchapixel, “Global Illumination in Practice: Monte Carlo Path Tracing,” *Scratchapixel 2.0*, 2016. [Online]. Available: https://www.scratchapixel.com/lessons/3d-basic-rendering/global-illumination-path-tracing/global-illumination-path-tracing-practical-implementation.

[17] S. Kopacz, “Spacecraft,” *ArtStation*, 2018. [Online]. Available: https://www.artstation.com/artwork/y969K.

[18] Blender, “Fresnel Node,” *Blender*, 2020. .

[19] Y. Chernikov, “Hazel Engine,” *github*, 2020. [Online]. Available: https://github.com/TheCherno/Hazel.

[20] A. Overvoorde, “Vulkan Tutorial,” *vulkan-tutorial*, 2017. [Online]. Available: https://vulkan-tutorial.com/.

[21] S. Willems, “Vulkan C++ examples and demos,” *github*, 2020. [Online]. Available: https://github.com/SaschaWillems/Vulkan.

[22] M.-K. Lefrançois and P. Gautron, “NVIDIA Vulkan Ray Tracing Tutorial,” *Nvidia*, 2019. [Online]. Available: https://developer.nvidia.com/rtx/raytracing/vkray.

[23] J. Boksansky, “RTAO,” *github*, 2019. .