Article Type: Theme Article

On the Impacts of Spherical Harmonics and Gaussian Counts in WebGL-based 3D Gaussian Splatting

Mufeng Zhu, *Rutgers University, Piscataway, NJ, 08854, USA*

Yao Liu, *Rutgers University, Piscataway, NJ, 08854, USA*

Abstract—3D Gaussian Splatting (3DGS) is an emerging technique for training and representing photorealistic real-world 3D scenes. Due to its high rendering speed (e.g., over 100 frames-per-second on a CUDA-enabled GPU), it has been viewed as a game-changing representation for immersive media. One popular application of 3DGS is streaming and rendering 3D Gaussians directly in modern browsers via WebGL. The large number of Gaussians in a trained 3DGS scene and their attributes, particularly the spherical harmonics for view-dependent color, make optimizing WebGL rendering efficiency a significant challenge. This paper describes a WebGL-based web viewer that supports rendering 3DGS scenes with different spherical harmonics (SH) levels. We systematically evaluate the impact of SH levels and Gaussian count on rendering performance, in terms of visual quality and frame rates. Our framework provides researchers with a platform to explore optimization techniques for efficient web-based 3DGS rendering.

Immersive technologies have experienced substantial growth in recent years, along with the emergence of advanced VR/AR hardware like Apple Vision Pro and Meta Quest devices. This market expansion has increased the demand for immersive applications, particularly for free-world virtual navigation of reconstructed real-world scenes in 6-degrees-of-freedom (6-DoF). Although point clouds and triangular meshes are traditional 6-DoF representations, they often suffer from low reconstruction quality for real-world scenes. Point cloud acquisition, typically through LiDAR or photogrammetry techniques, faces significant challenges including sensor-introduced noise and inconsistent point distribution problems that result in geometric inaccuracies and visible "holes" in rendered views.[1] Similarly, mesh-based representations – usually derived from point data through complex surface reconstruction algorithms – struggle to achieve photorealistic quality, especially when representing real-world geometries and material properties.[2]

Recently, learning-based approaches for 3D reconstruction and novel view synthesis have emerged, with Neural Radiance Field[3](NeRF) and 3D Gaussian Splatting[4] (3DGS) being the two leading methods, alongside their follow-up works. NeRF represents 3D scenes as neural networks, transforming 2D images into immersive 3D representations. While NeRF can generate renderings of photorealistic visual quality, its primary limitation is the computationally intensive neural network inference required for rendering, resulting in very low frame rates. Compared to NeRF, 3DGS enables real-time rendering, often achieving over 100 frames-per-second rendering speed on a CUDA-enabled GPU, while delivering higher or similar visual quality.

3DGS represents the scene using a set of anisotropic 3D Gaussians. Thus, it is a point-based representation. However, different from traditional point clouds, each 3D Gaussian contains more attributes, including its position, scale and quaternion for the covariance matrix, opacity, and color attributes. Thresholding the underlying Gaussian distribution, e.g., at 3σ, defines the volume and surface of a 3D ellipsoid. Using a point cloud generated from Structure-from-Motion[5](SfM) as input, these Gaussian attributes are treated as learnable parameters and fine-tuned via

Digital Object Identifier 10.1109/XXX.0000.0000000

gradient descent, as shown in Figure 1. During rendering, each 3D Gaussian is projected to a 2D splat and thus occupies several pixels. The color of each pixel is ultimately calculated from the contributing 2D splats through alpha blending.

Although 3DGS achieves photorealistic novel views at high rendering speeds, its large file size constrains practical implementation in immersive applications. A significant factor contributing to this size issue is the spherical harmonic coefficients used for view-dependent color calculations for each Gaussian. The current 3DGS training pipeline supports spherical harmonics (SH) up to level 3. At SH level 3, SH coefficients include 3 sets of 16 `float32` values (one set per color channel), consuming 192 bytes out of the total 236 bytes of attributes per Gaussian. As a result, the trained `garden` scene with over 5.8 million Gaussians has a large storage size of 1.4 GBytes. On one hand, this creates challenges for transmitting 3D Gaussian scenes over real-world networks. Current solutions such as *SGSS*[6] and *L3GS*[7] focus on optimizing the downloading process for streaming 3D Gaussian content. On the other hand, while 3DGS achieves extremely high rendering frame rates through its fast differentiable rasterizer on CUDA, the complexity of 3D Gaussian rendering – especially the view-dependent color calculation and Gaussian depth sorting – creates a significant bottleneck in environments without CUDA acceleration, such as web browsers. To address this challenge, existing solutions either assign different SH levels to 3D Gaussians in the scene adaptively[8] or simply discard higher-order SH coefficients beyond level 0,[9] known as diffuse color. None of these solutions systematically evaluates how different SH levels affect the 3DGS rendering performance in web browsers.

In this paper, we describe a web viewer that supports rendering 3D Gaussians with varying SH levels (0 through 3), using WebGL2. We then conduct experiments measuring both visual quality and rendering speed in browser environments. Our implementation aims to offer researchers a flexible platform for developing and evaluating optimization strategies that balance visual quality and computational efficiency in web-based 3D Gaussian Splatting.

Background and Related Work

The 3DGS Representation. A 3DGS scene consists of numerous 3D Gaussians whose parameters are optimized through backpropagation. Each Gaussian is defined by its position $\boldsymbol{\mu} \in \mathbb{R}^3$, covariance matrix $\Sigma \in \mathbb{R}^{3\times3}$ (parameterized as scale $\mathbf{s} \in \mathbb{R}^3$ and rotation quaternion $\mathbf{q} \in \mathbb{R}^4$), opacity $o \in \mathbb{R}$, and view-dependent color. For efficient storage and optimization, the covariance matrix follows $\Sigma = RSS^TR^T$, where $S = \text{diag}(\mathbf{s})$, and R is the rotation matrix derived from $\mathbf{q}$.

Input point cloud Gaussian ellipsoid Rendered 3DGS

FIGURE 1. Visualized generation of 3D Gaussians for scene `kitchen`,[10] from input point cloud to ellipsoid to rendered 3D Gaussians.

View-dependent color is encoded using spherical harmonics (SH), requiring $(k+1)^2$ coefficients per color channel at SH level k. Spherical harmonics are incrementally progressive. That is, higher-level SH representations are super-sets of lower-level ones, and SH level k contains $2k+1$ additional coefficients compared to SH level $k-1$. Spherical harmonics are basis functions defined on the surface of a sphere, enabling efficient representation of how appearance varies with viewing direction. Higher SH levels capture increasingly detailed directional variations by representing higher frequency signals in the spherical domain. SH0 represents diffuse (non-directional) color which is calculated as

$$\text{RGB} = \text{SH0} \times \text{C0} + 0.5 \tag{1}$$

where C0 = 0.28209479177387814, and SH0 includes one coefficient for each of the RGB color channels.

For higher SH levels, color computation incorporates view direction vectors combined with additional SH coefficients through a series of mathematical operations. A direct consequence of using higher SH levels is the significant increase in storage requirements. At SH level 3, this amounts to 48 coefficients across RGB channels. In total, each Gaussian requires 59 floating-point values (236 bytes), with SH coefficients constituting the majority (192 bytes).

Rendering Process. During rendering, 3D Gaussians project to 2D splats via the transformation $\Sigma' = JW\Sigma W^TJ^T$, where Σ' is the resulting 2D covariance

matrix, W is the view transformation matrix, and J approximates perspective projection. For a pixel i, overlapping Gaussians are depth-sorted and blended according to the following equation:

$$C_i = \sum_{n \leq \mathcal{N}} \left(c_n \cdot \alpha_n \cdot \prod_{m<n} (1 - \alpha_m) \right) \quad (2)$$

where $\mathcal{N}$ is the number of Gaussians that overlap pixel i, and α_n is derived from the 2D covariance matrix and opacity, determining each Gaussian's contribution to the final pixel color.

Web-Based Gaussian Splat Viewer. To date, several web-based Gaussian Splat viewers are available without relying on CUDA acceleration. Early implementations such as *gsplat*[11] and *splat*[9] render 3D Gaussian scenes using WebGL but omit spherical harmonics calculations, computing only diffuse colors (i.e., SH level 0) for each Gaussian and storing attributes in binary format. *GaussianSplat3D*[12] utilizes `three.js` for rendering and introduces a customized `ksplat` format that applies codebook quantization to raw 3DGS scenes. Similarly, *reducing 3DGS*[8] provides an interactive WebGL viewer employing codebook quantization, but additionally reduces the total Gaussian count and adaptively assigns varying SH levels to different Gaussians based on their visual importance. *SGSS*[6] focuses on optimizing the network transmission of 3DGS scenes rather than introducing novel rendering techniques. It uses the similar WebGL-based rendering approach as *splat*[9] without significant modifications to how Gaussians are rendered in the browser.

However, none of these implementations systematically investigate how scene characteristics affect web-based rendering performance. Specifically, prior work has not thoroughly examined the relationship between rendering efficiency and scene parameters such as the total number of Gaussians and the impact of varying spherical harmonics levels. Our analysis in this paper addresses this gap by quantifying how these factors directly influence achievable frame rates in browser-based environments, providing important insights for optimizing web-based 3D Gaussian rendering systems.

Supporting Higher SH Levels for View-Dependent Color in Web Viewers

Figure 2 illustrates the workflow of rendering 3DGS scenes using WebGL2 in the web browser, as implemented in the *splat* framework. Raw 3DGS scenes, as they are trained, are in the `.ply` format. The raw representation first undergoes a preprocessing step to a binary data array. During this step, Gaussians may be sorted based on an importance metric,[9] raw opacity values are transformed using the sigmoid function, and raw scale values are transformed using the exponential function.

A web worker then generates a WebGL-compatible texture from the preprocessed array. In this texture, attributes of each Gaussian are stored as a texel, which can be fetched using WebGL's `texelFetch()` function. The web worker returns the generated texture to the main thread, which subsequently uploads the texture data to the GPU for rendering. Concurrently, the web worker receives continuous viewport updates from the main thread and performs depth-based sorting of 3D Gaussians. It returns the sorted indices to the main thread, which then transfers them to the GPU. The GPU fetches Gaussian attributes from the texture according to these depth-sorted indices using the defined vertex shader program.[13] This shader represents each projected 3D Gaussian (2D splat) as a quad primitive in WebGL, calculating vertex positions and passing this information along with color data to the fragment shader program. The fragment shader program then computes the opacity for each fragment. This is achieved by multiplying the per-Gaussian opacity with the projected 2D Gaussian evaluated at the point, thereby generating the final color values in the RGBA (four channels). Through this pipeline, 3D Gaussians are efficiently rendered and blended in the web browser, with careful coordination between JavaScript threads and GPU processing.

While the *splat* web viewer can only support SH level 0, we extended this framework to support higher SH levels. The primary modifications occur in texture generation and vertex shader implementation. The increased number of attributes introduced by higher SH coefficients necessitates appropriately sized textures to accommodate this additional data. At different SH levels, each Gaussian requires varying numbers of floating-point attributes (with all Gaussian attributes, position, scale, quaternion, opacity, and SH coefficients, included): 59 for SH3, 38 for SH2, and 23 for SH1, all stored in `float32` format. The *splat* implementation employs bit-shifting operations in texture lookup calculations to enhance computational efficiency. This introduces a constraint that storage allocation for each Gaussian must be a power-of-2 integer. Therefore, we allocate 64 `float32` values for both SH3 and SH2, and 32 `float32` values for SH1. This memory alignment requirement results in some storage inefficiency but ensures compatibility with WebGL's texture addressing mechanisms. Additionally, we

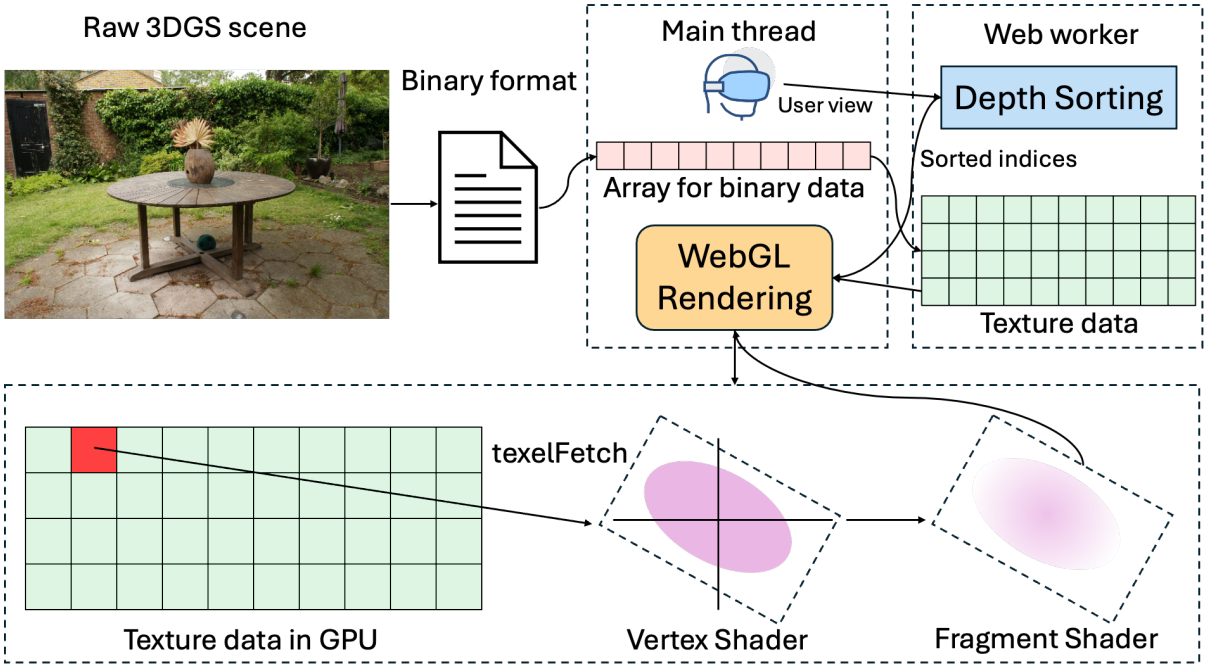

FIGURE 2. Workflow of rendering 3D Gaussian scenes in a web browser using WebGL2. The example raw 3DGS scene is `garden`.[10]

implement the view-dependent color calculation from SH coefficients in the vertex shader, following the code of the original 3DGS work. This ensures consistency with the reference implementation while enabling real-time computation of directional appearance effects during rendering.

Performance Impacts of Spherical Harmonics and Gaussian Count

In our experiments, we use 13 pre-trained 3D Gaussian Splatting scenes from the original 3DGS work, comprising 6 indoor scenes and 7 outdoor environments. We augment this dataset with two additional indoor scenes (`nyc` and `berlin`) from the Zip-NeRF dataset,[14] which we trained for our evaluation. We excluded other Zip-NeRF scenes that contain mixed indoor and outdoor environments to maintain environmental consistency within our analysis. Throughout our evaluation, we leverage the *SGSS* codebase which stores all data in `float32` format, ensuring fair comparisons across different SH level configurations.

Impact of SH Level on Visual Quality

Experiment Setup. To evaluate the effects of different SH levels on visual quality, we first generate camera traces following the orbital trace methodology from *SGSS*'s evaluation. The orbital trace simulates user navigation through the scene over an 80-second period, traversing 40 camera poses from each scene's training dataset at intervals of 2 seconds. For each scene, we preprocess and save individual binary formats at four distinct SH level configurations (recall that spherical harmonics are incrementally progressive): SH0 only, SH1, SH2, and SH3 (the complete model). This preprocessing approach enables direct evaluation of the incremental visual and performance impacts as SH complexity increases. For each configuration (scene+SH level), we first ensure complete loading of the binary file and initialization of all WebGL rendering texture before navigation starts. We then explore the scene using the orbital trace, capturing rendered frames in lossless `PNG` format when the camera is located at one of the poses used to generate orbital trace. Ground truth image is available for visual quality evaluation at these poses.

Results. We evaluate visual quality by calculating Peak Signal-to-Noise Ratio (PSNR) between rendered views and ground truth for each SH configuration. Figure 3 illustrates the PSNR improvement from SH0 to SH3 across all scenes. From the result, we can observe that higher SH levels generally enhance visual quality, though the contribution of each successive level

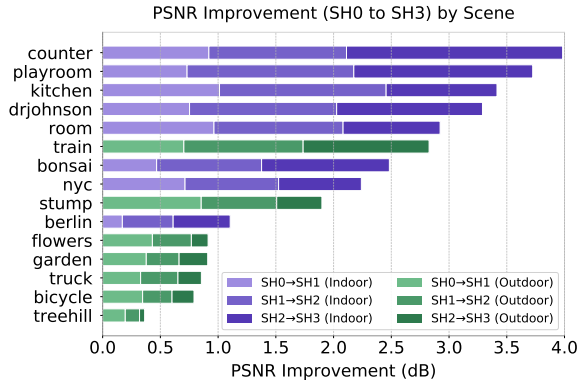

FIGURE 3. Visual quality improvement from SH0 to SH3 in indoor and outdoor scenes.

varies significantly between scenes. For instance, the `counter` scene shows the most improvement from SH2 to SH3, while `treehill` exhibits the minimal.

From the result, we can see indoor scenes (represented by purple bars) demonstrate significantly higher sensitivity to SH level changes, with six of the top seven PSNR improvements occurring in indoor environments. This difference indicates indoor scenes containing more complex light interactions across different view directions, requiring higher-order SH coefficients to accurately represent view-dependent colors. This finding suggests a potential optimization strategy: higher SH levels (SH2 or SH3) should be prioritized for indoor environments to maintain rendering accuracy, while outdoor scenes can utilize lower SH levels (SH0 or SH1) with limited quality degradation, as some of their PSNR improvement from SH0 to SH3 typically remains below 1 dB.

Impact of SH Level on Rendering Speed

Measuring accurate rendering speed in web-based viewers presents several technical challenges. The rendering performance is primarily controlled by two key factors: the monitor's refresh rate and the JavaScript `requestAnimationFrame()`[15] API. Due to vertical synchronization (VSync), a monitor with a 60Hz refresh rate imposes an upper limit of 60 frames-per-second (FPS), even when the frame processing time is less than 16.67 ms. The `requestAnimationFrame()` API optimizes rendering performance by balancing multiple factors, including the GPU frame processing time, and dynamically adjusts the rendering frequency of 3DGS scenes. For instance, while the actual frame latency might be 29 ms (theoretically corresponding to 34.4 FPS), the `requestAnimationFrame()` scheduling may regulate the effective frame rate to 30 FPS. Considering these challenges, our experimental results only reported four discrete refresh rates: 15 FPS, 20 FPS, 30 FPS, and 60 FPS, because we are using a display with a 60Hz refresh rate.

TABLE 1. Rendering Performance (FPS) with Different SH Levels and Gaussian Counts

# Gaussians	SH3	SH2	SH1	SH0
>3.5M	<15	<20	<60	<60
3M-3.5M	15	30	60	60
2.5M-3M	30	60	60	60
<2.5M	60	60	60	60

In our evaluation of various scenes with different SH levels, we observed significant performance variations. Some scenes maintained 60 FPS regardless of SH levels, while others exhibited substantial performance degradation, dropping from 60 FPS at SH0 to as low as 15 FPS at SH3. A primary factor contributing to this performance difference is the number of Gaussians rendered in each scene. Higher SH levels require more complex view-dependent color calculations, increasing the computational load. This performance impact is particularly noticeable in scenes with a large number of Gaussians. The rendering speed is thus affected by both the SH level complexity and the total number of Gaussians requiring processing.

To understand the impact of SH level and the number of Gaussians on the rendering performance, we first conducted the following experiments. We select the `garden` scene, which contains nearly 5.8 million Gaussians, and use the Importance Sorting from *SGSS* to rank the Gaussians over the full scene. For each SH level, we extracted the first 2.5 million, 2.75 million, 3 million, 3.25 million, and 3.5 million Gaussians in the scene for rendering. In the web viewer, for each configuration, we moved the camera far away from the center of the scene to ensure all the Gaussians are visible in the viewport and record the frame rate.

We ran the experiment on Alienware Aurora Ryzen Edition R14 with Nvidia RTX 3090. Note that although the GPU on our experiment setup supports CUDA, our rendering was WebGL-based and did not use CUDA.

Table 1 shows the relationship between Gaussian count, SH level, and the rendering performance. When rendering between 3 million and 3.5 million Gaussians, using SH levels higher than 2 causes a significant performance degradation. This demonstrates that complex view-dependent color calculations in WebGL substantially impact the rendering efficiency. Additionally,

maintaining a fixed SH level while increasing the Gaussian count leads to notable performance reductions. Our results also indicate that at SH levels below 2, rendering performance remains relatively resilient to the increases in Gaussian count. During our experiments, scenes with fewer Gaussians than the `garden` scene consistently maintained 60 FPS performance. We however acknowledge that the rendering speed may degrade when the Gaussian count significantly increases. We therefore present the FPS as "$<$ 60" in Table 1 when total rendered Gaussian count is larger than 3.5 million.

Impact of Gaussian Count and Visible Gaussians on Rendering Speed

During our evaluation, we also observed an interesting finding. Even when rendering approximately the same number of visible Gaussians at SH3 level within comparable viewports, different scenes exhibited markedly different rendering speeds. For example, when rendering approximately 1.3 million Gaussians in both the `garden` and `truck` scenes, the `garden` scene achieved only 20 FPS while the `truck` scene maintained 60 FPS.

The key difference between these two scenes is the total number of Gaussians in the complete scene: `garden` contains over 5.8 million Gaussians while `truck` contains only 2.5 million Gaussians. We duplicated all Gaussians in the `truck` scene while preserving their individual attributes, effectively doubling the number of Gaussians in the scene to 5 million. When rendering approximately 1.3 million visible Gaussians in the duplicated `truck` scene, the rendering speed decreased significantly from 60 FPS to 20 FPS, matching the performance observed in the `garden` scene.

Therefore, we evaluate how the number of visible Gaussians impacts rendering speed when the full scene contains over 5 million Gaussians. Since this effect is most significant when rendering at SH level 3, we present results exclusively for this configuration. We initialized the camera position to view an empty region, then gradually adjusted the viewport to incrementally increase the number of visible Gaussians while recording the corresponding changes in frame rate.

Figure 4 shows the change of frame rate when rendering increasing number of Gaussians in the scene with over 5 million Gaussians. We observed that the frame rate drops to a lower performance tier with each additional increment of approximately 0.5 million visible Gaussians. Determining the precise threshold boundaries proved challenging, as even minor viewport adjustments can significantly alter the number of visible

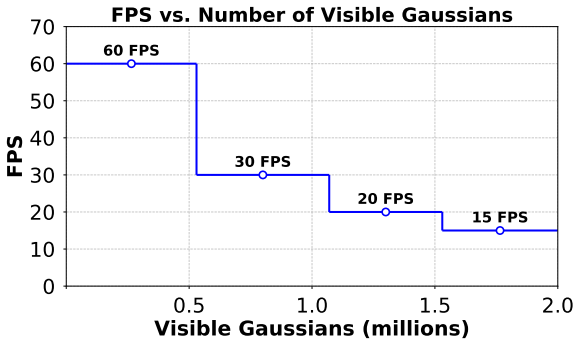

FIGURE 4. Rendering speed relationship with visible Gaussian count for scenes containing over 5 million total Gaussians at SH level 3.

Gaussians in the scene. This result indicates that rendering performance is tightly coupled with the total number of Gaussians in the scene, not just those visible in the viewport. Considering the WebGL rendering pipeline, the depth sorting indices must be transmitted to the vertex shader to enable proper fetching and blending of Gaussians in the correct order. Consequently, retrieving 1.3 million Gaussians from a scene containing 5 million Gaussians is significantly more time-consuming than retrieving the same number from a scene with only 2 million Gaussians.

Discussion

Our systematic performance evaluation revealed that indoor scenes exhibit more sensitivity to spherical harmonic (SH) levels than outdoor scenes. This is likely due to the complex lighting interactions and varied surface materials typically present in indoor environments, where accurate directional appearance effects are essential for achieving photorealistic rendering quality.

Furthermore, our evaluation results indicate that WebGL-based rendering speed is influenced by multiple factors beyond just the SH level, but also includes the number of visible Gaussians in the viewport and the total Gaussian count in the full scene. When rendering scenes containing millions of Gaussians, higher SH levels lead to significant frame rate reductions due to increased computational complexity. Notably, even when rendering an identical number of visible Gaussians, scenes with larger total Gaussian counts can exhibit significantly lower frame rates, due to more complex and time-consuming GPU operations to access the data.

Based on our findings, there are several potential optimization directions for web-based 3DGS rendering.

Intuitively, our experiments demonstrate that web viewers can effectively render a few million Gaussians at high frame rates under optimal conditions. However, as the total scene complexity increases, even Gaussians not contributing to the current view can negatively impact performance. This observation motivates the development of specialized optimization schemes and algorithms to reduce the complexity of GPU operations for scenes with an extremely large number of Gaussians. Recently, EyeNavGS[16] proposed a user navigation dataset for 3DGS scenes, which can significantly benefit research in this direction.

Additionally, the current strict constraint that storage allocation for each Gaussian in GPU textures must be a power-of-2 integer, enforced by the bit-shifting operations used for efficient texture lookups, introduces significant memory inefficiency in texture storage. For example, SH2 representation requires 38 floating-point values but must allocate 64 slots, resulting in nearly 40% unused memory across potentially millions of Gaussians. Therefore, developing more efficient data structures or storage methods for GPU textures is also a valuable research direction.

Conclusion

In this paper, we extended existing 3D Gaussian Splatting web viewers to support rendering with higher spherical harmonics levels, enabling more accurate view-dependent color calculation in browser-based environments. Through comprehensive evaluations, we characterized the impacts of varying SH levels and Gaussian counts on rendering performance. We hope our framework and experimental findings will help researchers in developing more efficient approaches to web-based 3D Gaussian rendering, ultimately enabling more complex and visually enriching immersive experiences within browser environments.

Acknowledgments

We appreciate constructive comments from anonymous referees. This work is partially supported by NSF under grant CNS-2200048.

REFERENCES

1. M. Zhu, Y.-C. Sun, N. Li, J. Zhou, S. Chen, C.-H. Hsu, and Y. Liu, “Dynamic 6-DoF Volumetric Video Generation: Software Toolkit and Dataset,” in *2024 IEEE 26th International Workshop on Multimedia Signal Processing (MMSP)*. IEEE, 2024, pp. 1–6.
2. R. Sulzer, R. Marlet, B. Vallet, and L. Landrieu, “A survey and benchmark of automatic surface reconstruction from point clouds,” *arXiv preprint arXiv:2301.13656*, 2023.
3. B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, “Nerf: Representing scenes as neural radiance fields for view synthesis,” in *ECCV*, 2020.
4. B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, “3d gaussian splatting for real-time radiance field rendering,” *ACM Transactions on Graphics*, vol. 42, no. 4, pp. 1–14, 2023.
5. N. Snavely, S. M. Seitz, and R. Szeliski, “Photo tourism: exploring photo collections in 3d,” in *ACM siggraph 2006 papers*, 2006, pp. 835–846.
6. M. Zhu, M. Liu, C. Yu, C.-H. Hsu, and Y. Liu, “SGSS: Streaming 6-DoF Navigation of Gaussian Splat Scenes,” in *Proceedings of the 16th ACM Multimedia Systems Conference*, 2025, pp. 46–56.
7. Y.-Z. Tsai, X. Zhang, Z. Li, and J. Chen, “L3GS: Layered 3D Gaussian Splats for Efficient 3D Scene Delivery,” *arXiv preprint arXiv:2504.05517*, 2025.
8. P. Papantonakis, G. Kopanas, B. Kerbl, A. Lanvin, and G. Drettakis, “Reducing the Memory Footprint of 3D Gaussian Splatting,” *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, vol. 7, no. 1, pp. 1–17, 2024.
9. “Github: antimatter15/splat,” https://github.com/antimatter15/splat.
10. J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, “Mip-nerf 360: Unbounded anti-aliased neural radiance fields,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 5470–5479.
11. “gsplat — 3D Gaussian Splatting WebGL viewer ,” https://gsplat.tech/.
12. “Github: mkkellogg/GaussianSplats3D,” https://github.com/mkkellogg/GaussianSplats3D.
13. “WebGL Shaders and GLSL ,” https://webglfundamentals.org/webgl/lessons/webgl-shaders-and-glsl.html.
14. J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, “Zip-NeRF: Anti-Aliased Grid-Based Neural Radiance Fields,” *ICCV*, 2023.
15. “Window: requestAnimationFrame() method ,” https://developer.mozilla.org/en-US/docs/Web/API/Window/requestAnimationFrame.
16. Z. Ding, C.-T. Lee, M. Zhu, T. Guan, Y.-C. Sun, C.-H. Hsu, and Y. Liu, “EyeNavGS: A 6-DoF Navigation Dataset and Record-n-Replay Software for Real-World 3DGS Scenes in VR,” *arXiv preprint arXiv:2506.02380*, 2025.

Mufeng Zhu is a PhD student in Electrical and Computer Engineering at Rutgers University, supervised by Prof. Yao Liu. His research focuses on immersive video streaming technologies, including streaming 3D Gaussian Splatting and Neural Radiance Fields (NeRF). His research projects address system challenges in delivering immersive experiences, such as bandwidth and computation limitations. He received his M.S. degree in Electrical and Computer Engineering from Rutgers University in 2022. Contact him at mz526@scarletmail.rutgers.edu.

Yao Liu is an Assistant Professor in the ECE Department at Rutgers University. Her research interests include immersive computing, multimedia systems, mobile computing, and edge/cloud computing. She received her PhD degree in Computer Science from George Mason University. Contact her at yaoliu@rutgers.edu.