Large-Scale 3D Terrain Reconstruction Using 3D Gaussian Splatting for Visualization and Simulation

Meida Chen [1], Devashish Lal [1], Zifan Yu [2], Jiuyi Xu [1], Andrew Feng [1], Suya You [3], Abdul Nurunnabi [4], Yangming Shi [5]

[1] Institute for Creative Technologies, University of Southern California, Los Angeles, CA 90094, USA – (mechen, dlal, jiuxu, feng)@ict.usc.edu
[2] Department of Computer Science, Arizona State University, Arizona – zifanyu@asu.edu
[3] DEVCOM Army Research Laboratory – suya.you.civ@army.mil
[4] Geodesy and Geospatial Engineering, Faculty of Science, Technology and Medicine, University of Luxembourg – abdul.nurunnabi@uni.lu
[5] the Department of Civil and Environmental Engineering, Colorado School of Mines – yangming.shi@mines.edu

Keywords: 3D Gaussian Splatting, Large-scale Terrain Reconstruction, 3D Visualization, Game Development, Simulation.

Abstract

The fusion of low-cost unmanned aerial systems (UAS) with advanced photogrammetric techniques has revolutionized 3D terrain reconstruction, enabling the automated creation of detailed models. Concurrently, the advent of 3D Gaussian Splatting has introduced a paradigm shift in 3D data representation, offering visually realistic renditions distinct from traditional polygon-based models. Our research builds upon this foundation, aiming to integrate Gaussian Splatting into interactive simulations for immersive virtual environments. We address challenges such as collision detection by adopting a hybrid approach, combining Gaussian Splatting with photogrammetry-derived meshes. Through comprehensive experimentation covering varying terrain sizes and Gaussian densities, we evaluate scalability, performance, and limitations. Our findings contribute to advancing the use of advanced computer graphics techniques for enhanced 3D terrain visualization and simulation.

1. Introduction

The integration of low-cost unmanned aerial systems (UAS) equipped with high-resolution cameras, alongside advances in photogrammetric techniques, has significantly transformed the domain of 3D terrain reconstruction. This evolution has facilitated the automated creation of detailed and realistic 3D meshes, marking a pivotal shift towards enhancing the accuracy and realism of digital terrain models across various applications.

Within this innovative landscape, a compelling development has emerged from the realm of computer graphics: the invention of 3D Gaussian Splatting (Kerbl, Bernhard, et al. 2023). This novel technique, which represents a significant advancement in the representation of 3D data, offers a visually realistic rendering that stands apart from traditional polygon-based models. 3D Gaussian Splatting demonstrates a superior capability for generating visually realistic outputs, positioning it as a valuable asset for applications that demand high-fidelity visualizations, particularly in the creation of virtual gaming and simulation environments.

Building upon the foundational advancements introduced by 3D Gaussian Splatting, our research endeavors to bridge the gap between high-quality visualization and interactive simulation capabilities. This integration is critical for leveraging the full potential of Gaussian Splatting within dynamic and immersive applications. Our investigation focuses on the application of this technique for reconstructing large-scale 3D terrains, thoroughly examining its performance and effectiveness in creating virtual gaming environments. In particular, the point cloud nature of Gaussian Splatting presents specific challenges, notably its incompatibility with interactive simulations that require collision detection. To address this, our research adopts a hybrid methodology, leveraging photogrammetry-derived meshes as invisible collision substrates beneath the Gaussian Splatting visual layer. This approach seeks to harness the visual advantages of Gaussian Splatting while ensuring the dynamic interactions essential for an immersive simulation environment.

Our experimentation covers a range of terrain sizes and tests the Gaussian Splatting technique with varying numbers of 3D Gaussians, from one million up to 28 million. This comprehensive analysis seeks to assess the scalability, performance, and potential limitations of using Gaussian Splatting for the visualization of large-scale 3D terrains within interactive simulations. Through this paper, we present our methodology, experimental setup, and the insights gained, contributing to the broader discussion on leveraging advanced computer graphics techniques for enhanced 3D terrain visualization and simulation.

2. Literature Review

2.1 3D Gaussian Splatting

The seminal work on Gaussian splatting introduced an innovative approach to view synthesis, culminating in the achievement of real-time rendering of 1080p resolution at 30 frames per second, concurrently obtaining comparable visual fidelity to conventional neural radiance field methodologies (Kerbl, Bernhard, et al. 2023). Gaussian splatting methodology integrates three-dimensional Gaussian distributions to depict the scene, thereby retaining the advantageous attributes of continuous volumetric radiance fields for scene optimization while circumventing superfluous computational overhead in void regions. Subsequent to the representation of the scene via three-dimensional Gaussians, an interleaved optimization and density control process is employed, which iteratively refines three-dimensional position, opacity, anisotropic covariance, and spherical harmonics parameters to establish a faithful rendition of the scene. Conclusively, a tile-based rendering approach, cognizant of visibility considerations, is implemented to facilitate anisotropic splatting, thereby expediting training and enabling real-time rendering capabilities. Many follow-up works have occurred just for the past year that enabling different applications such as relighting, meshing, SLAM, etc. shows great potential of further R&D on the topic.

2.2 Real-Time Terrain Systems

The domain of computer graphics has witnessed substantial progress in terrain systems, particularly propelled by advancements in game engine technology. These systems have enabled real-time rendering of expansive environments through the adoption of tiling and streaming methodologies. Notably, Unreal Engine 5 has spearheaded numerous enhancements in its terrain authoring toolkit, notably with the incorporation of world partitioning, streamlining the creation of terrain tiles. This feature dynamically regulates the streaming of tiles based on the camera's perspective, facilitating the seamless exploration of vast terrains in real-time. The fidelity of rendered terrain is intricately linked to the density of the underlying mesh, with heightened fidelity necessitating exponentially escalating vertex counts. To mitigate this computational burden, level-of-detail systems are conventionally employed, albeit with potential implications for rendering quality. Texture mapping terrain poses another formidable challenge, addressed through the utilization of terrain materials that amalgamate multiple shaders in a stratified arrangement. This framework enables the texturing of a singular terrain surface with diverse textures, governed by procedural mechanisms that leverage surface information at discrete points. Gaussian splatting emerges as a leading-edge technique for terrain representation, imbuing color information via the acquired spherical harmonics. To tackle the challenges inherent in managing large-scale terrains, a tile-based optimization strategy is embraced, complemented by a partitioned rendering scheme. This innovation effectively circumvents the particle count limitations inherent in a singular Niagara particle system, which constitutes the backbone of the simulation-enabled renderer.

3. Method

3D Gaussian Splatting employs the output of Structure-from-Motion (SfM), specifically camera poses and a sparse point cloud, as its foundational input. This method models scenes through the deployment of 3D Gaussians, each defined by parameters such as position (x,y,z), scale (indicating the Gaussian's size), rotation, an alpha value (α) determining its transparency, and spherical harmonics (SH) for view-dependent color representation. It offers an expressive framework, wherein each Gaussian parameter is subject to direct optimization during the training process, enhancing the accuracy and realism of the rendered scene. Central to Gaussian Splatting is the development of a fast-differentiable rasterizer that facilitates the projection of 3D Gaussians onto a 2D plane, enabling the comprehensive training of all Gaussian parameters.

Results from applying Gaussian Splatting to existing datasets reveal its capacity for achieving quality on par with or surpassing that of previous state-of-the-art implicit radiance field methods. However, tackling **Large-Scale Terrain** reconstruction using Gaussian Splatting presents its own set of challenges such as hardware constraints including limited VRAM and RAM. Furthermore, the adequacy of training iterations becomes a concern, as a large dataset of images may require more iterations for optimal model refinement than is feasible without significantly extending training time or compromising on model quality. To circumvent these challenges, we have designed and implemented an adaptive tiling system that segments the entire scene into smaller, more manageable pieces. This segmentation is determined by both the quantity of images tied to each segment and the predicted count of final Gaussians within them. Specifically, our partitioning algorithm continued to split the entire 3D scene into smaller chunks with a predefined threshold of number of pixels fall into each chunk. This number of pixels threshold is determined by the RAM that is available during the training process. In addition, the source images were also cropped to smaller sizes for each tile so that only the valid pixels on the tile are loaded into the RAM which further ensure the effective uses of the limited computing resources. Consequently, our approach not only mitigates hardware limitations by distributing the computational load across multiple GPUs/nodes but also enhances the training efficiency and effectiveness of the Gaussian Splatting model across extensive terrains, ensuring consistent model performance and quality visualization across all segments.
In order to seamlessly incorporate the output of our Gaussian splatting process, represented as point clouds, into the Unreal Engine environment, a meticulous approach is undertaken. Initially, the train Gaussians are partitioned into smaller subsets, each accommodating a maximum of two million points, thereby adhering to the operational constraints of the Unreal Engine's Niagra particle system. Concurrently, the alignment of collision meshes, derived from the photogrammetry procedure, with the Gaussians is conducted, streamlining the importation and scene assembly workflows within the Unreal Engine framework. Subsequent to this preparatory phase, rendering of the Gaussians is facilitated through the utilization of a customized iteration of the particle system, facilitated by the Luma AI plugin, which furnishes enhanced simulation functionalities.

3.1 Collision Meshes

In physics engines, collisions are traditionally handled using low-fidelity convex meshes that roughly describe the bounding volume of the actual mesh. These meshes are optimized for fast and efficient collision checking by the physics engine, which cannot take advantage of highly parallel GPU computing. Unreal Engine has two ways to import colliders for meshes: a low-quality but optimized convex mesh, or a high-quality performance mesh. To match the fidelity of our terrain, we import the derived meshes from the photogrammetry process in their high-quality collider representation. While this may have a minor impact on performance, we cannot use convex meshes with our high-fidelity terrain representation using Gaussian Splatting. Our Gaussian portioning process aligns our meshes with the point clouds using world positions for vertex positions, and these meshes are finally imported as OBJ files.

For each mesh, we instantiate a Gaussian Collision actor - a custom blueprint extending the actor class - solely responsible for propagating simulation information, such as the hit point, to the corresponding Niagara system for simulation purposes.

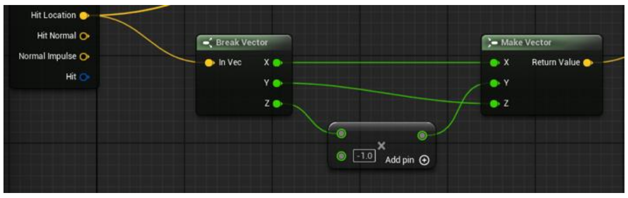

Figure 1. World space to point cloud space transformation

To ensure that the simulation effects are correctly applied within the Niagara particle systems, it is essential to convert the world positions to the coordinate system used for storing point clouds after the Gaussian splatting optimization process (Figure 1). This conversion accurately aligns the positions with the intended region and allows for seamless integration with the Unreal Engines coordinate system. At the instance level, each Gaussian collision actor maintains a list of Niagara systems that overlap the same region as the collider to ensure that the simulation information is passed to the correct instances of Niagara systems. Since updating variables in Niagara costs similar to pushing uniforms to the GPU, minimizing the number of updates is essential. Our tilling/partitioning approach helps us achieve this by allowing us to describe the mapping between Gaussian collision actor instances and Gaussian splatting Niagara renderer instances.

3.2 Destruction Simulation

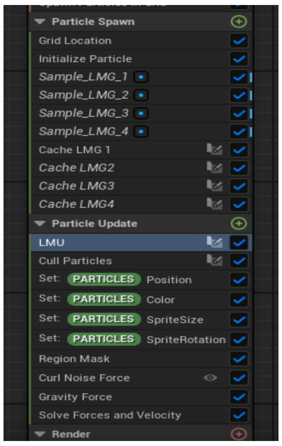
Particle Spawn
Grid Location
Initialize Particle
Sample_LMG_1
Sample_LMG_2
Sample_LMG_3
Sample_LMG_4
Cache LMG 1
Cache LMG2
Cache LMG3
Cache LMG4
Particle Update
LMU
Cull Particles
Set: PARTICLES Position
Set: PARTICLES Color
Set: PARTICLES SpriteSize
Set: PARTICLES SpriteRotation
Region Mask
Curl Noise Force
Gravity Force
Solve Forces and Velocity
Render

Figure 2. Niagara system modules

Niagara, a programmable particle system provided by Unreal Engine, can run on either CPU or GPU. While the GPU mode allows for higher particle counts and improved performance, its capabilities are limited compared to the CPU mode. Niagara's basic building block of programmable logic is called a module (Figure 2), and the Engine includes several modules out of the box for common tasks like particle color changes over time and velocity manipulation. Although these modules can be created visually in the editor, Niagara also allows for the creation of modules using HLSL code for GPU simulation. The Luma AI plugin leverages this capability by implementing various layers of the Gaussian Splatting renderer, such as frustum culling and view sorting, through Niagara modules. Modules in Niagara are registered into two lifecycle events: particle spawn and particle update. The initialization phase, during which 3D Gaussians are instantiated from the point cloud, occurs in particle spawn. The crux of the Gaussian splatting renderer is formed by modules registered in particle update. The LumaAI plugin's modules perform culling, sorting, view-dependent computation or splatting, and update each Gaussian's position, opacity, shape, and color. Each particle represents a splatted 2D Gaussian with an ellipse shape. The simulation logic is chained after the rendering logic of the LumaAI plugin. Each instance of the Niagara system exposes a hit point vector and other variables to the Gaussian collision actor. These variables are read in the destruction simulation logic to apply a combination of forces in a spherical radius. All Gaussians within this masked region have a velocity applied to simulate destruction.

The interactive capabilities of the simulation-enabled renderer, in tandem with collision meshes, provide high-fidelity interactions within large-scale Gaussian environments. Regardless of the number of simulated particles, the simulation maintains a constant cost as the calculations are performed for each Gaussian. However, only the Gaussians within the region mask exhibit non-zero velocities. This constant-time simulation is essential for conducting large-scale destruction simulations in expansive terrains.

4. Experiments

To assess Gaussian Splatting's efficacy in visualizing and simulating large-scale 3D terrains, we imported the trained models into Unreal Engine 5 (UE5) for dynamic simulations. Photogrammetry-derived meshes were utilized to construct invisible collision layers, facilitating accurate physical interactions within the terrains rendered by Gaussian Splatting. The import process was simplified and made efficient with the LumaAI UE5 plugin, ensuring the retention of GS's high visual fidelity. For added realism and interactive engagement, we leveraged UE5's Niagara System, enabling the simulation of complex destruction effects that seamlessly interact with the Gaussian models.

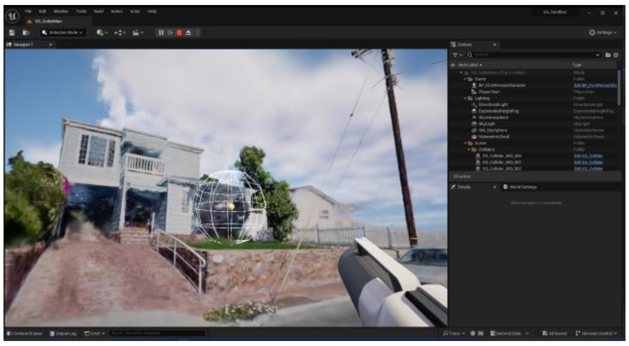
Figure 3: Environmental destruction in Soibelman Bld

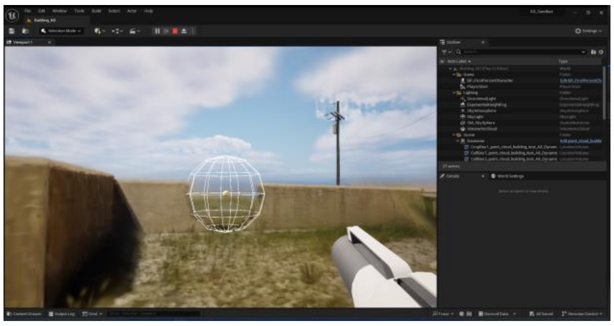
Figure 4: Wall destruction in Building scene

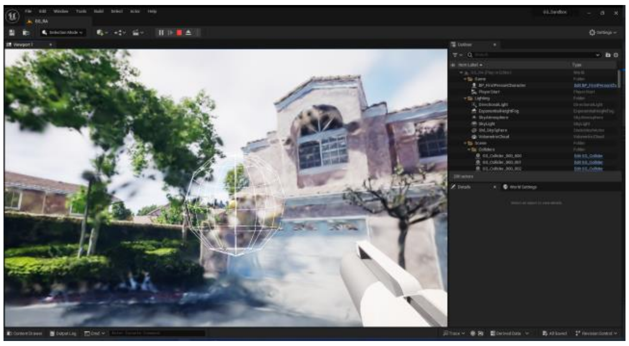
Figure 5: Destruction of a house in RA scene

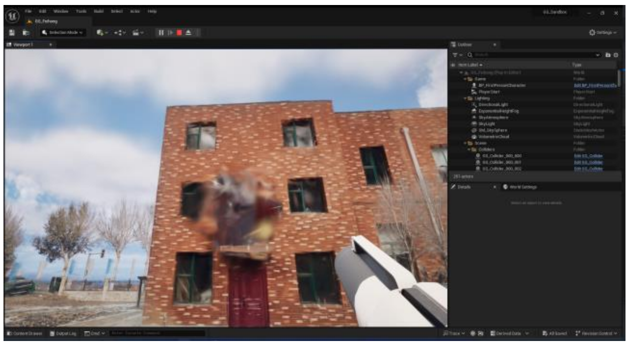
Figure 6: Destruction of a building in Feihong scene

Dataset	SIBR	Unreal Engine 5
Building (Bld)	60fps	60fps
Residential Area (RA)	30fps	16fps
Soibelman Bld	110fps	50fps
Feihong Bld	32fps	17fps

Table 1: Frame rate comparison

Dataset	SIBR	Unreal Engine 5
Building (Bld)	3.2GB	2.4GB
Residential Area (RA)	15.5GB	11.5GB
Soibelman Bld	5.2GB	4.2GB
Feihong Bld	15.6GB	12GB

Table 2: GPU memory usage comparison

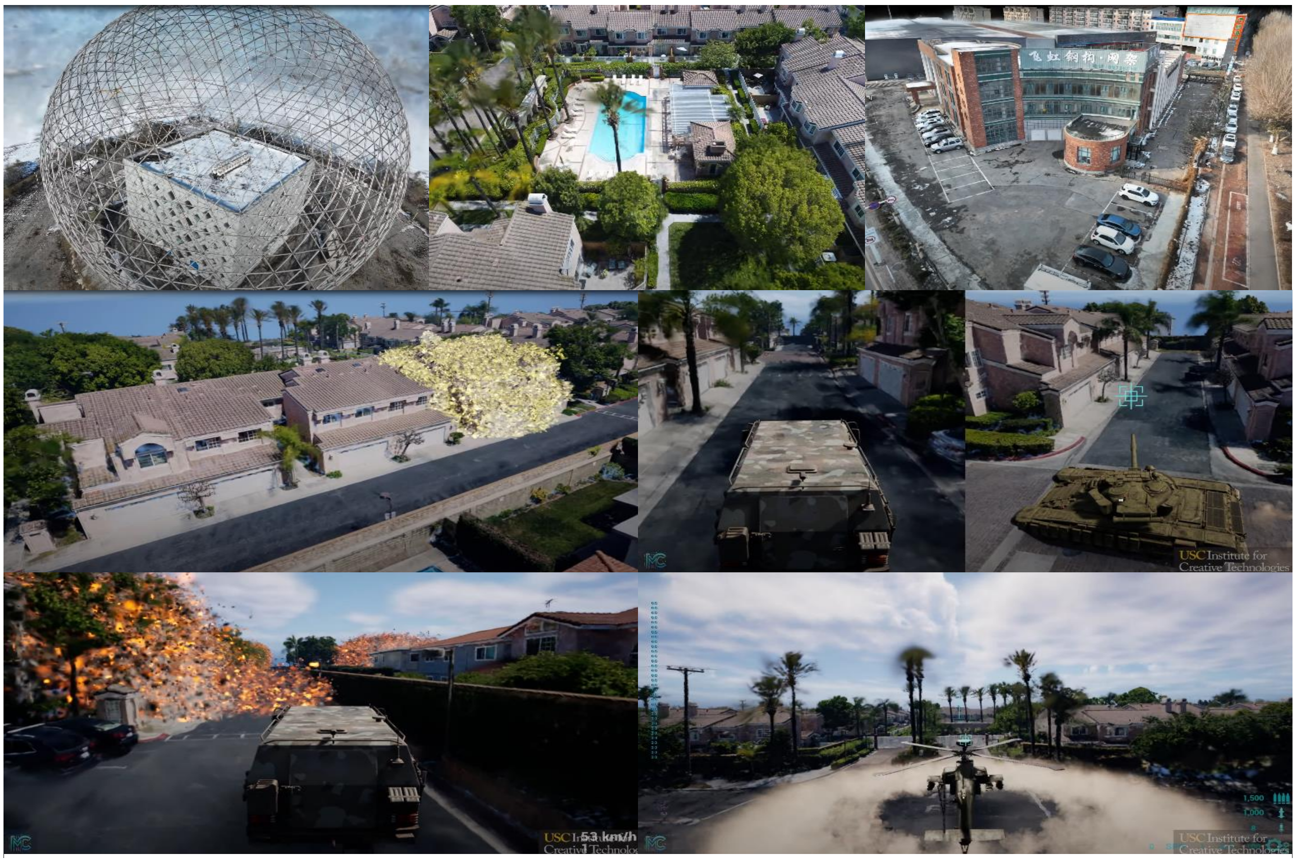
Figure 7. Visualization results

Our experiments were conducted on the real-world Residential Area (RA) as shown in Figure 3, dataset from our previously published large-scale aerial photogrammetric benchmark i.e., STPLS3D (Chen, Meida, et al. 2022), along with three newly acquired datasets that mirrored the data collection methodology of STPLS3D, as visualized in Figure 4, Figure 5 and Figure 6. These datasets, representing a variety of urban settings, were chosen to test the model's adaptability and performance across different scales and complexities of terrain. During our experiments, we documented VRAM usage as well as frames per second (FPS) in both the original Gaussian Splatting viewer i.e., SIBR and within UE5 to evaluate the model's rendering efficiency and responsiveness in real-time simulations, summarized in Table 1 and Table 2. Characteristics about the actual data are presented in Table 3. This dual-phase testing provided comprehensive insights into the model's capability to deliver high-quality visualizations and its responsiveness under interactive conditions.

Figure 7 presents our visualization results, with the top row depicting three distinct Gaussian Splatting models applied to a single building, a residential area, and an urban area featuring a mix of commercial, residential, and industrial buildings. The lower rows illustrate the visualization of the STPLS3D RA Gaussian Splatting model within Unreal Engine 5 with additional simulations. 60 fps could be achieved while visualizing the single-building GS model inside of the original GS SIBR viewer. However, visualization of large areas such as the STPLS3D - RA dataset in the SIBR viewer resulted in a reduced frame rate of 30 fps, which further decreased to 16 fps upon transferring the simulation into UE5.

Dataset	Size	Gaussian Count
Building (Bld)	282MB	1,196,163
Residential Area (RA)	6.64GB	28,778,729
Soibelman Bld	1.45GB	6,304,311
Feihong Bld	5.35GB	23,177,147

Table 3: Dataset statistics

To enhance the interactive experience, users can navigate the scenes using a first-person controller and a weapon that fires projectiles. These features are provided by the standard assets included in the Unreal Engine 5. Gaussian colliders and Niagara renderers are utilized for each sense and can be customized as desired. Upon impact, the destruction simulation is applied to Gaussians within a 2-meter radius sphere, which can be adjusted to suit individual preferences. This process is efficient and does not impact performance, as the destruction simulation operates at a constant speed regardless of the number of Gaussians with non-zero velocity.

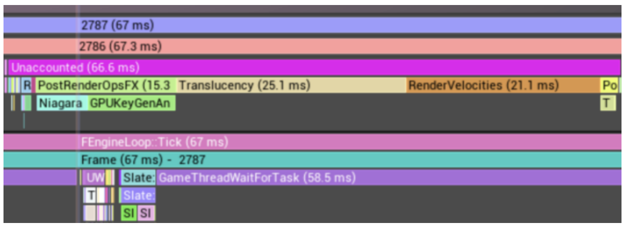

Figure 8. Timings for a random single frame

We utilized Unreal Engine's built-in profiler as shown in Figure 8 to measure our performance metrics. While running the profiler can impact performance, it provides reasonable measures for relative comparisons. Our focus was on the GPU markers, specifically Niagara. This represents the time spent by the Niagara system not accounted for by more granular markers. We also analyzed GpuKeyGenAndSort, which represents the time spent creating keys for Gaussian and performing view-dependent sorting, Translucency which is the time spent performing alpha blending for the Gaussians, and RenderVelocities, which indicates the time spent computing destruction simulation. These metrics were captured during initialization, first frame, and general frame average. Unreal Engine caches compiled Niagara code for performance reasons, but we purged the cache for each level to ensure accurate measurements. As a result, the startup time was increased by roughly 100ms.

In Table 4, we illustrate the timings for the period between pressing the play button and the level becoming playable. During this time, a significant portion is devoted to the initialization of the gameplay framework by the Unreal Engine, as well as the unpacking of the level from disk and the initialization of all the actors within it. Additionally, if the cache for the Niagara system is absent, a re-compilation of the system may take place, which typically takes around 100ms.

Metric	Bld	RA	Soibelman	Feihong
Niagara Initialization	100.3ms	100.1ms	99.9ms	100.6ms
Level Startup	85.7ms	1100ms	336.6ms	647.4ms

Table 4: Initialization and Startup Performance

In Table 5, we present an analysis of the initial frame performance. Typically, performance exhibits significant enhancement subsequent to the initial frames, partially attributable to engine overheads and the garbage collection mechanisms inherent in the Unreal Engine. Furthermore, it is during the first frame that we predominantly observe the most substantial fluctuations in our timing measurements.

Metric	Bld	RA	Soibelman	Feihong
Niagara	0.2ms	56.8ms	1ms	2.9ms
View-based sorting	0.2ms	39.4ms	1.8ms	7.7ms
Alpha blending	2.5ms	59.4ms	5.1ms	17.4ms
Simulation	1.3ms	22.1ms	4.7ms	16.6ms
Frame Time	23.8ms	477ms	336.6ms	647.4ms

Table 5: First frame performance

Table 6 exhibits the average performance metrics garnered from numerous three-minute iterations, wherein participants traversed the scene in first-person perspective, concurrently engaging in projectile launches and orchestrating destruction simulations involving a substantial quantity of Gaussian entities.

Metric	Bld	RA	Soibelman	Feihong
Niagara	0.28ms	5.5ms	1.2ms	4.4ms
View-based sorting	0.25ms	9.9ms	1.8ms	8.1ms
Alpha blending	2.2ms	22.7ms	7.1ms	23.4ms
Simulation	1.2ms	21ms	5.4ms	18.4ms
Frame Time	16.3ms	61.3ms	19.8ms	58.5ms

Table 6: Average frame performance

5. Discussion and Limitations

During our experiments, we utilized a high-end desktop with an Nvidia RTX 3090 graphics card that boasted 24GB of VRAM. Despite this, we encountered an issue during testing where our custom renderer, created using the LumaAI plugin in Unreal Engine, was unable to render one of our test datasets (not included in the experiments) due to GPU memory limitations. This highlighted the importance of Gaussian streaming when it comes to efficiently rendering large environments. By streaming Gaussian splats, we can eliminate the size limitations of the environment and allow for the rendering of expansive scenes. Additionally, these systems can be designed to maintain a consistent number of active Gaussians in the scene and GPU memory, ensuring near-constant frame rates. To take the first step towards streaming, we partitioned our Gaussian point cloud into smaller clouds, each with a limit of 2 million Gaussians. This approach allows us to stream these smaller clouds independently and overcome the 2 million particle limit of Unreal Engine's Niagara system.

Our current collision detection relies on meshes generated through photogrammetric methods. However, even the most advanced methods available today struggle with producing high-fidelity meshes when it comes to thin objects like poles (Figure 9). Since these objects are not present in the collision mesh, it becomes difficult to detect projectile hits. Gaussian point clouds can represent thin objects as highly stretched ellipsoids, but this approach can limit the accuracy of our destruction simulation. Moreover, photogrammetry meshes may not always align with the renderer point cloud, leading to slightly inaccurate hit point detection. One possible solution is to mesh the Gaussians with high-resolution voxel meshing algorithms, but this can impact memory and slow down the physics engine. A better approach would be to test collisions against the point cloud directly. Unfortunately, no off-the-shelf physics engines support this, and custom solutions would be needed. By using the position, covariance matrix, and opacity of a Gaussian, we can determine whether a given point is inside or outside the bounding Gaussian volume. This approach involves discretizing the continuous Gaussian volume based on an opacity threshold, which also controls collision detection accuracy. The advantage of this approach is that we can simplify the workflow and save memory used by collision meshes as part of the physics system by eliminating the need for photogrammetry.

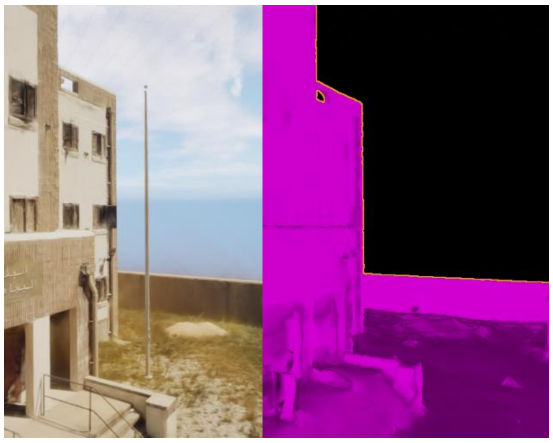
Figure 9. No vertices for thin objects in collision mesh

The Niagara renderer, powered by the LumaAI plugin, is not open source, which makes it difficult to modify its modules. This becomes a problem when we need to adjust the GpuKeyGenAndSort modules responsible for view-dependent sorting of Gaussians, which is crucial for rendering speed. To enable streaming for large-scale Gaussian environments, we divided the Gaussian point cloud into multiple smaller point clouds, each with a limit of two million Gaussians. These small clouds are then instantiated as separate Niagara systems in the scene, with the GpuKeyGenAndSort module running in isolation and localized to each system. Unfortunately, this causes artifacts in the alpha blending phase, resulting in flickering when sorting is localized to the two million Gaussians of that particular instance of the Niagara renderer. As the number of active Niagara systems grows, the flickering becomes more evident and less immersive. Currently, there isn't an off-the-shelf package workaround for this issue, but a possible solution would be to have the GpuKeyGenAndSort module run globally for all active Gaussians. In addition, due to the closed-source nature of the LumaAI plugin, we can only suggest changes that would fix this issue if we use an open-source renderer or create our renderer. Further development on a custom renderer with complete control for Gaussian streaming is needed to achieve high-fidelity, large-scale 3D Gaussian splatting environments with maximum performance.

6. Conclusion

The 3D Gaussian Splatting models demonstrate significant potential for both visualization and simulation, offering promising avenues for rendering detailed urban environments with high fidelity. However, when addressing large-scale areas, the GS model reveals the necessity for further research into optimized data formats and visualization techniques.

The key factor contributing to the impressive performance boost in Gaussian splatting compared to neural radiance field techniques is the visibility-aware sorting method. This method allows for direct updating of the frame buffer with alpha blended values, eliminating the need for per-pixel alpha blending through depth testing. As a result, rendering performance is significantly improved, allowing for real-time rendering at over a hundred frames per second. The benefits of this approach extend beyond rendering, as it also speeds up training times by enabling a fast backpropagation step through reverse sorted traversal of the Gaussians. Additionally, increasing the number of Gaussians that receive gradients has little to no impact on performance, making it an efficient choice for training.

In order to facilitate the training of expansive scenes, we have implemented a tiling methodology that partitions the scenes into smaller sections and trains them using the corresponding subset of images. The outcome is then merged into a single point cloud or divided into multiple manageable point clouds that can be streamed seamlessly.

The LumaAI plugin boasts a competitive Gaussian splatting renderer for Unreal Engine. However, it has a two million particle count restriction of Niagara that limits its potential. To address this, we introduced a partitioning technique that divides Gaussian point clouds into smaller point clouds. These can be instantiated as individual Niagara systems and streamed on demand, thus optimizing performance and resource usage. Taking it a step further, we integrated simulation capabilities into the renderer provided by the LumaAI plugin. To communicate simulation data with the Niagara particle system, we introduced Gaussian collider actors, along with a scene-specific mapping of renderers to colliders. This minimizes the number of GPU calls to push data into the VRAM. With these techniques, we achieved a first-person interactive Gaussian splatting environment at playable frame rates with high fidelity. Our next steps involve developing more efficient streaming mechanisms, integrating with Unreal Engine's chaos destruction engine, and implementing Gaussian-based collisions without the need for photogrammetric meshes. These advancements will help increase performance and simplify the workflow for creating large-scale 3D terrain reconstruction using 3D Gaussian splatting for visualization and simulation.

Acknowledgements

The authors would like to thank our primary sponsors of this research: US Army Simulation and Training Technology Center (STTC). They would also like to acknowledge the assistance provided by Army Futures Command (AFC) and Synthetic Training Environment (STE). This work is supported by University Affiliated Research Center (UARC) award W911NF-14-D-0005. Statements and opinions expressed and content included do not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

References

Kerbl, B., Kopanas, G., Leimkühler, T., & Drettakis, G. (2023). 3D Gaussian Splatting for Real-Time Radiance Field Rendering. ACM Transactions on Graphics, 42(4).

Chen, M., Hu, Q., Yu, Z., Thomas, H., Feng, A., Hou, Y., McCullough, K., Ren, F., & Soibelman, L. (2022). STPLS3D: A large-scale synthetic and real aerial photogrammetry 3d point cloud dataset. 33rd British Machine Vision Conference, (BMVC), London, UK.