NVGS: Neural Visibility for Occlusion Culling in 3D Gaussian Splatting

Brent Zoomers[1] Florian Hahlbohm[2] Joni Vanherck[1] Lode Jorissen[1]
Marcus Magnor[2] Nick Michiels[1]
[1]Digital Future Lab, Hasselt University, Belgium [2]Computer Graphics Lab, TU Braunschweig, Germany
https://brent-zoomers.github.io/nvgs/

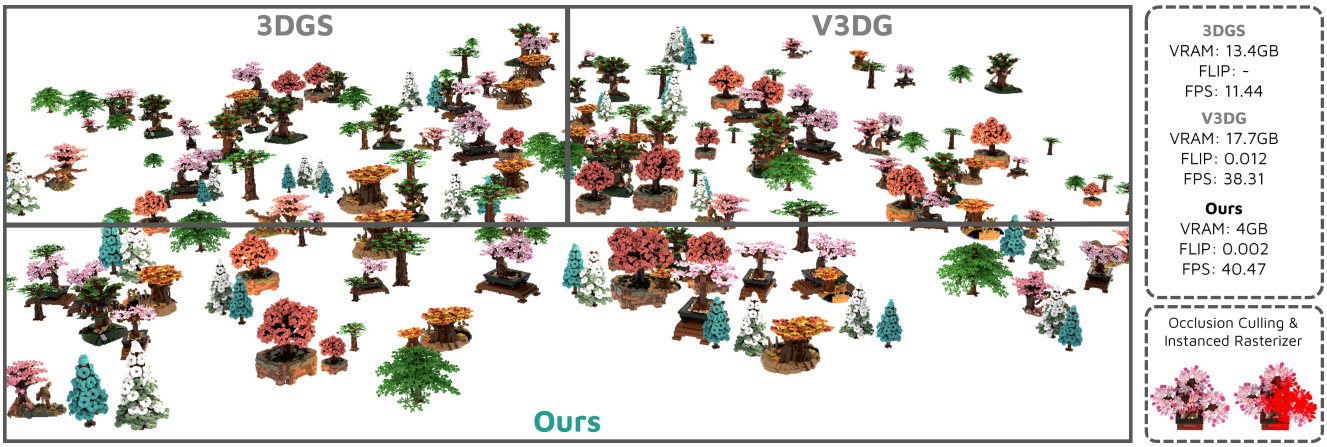


Figure 1. Shows a scene composed of multiple, separately trained 3DGS assets. *Top left* shows the gsplat implementation of 3DGS[28], *Top right* V3DG [27] and *Bottom* Ours. Our approach uses significantly less VRAM, consistently achieves higher image quality, and increases FPS. We achieve this by combining a novel instanced rasterizer and our neural visibility MLP, which enables occlusion culling.

Abstract

3D Gaussian Splatting can exploit frustum culling and level-of-detail strategies to accelerate rendering of scenes containing a large number of primitives. However, the semi-transparent nature of Gaussians prevents the application of another highly effective technique: occlusion culling. We address this limitation by proposing a novel method to learn the viewpoint-dependent visibility function of all Gaussians in a trained model using a small, shared MLP across instances of an asset in a scene. By querying it for Gaussians within the viewing frustum prior to rasterization, our method can discard occluded primitives during rendering. Leveraging Tensor Cores for efficient computation, we integrate these neural queries directly into a novel instanced software rasterizer. Our approach outperforms the current state of the art for composed scenes in terms of VRAM usage and image quality, utilizing a combination of our instanced rasterizer and occlusion culling MLP, and exhibits complementary properties to existing LoD techniques.

1. Introduction

3D Gaussian Splatting [5] (3DGS) has proven itself to be a valuable tool for 3D reconstruction in recent years, due to its ability to represent scenes with great accuracy while also facilitating both fast training and rendering times. As 3DGS transitions from research to actual applications, the focus on efficiently rendering scenes, which are increasing in both scale and complexity, becomes more important. Previous methods [6, 9, 19, 21, 27] have followed this line of thinking and introduced the traditional Level of Detail (LoD) approach to the context of 3DGS. This enables larger scenes to be trained and rendered by utilizing hierarchical techniques to select the currently relevant parts of a scene. Other widely used approaches in graphics, such as occlusion culling, however, are not a natural fit due to the semi-transparent nature of 3D Gaussians. In graphics, these techniques typically rely on the use of opaque triangles, for which determining visibility is straightforward.

Our results show that standard volume rendering implicitly encodes a soft form of occlusion: once transmittance saturates, subsequent splats can be discarded without any significant loss in color. This allows us to determine Gaus-

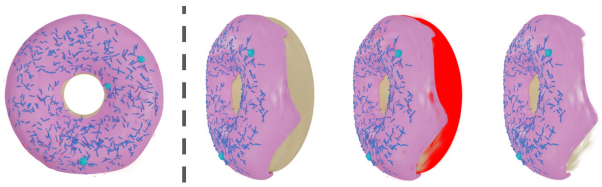


Figure 2. *Left* shows the donut asset from the front. *Right* we show the donut as if rendered from the front from a different viewpoint. We then show the Gaussians that our approach culls in red, along with what we would optimally render.

sians that are not visible to the current viewpoint, which, similar to backface culling for meshes, are located mainly on the backside of the object, as shown in Fig. 2. Another nice property of this formulation is that it scales with distance (Fig. 3). As more primitives are mapped to the same set of pixels, more Gaussians can be discarded, even those lying on the front of the object. Capturing this information, however, requires a large amount of information to be stored. In this work, we propose an efficient pipeline that captures and utilizes visibility information during rendering to discard Gaussians, thereby avoiding the need for expensive preprocessing in the rasterization pipeline. We also propose a novel instanced rasterizer specifically tailored for composed scenes consisting of multiple 3DGS assets. Gaussians are only instantiated after passing through multiple culling phases, which drastically impacts both VRAM usage and rendering speed. Baking the visibility data into a lightweight MLP that is fully integrated into this rasterizer results in a highly optimized pipeline for composed scenes. Similar to their complementary use in graphics, our occlusion culling approach shows complementary properties with LoD, opening up avenues for further optimization of large-scale scenes. In summary, our contributions are:

- We propose a novel approach for occlusion culling in 3DGS rendering that is based on compressing the view-dependent visibility functions of all Gaussians into a small MLP.
- We provide an efficient, virtualized, and instanced rendering pipeline for 3DGS capable of processing scenes over 100 million Gaussians at real-time frame rates.
- We demonstrate that neural network queries can be integrated into the renderer to minimize excessive VRAM usage and global memory traffic, thereby improving performance.

In combination, these contributions outperform the current state of the art for composed scenes in terms of VRAM usage, image quality, and frame rate for short, medium, and, in some cases, long distances.

2. Related Work

2.1. 3D Gaussian Splatting

3DGS [5] has gained immense popularity within the community ever since its introduction. Despite its success, 3DGS still had some limitations, some of which have been addressed in follow-up work, such as its memory footprint and popping artifacts. To reduce the memory footprint, several approaches have been proposed that compress the final trained scene [2, 13, 15], utilizing codebooks or exploiting the redundant nature within the Gaussian parameters. Other works, such as StopThePop [18], have focused on solving the popping artefacts resulting from global sorting. The densification step used in 3DGS to promote exploration also inspired several follow-up works [7, 20]. These works propose alternative strategies for splitting, cloning, or simply moving Gaussians, enabling higher-quality scene reconstructions. In our work, we focus on a different issue related to compression: redundancy at render time. By baking the visibility of Gaussians into a lightweight MLP, we can efficiently avoid expensive preprocessing operations on Gaussians that will not be used during rendering.

2.2. Pruning Gaussians

Within 3DGS, a large degree of redundancy exists, as evidenced by the extensive work on pruning and compression. One common trend among these methods is their focus on redundancy at a global level, meaning that Gaussians that do not contribute to the scene will be pruned. Zhang *et al*. [30] learns this masking by using the Gimbal-Sigmoid method to achieve an approximate binary mask, which can then be used to prune Gaussians. Papantonakis *et al*. [17] propose pruning based on spatial redundancy, where areas with overlapping, low-contributing Gaussians show potential

Figure 3. *Left*: Avatar rendered at a near and far distance. *Right*: We show the original render, then the Gaussians that are not used at the far distance marked in red, and on the right the Gaussians that get used at the far distance.

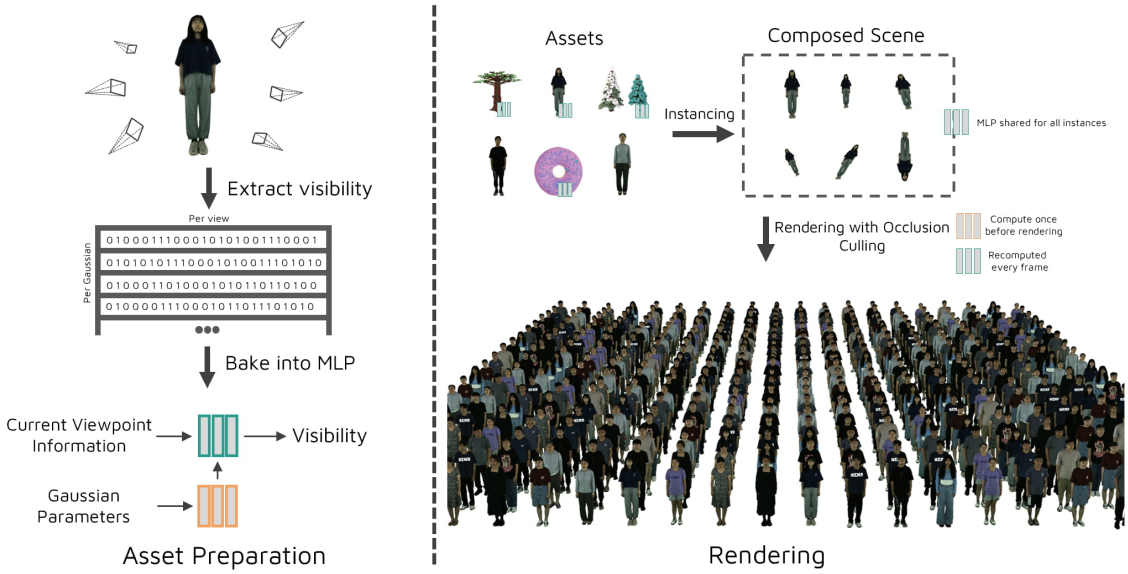


Figure 4. We extract per-Gaussian visibility by rendering to a set of training views and baking the visibility information into a lightweight MLP. Per-Gaussian parameters are encoded using a secondary MLP and precomputed once prior to rendering. At render time, our instanced rasterizer renders composed scenes containing both neural-visibility and standard 3DGS assets, leveraging neural visibility whenever available to improve efficiency by culling zero-contribution Gaussians.

for pruning. EAGLES [3] performs pruning during training, considering the contribution of Gaussians to a pixel's final color as a metric of importance.

Our method stands out as it does not perform pruning as a global step during or after training. We instead consider redundancy at render-time to reduce the workload per frame. This dramatically reduces VRAM usage and increases rendering speed by avoiding expensive preprocessing on many Gaussians that have no contribution to the render. Our approach also scales better when the amount of compute per Gaussian increases, facilitating more expensive per-Gaussian computations in future work, such as different, more expressive lighting models (see Sec. 4).

2.3. Rendering of Large Scenes

As 3DGS extends to larger and more complex scenes, the community has looked back to well-established graphics techniques for inspiration. One such technique is Level of Detail (LoD). LoD has historically played a crucial role in graphics, enabling the drastic reduction of vertices needed to be rendered while having no or minimal impact on quality. Recent work has brought LoD to the context of 3DGS, with most approaches focusing on both training and rendering of large scenes. H3DG [6] employs a divide-and-conquer strategy, where the scene is divided into distinct chunks. Each chunk is then trained separately, after which the full scene can be rendered by efficiently consolidating the different chunks into a single hierarchy. LODGE [9] builds different LoD levels based on the distance of Gaussians to the camera. For far-away Gaussians, they use the 3D smoothing filter from MipSlatting [29] and prune Gaussians using their contribution as done by several other works[3, 16, 16, 31]. Similar to other LoD approaches, they then use a chunk-based rendering approach to reduce required memory. Octree-GS [19] is based on Scaffold-GS [12] and uses the concept of neural anchors, which are enhanced for LoD by employing different levels of voxel grids. The Gaussians are then spawned in using the appropriate levels to achieve the desired LoD. This, combined with a coarse-to-fine training strategy, enables the anchors to more accurately represent different levels of detail. FLoD [21] trains different levels of 3D Gaussian representations where each level independently learns how to represent the scene. A level can be chosen depending on the hardware constraints, or different levels can be combined to achieve proper LoD, where objects further away will use a level that contains less detailed Gaussians. Closest to our approach is V3DG [27], which also focuses specifically on scenes composed of individual assets. Their approach creates different levels of clusters based on spatial proximity. By downsampling each cluster and optimizing it to match the original, they create a hierarchy that allows for different LoD levels. Their approach, however, requires a large amount of VRAM and doubles storage requirements due to needing to store the different clusters. Recently, Liu *et al*. [11] used occlusion-aware scene partitioning to divide a large scene into chunks. This differs from previous works, where chunks typically follow a grid-like structure, allowing them to select Gaussians that will be used more efficiently.

Each chunk is then trained individually, after which they are consolidated again in a similar fashion to the aforementioned LoD approaches. Our approach differs from previous methods as we do not rely on LoD. Instead, we utilize a different method commonly used in graphics, which is occlusion culling. Compared to OccluGaussian[11], our approach is more fine-grained, as we focus on occlusion at a Gaussian level instead of a scene level. The combination of our instanced rasterizer and visibility MLP significantly reduces the number of Gaussians, outperforming existing LoD approaches on VRAM usage and quality metrics. We want to emphasize that LoD and occlusion culling do not exclude use of the other.

3. Method

Our approach consists of three main components. First, we extract the visibility data from a pre-trained 3D Gaussian Splatting (3DGS) asset (Sec. 3.1). Next, the visibility data is distilled into a lightweight Multilayer Perceptron (MLP) representation, enabling efficient inference during rendering (Sec. 3.2). Finally, we introduce a novel instanced rasterizer specifically designed for 3DGS, capable of rendering scenes consisting of different assets with a total number of over 100 million Gaussians efficiently by utilizing instancing and occlusion culling (Sec. 3.3). We detail each of these components in the following subsections.

3.1. Extracting Visibility

Given a trained 3DGS asset, we first extract visibility information suitable for training an MLP. The extracted data must support unbiased training, ensuring that no periodic patterns or systematic biases are introduced. This can result from periodicity in direction or distance sampling. Additionally, it should remain robust to common 3DGS artifacts such as popping and aliasing, as we want our approach to be usable for any 3DGS-inspired approach. In most 3DGS-inspired approaches, pruning and densification are halted long before training ends. This results in the resulting asset containing Gaussians that are effectively useless during rendering due to having too low opacity values. Our first step is thus to remove these Gaussians from the asset to simplify training the MLP and reduce VRAM usage. We also center the object to simplify subsequent steps and have a uniform process for all assets. Using the means of the centered assets, we compute a per-asset minimum and maximum distance between which the camera positions will be sampled. The minimum distance ensures that no camera is placed inside the object, while the maximum distance serves as a far bound, from which on the same quality will be displayed. These distances are calculated based on the projected screen size of the diagonal, where the near distance uses 90%, and the far distance 5%. To convert the projected screen size of an object into a distance value for the cameras, we use the following formula:

$$d = \frac{r}{\tan(\theta/2) \cdot p} \tag{1}$$

where d represents the distance, r is half the length of the diagonal of the bounding box of the object, p is the percentage of the image we want the diagonal to cover, and θ is the field of view used in the camera to create the training views. We then uniformly sample directions using the Fibonacci sphere sampling [4] method. The camera position is computed by uniformly sampling distance values between the minimum and maximum distances and multiplying them by the direction. To further increase robustness, specifically against projective inconsistencies, we add a random offset to the asset so that it does not appear at the center of each image. This offset is scaled based on the distance of the camera to the asset to ensure it still lies within the frustum. Popping in 3DGS can result in Gaussians being classified as invisible when they would be visible with a slight position change. We avoid training on these artifacts by also sampling a small set of auxiliary views on the intersection of a cone rotated towards the camera and the sphere at the same distance as the camera. Although this increases the time required for preprocessing, we demonstrate in Sec. 4 that it has a positive impact on quality metrics. Using this setup, a Gaussian is considered visible from a certain camera if it has a nonzero contribution to the main camera or any of the auxiliary views. The contribution of a Gaussian G at pixel p is defined as in previous work $C_{G,p} = \alpha_p \cdot T$ where α_p denotes the Gaussian's opacity at pixel p, and T represents transmittance up to that point [3, 5, 16, 31].

3.2. Training the Visibility MLP

The efficacy of our approach heavily depends on the trade-off between the time spent querying the MLP and the time it saves during rendering. The precomputed data in itself is too large to be used efficiently during rendering; hence, we bake it into a lightweight MLP implemented using tiny-cuda-nn (TCNN) [14]. We train this MLP once per asset, after which it can be used for all instances of that asset, regardless of instance transformations or different camera properties, such as resolution or Field of View. The normalized Gaussian's mean, normalized direction, and distance to the camera, as well as the forward vector of the camera and the Gaussian parameters, are passed to the MLP to predict visibility. Alongside the main visibility MLP, we train a secondary MLP to learn a six-dimensional feature vector for each Gaussian based on its parameters. Given how frequency encodings have been used in related work to increase the expressivity of the network for similar functions [22], we briefly justify our decision to omit them. While frequency encodings allow for capturing finer details, they also increase the number of inputs to the network and increase computation time due to

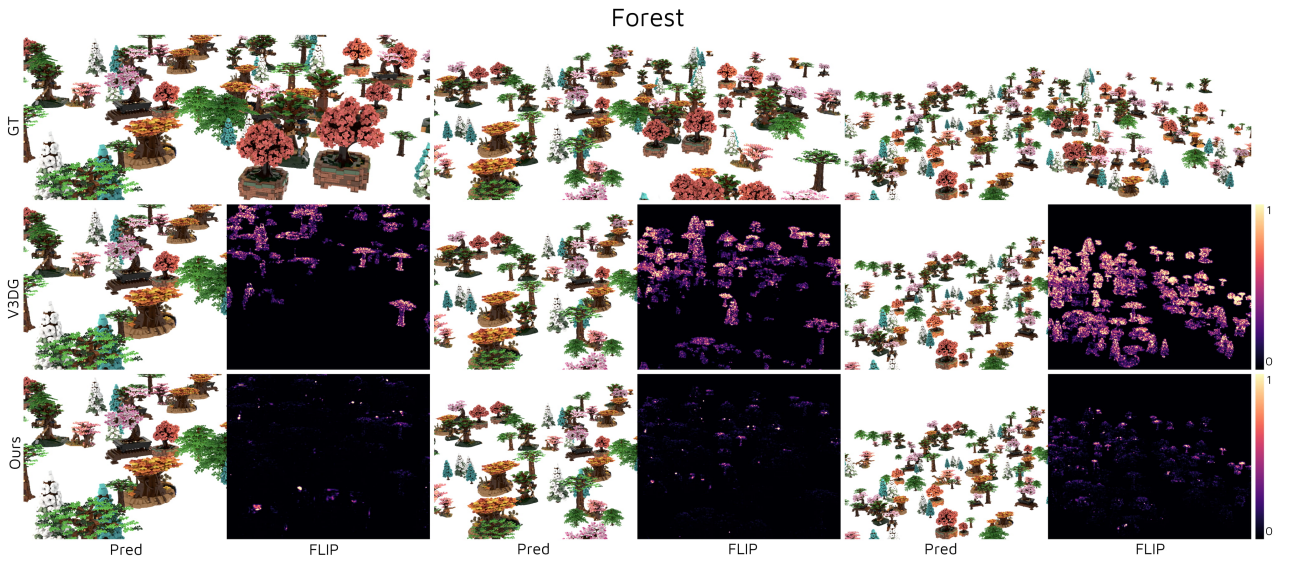


Figure 5. Shows GT render using gsplat, V3DG, and our method for close, medium, and far distances. For V3DG and ours, we show the prediction on the left and FLIP on the right. We scaled FLIP by a factor of five to facilitate better visual comparison.

the use of slow trigonometric functions. Using only five frequency bands on six inputs(mean + direction) increases the number of inputs from six to 60. Additionally, our approach attempts to utilize Tensor Cores optimally by aligning the input dimension with a multiple of 16. By omitting frequency encodings and using a learned feature vector for the Gaussian parameters, our approach uses exactly 16 inputs, making it optimal for the use of Tensor Cores.

3.3. Occlusion-Aware Instanced Rendering

We introduce a novel instanced rasterizer for 3DGS that enables the rendering of large composed scenes consisting of upwards of 100 million Gaussians. Each unique asset is stored once, together with a list of instance transforms. A Gaussian is then instantiated only if it survives several culling steps, drastically reducing VRAM usage. Before rendering, we precompute the secondary MLP once to extract the per-Gaussian feature vectors. Per frame, we then perform frustum culling based on the means of the Gaussian and decide whether to query the MLP based on the per-asset minimum distance. The input to the MLP is calculated per instance, hence we must convert the inputs from global space back into the local space that was used during training. For the viewing direction and forward vector of the camera, we simply apply the global-to-local rotation. The means of the Gaussians have to be translated and scaled back to be in local space. For distance, we need to undo two changes. First, we account for a potential difference in FoV between the cameras used for rendering and training. Second, we undo scaling that was applied to the object. The distance in local space is then calculated as follows:

$$d_t = d_r \cdot \frac{f_t}{f_r} \cdot \frac{1}{s} \tag{2}$$

where the subscripts t and r refer to training and rendering. d_t and d_r represent distance, f_t and f_r represent the focal length, and s the applied scaling factor to the asset. Finally, we map this distance into the $[-1, 1]$ range using standard min-max normalization, based on the per-asset minimum and maximum distances. Note how, thus far, all operations can be performed on the non-instanced Gaussians. These inputs, together with the per-Gaussian feature vector, are passed to the MLP, which predicts per-Gaussian visibility. Only after a Gaussian passes the previous culling steps is a per-tile instance created. From here, we use an optimized version of the standard 3DGS pipeline to render these instances. As shown by Lee *et al*. [10], the Gaussian parameters account for the largest portion of VRAM usage in 3DGS. Hence, by avoiding the instantiation of Gaussians unless they contribute to the view, we minimize this overhead. Furthermore, we also increase rendering speed as we do not have to perform preprocessing or sorting on discarded Gaussians. Our approach integrates both instanced rendering and occlusion culling to facilitate a highly optimized pipeline tailored to the rendering of composed scenes.

4. Experiments

To evaluate our approach, we first compare against the state of the art for composed scenes, V3DG [27]. We also compare against gsplat, and gsplat with the radius clipping parameter enabled. Finally, we include ablations to isolate key

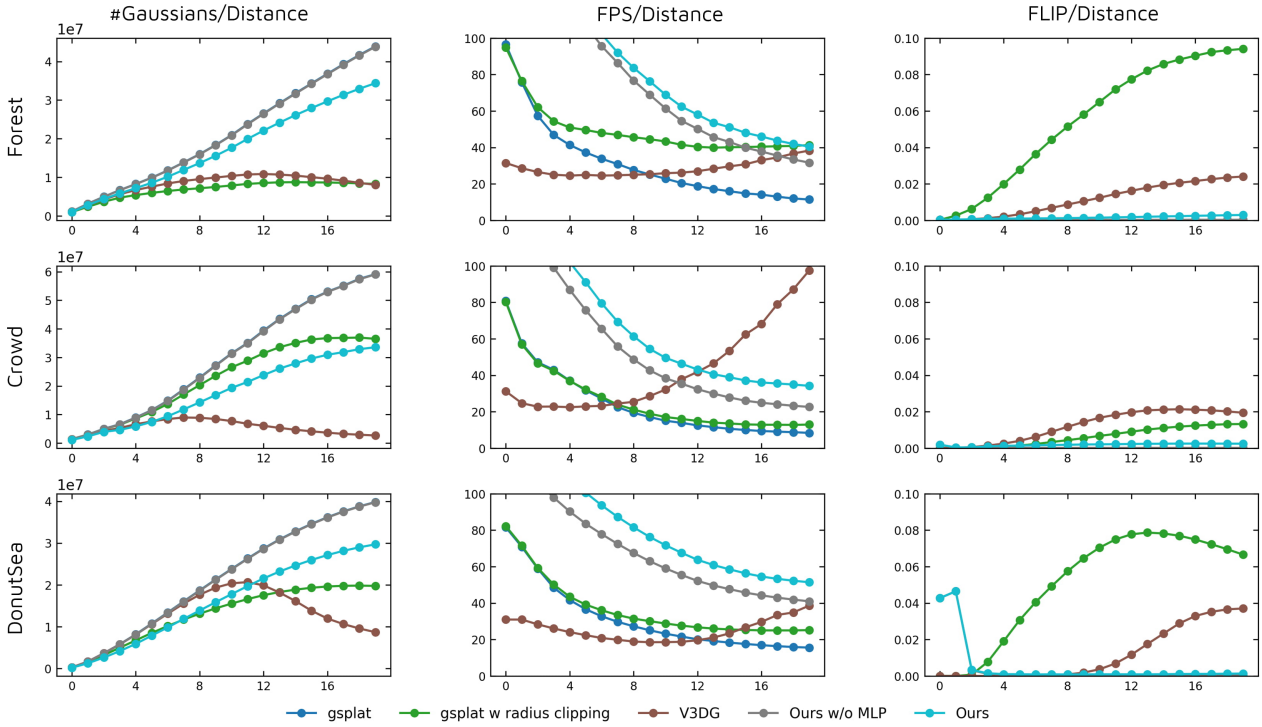


Figure 6. Shows how metrics evolve over distance for three scenes from V3DG [27]. We employ the same camera setups as in V3DG and demonstrate that, for near-to medium-range applications, our approach outperforms all other methods. For far distances, our approach remains competitive while using significantly less VRAM. We want to point out that the increase in FLIP for the DONUTSEA scene can be attributed to small numerical rounding errors when close to the near plane. This occurs for both our methods, as we do not use the gsplat rasterizer. For the number of Gaussians, our method without MLP and gsplat yield the same result.

components of our approach. Our novel instanced rasterizer and occlusion MLP allow our approach to compete with existing LoD methods. However, we treat LoD as complementary rather than competing. This comparison establishes the performance landscape for composed-scene pipelines and highlights the complementary aspects of both LOD and occlusion culling. All our experiments are conducted on a single NVIDIA RTX 3090 Ti with 24 GB of VRAM. For comparative experiments with gsplat [28] and V3DG[27], we used a version of the scenes downscaled by a factor of two, ensuring that all methods could be run without exceeding VRAM. We also demonstrate that our approach can run full-scale scenes without exceeding the VRAM limit. In Sec. 6 and on the project page, we provide additional videos for the full scenes and comparisons with other methods.

4.1. Implementation Details

Sampling points on the sphere to capture the contribution values is achieved using Fibonacci sphere sampling [4], where each point is then scaled uniformly between the minimum and maximum distance determined for each asset, as described in Sec. 3.1. The random offset is then applied to the center point of the object, with no offset being applied when the object is at its minimum distance. At other distances, the offset is scaled by linearly interpolating between zero and the ratio between the minimum and maximum scale of the object. The embedding for each Gaussian is produced by a small MLP consisting of two layers, each with 32 neurons, and outputs a 6D feature vector. We optimize using Adam [8], with an initial learning rate of 2e-3 that decays to 2e-4 via a scheduler. The scheduler applies a cosine warm-up for the first 20% of iterations, followed by an exponential decay until completion. Batch size is 2^{19}, with samples drawn across views and Gaussians.

4.2. Datasets and Metrics

Following V3DG, we evaluate the performance of our approach using the number of Gaussians, frames per second (FPS), and FLIP error [1]. Furthermore, we utilize the standard image quality metrics PSNR and SSIM [25], and VRAM usage for comparisons. We present results for three scenes, for which assets were sampled from three different datasets, including eight synthetic LEGO trees from the RTMV dataset [23], 16 real human avatars from the MVHu-

Table 1. **Quantitative comparisons** on three large composed scenes (∼60M Gaussians each, rendered at 1080p). We omit both baseline gsplat and ours without the MLP from image metrics, as both reproduce the ground-truth render. gsplat† applies radius-based culling. Ours†† is on the full-scale scenes with Ours w/o MLP used as ground-truth as gsplat runs out of VRAM. Best results are highlighted in **green**.

Method	FOREST				CROWD				DONUTSEA			
	PSNR↑	SSIM↑	FLIP↓	VRAM↓	PSNR↑	SSIM↑	FLIP↓	VRAM↓	PSNR↑	SSIM↑	FLIP↓	VRAM↓
gsplat	–	–	–	13.4GB	–	–	–	16.9GB	–	–	–	11.5GB
gsplat†	29.5	0.948	0.056	9.7GB	48.0	0.996	0.006	13.6GB	35.6	0.916	0.053	8.3GB
V3DG	42.3	0.993	0.012	17.7GB	43.2	0.991	0.013	20.4GB	58.2	0.983	0.012	13.9GB
Ours w/o MLP	–	–	–	5.1GB	–	–	–	6.6GB	–	–	–	4.2GB
Ours	52.7	1.000	0.002	4.0GB	48.6	1.000	0.002	4.5GB	58.3	1.000	0.006	3.1GB
Ours††	51.6	1.000	0.003	7.24GB	49.2	0.999	0.003	8.13GB	64.2	1.000	0.001	6.05GB

manNet dataset [26], and a donut asset [24]. The authors of V3DG provided several scene layouts constructed from these assets, which were trained with zero degrees for spherical harmonics. For all experiments, we used the preprocessed assets, where Gaussians with an opacity lower than $\frac{1}{255}$ have been pruned and the assets are recentered. We use their original layouts alongside downscaled versions to ensure they fit within VRAM for comparisons across all methods. To create the downscaled versions of each layout, we halved the number of instances for each asset in the scene.

4.3. Results

We present a qualitative visual comparison in Fig. 5 comparing our approach to baseline 3DGS and V3DG [27]. Our method reaches near-perfect reconstruction while significantly reducing the number of Gaussians. In Fig. 6, we show how the number of Gaussians, the FPS, and the FLIP metric change over distance. We want to point out that for DONUTSEA and FOREST, the majority of Gaussians are located on the glazing of the donut(∼75%), and the crown of the trees(∼66%). With most views taken from above, the number of Gaussians we can prune using occlusion is lower than for scenes with more balanced assets, such as in CROWD, as demonstrated by Fig. 6. Our approach discards the majority of these occluded Gaussians. Due to V3DG halving the number of Gaussians per level, they can use fewer Gaussians with the overhead of having to calculate which clusters to use. For distances of low to medium range, we achieve higher rendering speeds than both V3DG and gsplat with radius clipping enabled. For further distances, V3DG catches up, as the difference in the number of Gaussians becomes more significant. The graph clearly illustrates the complementary nature between LoD and occlusion culling as V3DG starts to increase in speed, as our approach slows down over distance. Regardless of distance, our approach outperforms gsplat with radius clipping and V3DG in terms of quality and VRAM usage. For DONUTSEA, the initial peak in FLIP can be explained by numerical differences between our instanced rasterizer and gsplat at the near plane, as this is the only scene where the camera goes inside objects. The inclusion of the MLP further increases FPS over our instanced rasterizer by an average of ∼10 FPS, showing the significant impact of discarding unused Gaussians. In Fig. 8, we show that the increase in rendering speed when using the MLP becomes larger as the degrees of spherical harmonics used are increased. As our approach works on the original asset, this allows creators to use more complex per-Gaussian calculations while still benefiting from our occlusion culling method. On average across all assets, training the MLP takes around 3.3 minutes. Extracting the contribution data from the assets takes roughly 2 minutes, while training the model requires 1.3 minutes for 30.000 iterations. The result is then an MLP checkpoint that uses 18 kB, which, when compared to the size of the PLY, occupies less than 0.1% of the total storage. Table 1 shows the average metrics that are averaged over all distances and the peak VRAM usage. For base gsplat and our approach without MLP (solely the instanced rasterizer), we do not show image quality metrics as they represent the ground truth (disregarding the small numerical difference at the near plane). The inclusion of the instanced rasterizer significantly decreases VRAM usage and increases FPS, as expected. This directly follows from instantiating Gaussians only if they are within the frustum. We use ∼4× less VRAM compared to V3DG, making our approach more scalable to larger scenes. Using the MLP, the process is further optimized by also discarding splats inside the frustum that are occluded by others. Besides the increase in FPS, this also reduces VRAM usage further by 25%. Our approach reaches a near-perfect score for all scenes in image

Table 2. **Build-time comparison (minutes)** for V3DG and our method, averaged across all assets. For our approach, we report the combined time for visibility extraction and model training. The RTMV–Trees subset contains 8 trees, MVHumanNet contains 16 humans, and Donut contains a single object, with average Gaussian counts of 287,965; 336,554; and 44,106; respectively.

Method	Trees (min)	MVHumanNet (min)	Donut (min)
V3DG	7.46	9.00	0.98
Ours	3.95	4.19	2.00

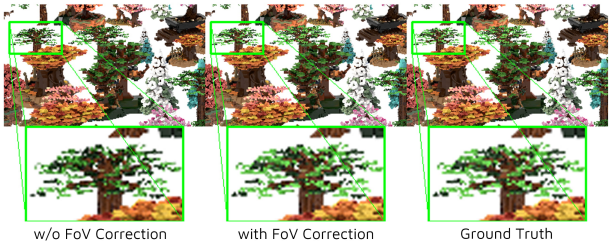


Figure 7. Shows the effect of FoV correction. Training data uses a FoV of 60°, while the rendering camera uses approximately ∼20°.

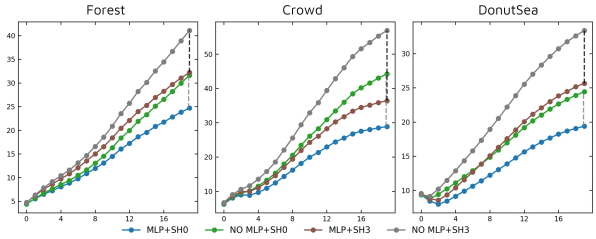


Figure 8. Shows render time in milliseconds at 20 relative distances. SH represents the number of degrees used for spherical harmonics. Increasing it from 0 to 3 makes adding our MLP more beneficial, as shown by the larger vertical distance between the graphs.

metrics while using significantly fewer Gaussians, showing the effectiveness of our occlusion culling MLP. Table 2 shows the build times for all datasets compared to V3DG. For our method, approximately 25% of the time is spent on visibility extraction, and about 75% on training the MLP.

4.4. Ablations

In Tab. 3, we show the impact of different additions to our approach. A first small improvement results from replacing Longitude-Latitude sampling with Fibonacci sphere sampling. Including the random offset and adding auxiliary views results in increased VRAM usage and a decrease in FPS. Both additions serve as a robustness measure against small artifacts, making it so more Gaussians are rightfully predicted as visible, as is reflected in the FLIP metric. In the case of auxiliary views, it helps against training on popping artifacts, while the random offset helps with projection artefacts. We argue that these overpredictions are preferred over worse visual results, even if they reduce FPS and increase VRAM usage. Applying the FoV correction results in the largest change to all metrics. Given that training was done using a camera with a FoV of 60°, and the layouts in V3DG all use lower FoVs, the perceived distance from camera to Gaussian was consistently under-estimated. This leads to the model predicting visibility as if the object were viewed from a greater distance. Lastly, we optionally apply radius clipping to accelerate the rendering of distant distances. In gsplat, radius clipping works by using the largest radius of the axis-aligned bounding box to decide whether a Gaussian should be pruned. We instead opt to use the determinant of the 2D covariance matrix, which relates to the 2D area at one σ. Including radius clipping increases the FLIP error, but allows for faster rendering at far distances.

Table 3. **Ablation study**. FPS is measured at the max distance to isolate the impact for far away viewpoints. FLIP is averaged over all distances, VRAM shows peak usage. We optionally use radius clipping to increase rendering speed, albeit at the cost of quality.

Method	FPS$^{\uparrow}$	FLIP$^{\downarrow}$	VRAM$^{\downarrow}$
LongLat sampling	61.04	0.0170	2.64GB
Fibonacci Sampling	61.61	0.0168	2.63GB
+ Random Offset	59.75	0.0142	2.72GB
+ Auxiliary Views	53.23	0.0113	2.98GB
Full (+ FoV Correction)	42.02	**0.0031**	3.88GB
+ Radius Clipping	50.59	0.0067	3.79GB

4.5. Limitations and Future Work

While our method significantly improves VRAM efficiency for rendering 3DGS scenes, it is not without its limitations. The performance of the MLP depends on the number of Gaussians and the complexity of their visibility functions. Objects with highly high-frequency visibility patterns may require additional overhead to ensure robustness. We have demonstrated the efficacy of our visibility MLP on individual assets, but it can be extended to full scenes using camera-sampling strategies that account for scene geometry. For large mesh-based scenes, occlusion culling and LoD are applied orthogonally to maintain rendering efficiency. Combining these approaches in the context of 3DGS could broaden the range of renderable scenes. We leave these points as future work for now.

5. Conclusion

In this work, we presented the first object-level occlusion culling for 3D Gaussian Splatting. By encoding visibility in a small MLP, we enable efficient rendering by discarding Gaussians, avoiding the need for expensive preprocessing for a significant number of Gaussians, which directly addresses a key bottleneck in prior approaches. We also introduced a novel instanced 3DGS rasterizer that integrates MLP evaluation into the rendering pipeline to enhance performance while reducing VRAM requirements. Combined, these contributions enable the efficient, real-time rendering of composed scenes with up to 100 million Gaussians. By introducing a novel pipeline specifically designed for composed scenes, we aim to lower the barrier to assembling and rendering large-scale composed scenes for industry applications, such as gaming or film production.

Acknowledgements

This research was partly funded by the FWO fellowship grants (1SHDZ24N, 1S80926N), the Special Research Fund (BOF) of Hasselt University (R-14360, R-14436), the NORM.AI SBO project, the DFG project "Real-Action VR" (ID 523421583), and the Flanders Make's XRTWin SBO project (R-12528). This work was made possible with the support of MAXVR-INFRA, a scalable and flexible infrastructure that facilitates the transition to digital-physical work environments.

References

[1] Pontus Andersson, Jim Nilsson, Tomas Akenine-Möller, Magnus Oskarsson, Kalle Åström, and Mark D. Fairchild. Flip: A difference evaluator for alternating images. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 3(2):15:1–15:23, 2020. 6

[2] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps, 2023. 2

[3] Sharath Girish, Kamal Gupta, and Abhinav Shrivastava. Eagles: Efficient accelerated 3d gaussians with lightweight encodings. In *Computer Vision – ECCV 2024*, pages 54–71, Cham, 2025. Springer Nature Switzerland. 3, 4

[4] Álvaro González. Measurement of areas on a sphere using fibonacci and latitude–longitude lattices. *Mathematical geosciences*, 42:49–64, 2010. 4, 6

[5] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), 2023. 1, 2, 4

[6] Bernhard Kerbl, Andreas Meuleman, Georgios Kopanas, Michael Wimmer, Alexandre Lanvin, and George Drettakis. A hierarchical 3d gaussian representation for real-time rendering of very large datasets. *ACM Transactions on Graphics*, 43(4), 2024. 1, 3

[7] Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Weiwei Sun, Yang-Che Tseng, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. 3d gaussian splatting as markov chain monte carlo. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. Spotlight Presentation. 2

[8] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2014. 6

[9] Jonas Kulhanek, Marie-Julie Rakotosaona, Fabian Manhardt, Christina Tsalicoglou, Michael Niemeyer, Torsten Sattler, Songyou Peng, and Federico Tombari. LODGE: Level-of-detail large-scale Gaussian splatting with efficient rendering. In *Proceedings of the 39th International Conference on Neural Information Processing Systems*, 2025. 1, 3

[10] Jiwon Lee, Yunjae Lee, Youngeun Kwon, and Minsoo Rhu. Characterization and analysis of the 3d gaussian splatting rendering pipeline. *IEEE Computer Architecture Letters*, 24 (1):13–16, 2025. 5

[11] Shiyong Liu, Xiao Tang, Zhihao Li, Yingfan He, Chongjie Ye, Jianzhuang Liu, Binxiao Huang, Shunbo Zhou, and Xiaofei Wu. Occlugaussian: Occlusion-aware gaussian splatting for large scene reconstruction and rendering. 2025. 3, 4

[12] Tao Lu, Mulin Yu, Linning Xu, Yuanbo Xiangli, Limin Wang, Dahua Lin, and Bo Dai. Scaffold-gs: Structured 3d gaussians for view-adaptive rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20654–20664, 2024. 3

[13] Wieland Morgenstern, Florian Barthel, Anna Hilsmann, and Peter Eisert. Compact 3d scene representation via self-organizing gaussian grids. In *Computer Vision – ECCV 2024*, pages 18–34, Cham, 2025. Springer Nature Switzerland. 2

[14] Thomas Müller. tiny-cuda-nn, 2021. 4

[15] KL Navaneet, Kossar Pourahmadi Meibodi, Soroush Abbasi Koohpayegani, and Hamed Pirsiavash. Compgs: Smaller and faster gaussian splatting with vector quantization. *ECCV*, 2024. 2

[16] Michael Niemeyer, Fabian Manhardt, Marie-Julie Rakotosaona, Michael Oechsle, Daniel Duckworth, Rama Gosula, Keisuke Tateno, John Bates, Dominik Kaeser, and Federico Tombari. Radsplat: Radiance field-informed gaussian splatting for robust real-time rendering with 900+ fps. In *International Conference on 3D Vision 2025*, 2025. 3, 4

[17] Panagiotis Papantonakis, Georgios Kopanas, Bernhard Kerbl, Alexandre Lanvin, and George Drettakis. Reducing the memory footprint of 3d gaussian splatting. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 7 (1), 2024. 2

[18] Lukas Radl, Michael Steiner, Mathias Parger, Alexander Weinrauch, Bernhard Kerbl, and Markus Steinberger. StopThePop: Sorted Gaussian Splatting for View-Consistent Real-time Rendering. *ACM Transactions on Graphics*, 4(43), 2024. 2

[19] Kerui Ren, Lihan Jiang, Tao Lu, Mulin Yu, Linning Xu, Zhangkai Ni, and Bo Dai. Octree-gs: Towards consistent real-time rendering with lod-structured 3d gaussians. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–15, 2025. 1, 3

[20] Samuel Rota Bulò, Lorenzo Porzi, and Peter Kontschieder. Revising densification in gaussian splatting. In *Computer Vision – ECCV 2024: 18th European Conference, Milan, Italy, September 29–October 4, 2024, Proceedings, Part LXIII*, page 347–362, Berlin, Heidelberg, 2024. Springer-Verlag. 2

[21] Yunji Seo, Young Sun Choi, Hyun Seung Son, and Youngjung Uh. Flod: Integrating flexible level of detail into 3d gaussian splatting for customizable rendering, 2024. 1, 3

[22] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *NeurIPS*, 2020. 4

[23] Jonathan Tremblay, Moustafa Meshry, Alex Evans, Jan Kautz, Alexander Keller, Sameh Khamis, Thomas Müller, Charles Loop, Nathan Morrical, Koki Nagano, Towaki Takikawa, and Stan Birchfield. RTMV: A ray-traced multi-view synthetic dataset for novel view synthesis. In *ECCV Workshop on Learning to Generate 3D Shapes and Scenes*, 2022. 6, 1, 3

[24] Karan Vagadia. Donut 3d model. https://www.cgtrader.com/free-3d-models/food/beverage/realistic-donut-fa8648b6-f2f6-4fb3-93d9-ce531d2ba013, 2021. 3D model, accessed on May 20, 2024. 7, 2, 3

[25] Zhou Wang, Alan Bovik, Hamid Sheikh, and Eero Simoncelli. Image quality assessment: From error visibility to structural similarity. *Image Processing, IEEE Transactions on*, 13:600 – 612, 2004. 6

[26] Zhangyang Xiong, Chenghong Li, Kenkun Liu, Hongjie Liao, Jianqiao Hu, Junyi Zhu, Shuliang Ning, Lingteng Qiu, Chongjie Wang, Shijie Wang, et al. Mvhumannet: A large-scale dataset of multi-view daily dressing human captures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19801–19811, 2024. 7, 1, 3

[27] Xijie Yang, Linning Xu, Lihan Jiang, Dahua Lin, and Bo Dai. Virtualized 3d gaussians: Flexible cluster-based level-of-detail system for real-time rendering of composed scenes. In *ACM SIGGRAPH 2025 Conference Papers*. Association for Computing Machinery, 2025. 1, 3, 5, 6, 7, 4

[28] Vickie Ye, Ruilong Li, Justin Kerr, Matias Turkulainen, Brent Yi, Zhuoyang Pan, Otto Seiskari, Jianbo Ye, Jeffrey Hu, Matthew Tancik, and Angjoo Kanazawa. gsplat: An open-source library for gaussian splatting. *Journal of Machine Learning Research*, 26(34):1–17, 2025. 1, 6

[29] Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. Mip-splatting: Alias-free 3d gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19447–19456, 2024. 3

[30] Zhaoliang Zhang, Tianchen Song, Yongjae Lee, Li Yang, Cheng Peng, Rama Chellappa, and Deliang Fan. LP-3DGS: Learning to prune 3d gaussian splatting. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. 2

[31] Brent Zoomers, Maarten Wijnants, Ivan Molenaers, Joni Vanherck, Jeroen Put, Lode Jorissen, and Nick Michiels. PRoGS: Progressive Rendering of Gaussian Splats . In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 3118–3127, Los Alamitos, CA, USA, 2025. IEEE Computer Society. 3, 4

NVGS: Neural Visibility for Occlusion Culling in 3D Gaussian Splatting

Supplementary Material

6. Supplementary

In our supplementary material, we present additional results for all the layout files and show renders and FLIP comparisons for all layouts. This includes the results on the full layout files compared with our baseline instanced rasterizer implementation. Results for individual assets are presented, along with visual differences between the ground truth contribution and our predicted values. We also encourage the reader to view the supplementary videos, as the results can be more clearly inspected when the camera is in motion.

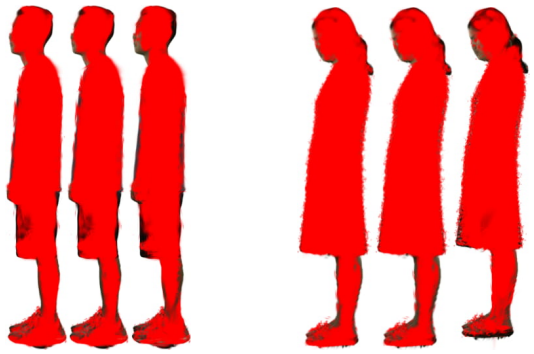
6.1. Per-asset Results

In Tab. 4 we show in-depth per-asset results. These metrics were measured using a circular trajectory around the asset. For the assets in the DONUTSEA and CROWD layouts, our MLP is able to cull a significant number of Gaussians, ranging from ∼29% to ∼40%. For the trees from the FOREST layout, the amount that can be pruned is heavily dependent on the crown and the thickness of the trunk. In the cases of oak and dinosaur, we show in the video that exactly these factors complicate the accurate learning of the visibility function. Our approach, on average, increases the percentage of Gaussians that are used relative to the number passed, and significantly decreases the number of Gaussians passed. We want to emphasize that these Gaussians can be completely discarded without any noticeable loss in quality, as they reflect zero-contribution values in the ground-truth renders.

In Fig. 9, Fig. 10, and Fig. 11, we present side-by-side images of a selection of assets, comparing the ground-truth selection with what our MLP predicts as occluded. From left to right, we show the ground truth on the single view, the ground truth we actually learn using the small set of auxiliary views to counter training on popping artifacts, and on the right, what our MLP predicts is occluded. For all views, (predicted) non-contributing splats are rendered in red as if the camera is located on the opposite side of the asset. For the avatars and trees, we display the asset with the best and worst performance in terms of the percentage of Gaussians pruned. For the trees, notice how the two trees that perform worse have lots of small gaps in their crown, making the visibility function have a higher frequency than for the avatars or donut. Including frequency encodings and not using auxiliary views both help increase the number of Gaussians we predict as occluded, at the cost of rendering speed and robustness against popping artifacts. Depending on the application, the selection of steps in the process can be fine-tuned; however, our proposed method, with all steps enabled, achieves consistent results across assets and datasets.

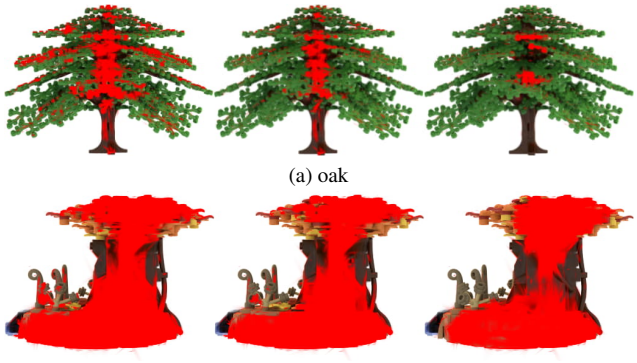
6.2. Large-scale Scenes

Our approach enables the rendering of large, composed scenes using significantly less VRAM than other methods, thanks to the combination of our instanced rasterizer and occlusion culling MLP. In Fig. 12, we show the renders and FLIP on the downscaled scenes compared to V3DG [27] for the scenes not in the main paper, and in Fig. 13, we show the renders of the scene at full scale compared to our baseline. For both our method and V3DG, we observe a concentration of errors on the borders in the CROWD scene. While we do



(a) 100843 (b) 100845

Figure 9. From left to right, we show the ground truth zero-contribution Gaussians, ground truth zero-contribution using 6 auxiliary views, and what the MLP predicts on the right. The two chosen assets represent the best- and worst-performing assets within the MVHumanNet [26] dataset.



(a) oak

(b) fall

Figure 10. From left to right, we show the ground truth zero-contribution Gaussians, ground truth zero-contribution using 6 auxiliary views, and what the MLP predicts on the right. The two chosen assets represent the best- and worst-performing assets within the RTMV [23] trees subset.

(a) donut

Figure 11. From left to right, we show the ground truth zero-contribution Gaussians, ground truth zero-contribution using 6 auxiliary views, and what the MLP predicts on the right. We show this for the donut asset [24]

not have access to how the original assets were trained, we speculate this could be due to the assets being trained on a different background than white. If this is the case, it can be solved by training on a random background, improving the results of V3DG and our method. Our approach can run these scenes on an NVIDIA RTX 3090 Ti with a limited amount of 24GB VRAM at $>$20FPS.

	GT			Ours			
Asset	Passed	Used	Used (%)	Passed	Used	Used (%)	**Δ Passed (%)**
100831	313,257	151,748	48.4%	199,207	148,328	74.5%	-36.41%
100833	273,513	134,121	49.0%	183,906	130,578	71.0%	-32.76%
100835	302,469	153,949	50.9%	206,470	151,189	73.2%	-31.74%
100837	331,681	164,372	49.6%	214,328	159,772	74.5%	-35.38%
100839	295,192	146,948	49.8%	202,081	143,489	71.0%	-31.54%
100841	301,718	154,509	51.2%	194,411	151,447	77.9%	-35.57%
100843	278,513	141,775	50.9%	198,186	138,574	69.9%	-28.84%
100845	824,204	393,820	47.8%	494,592	387,509	78.3%	-39.99%
100847	334,039	158,388	47.4%	207,953	154,531	74.3%	-37.75%
100849	382,225	182,777	47.8%	235,006	177,868	75.7%	-38.52%
100851	318,754	149,637	46.9%	201,116	145,765	72.5%	-36.91%
100853	368,617	180,419	48.9%	241,633	176,145	72.9%	-34.45%
100855	286,111	140,272	49.0%	189,134	137,433	72.7%	-33.90%
100857	281,235	136,311	48.5%	181,743	132,082	72.7%	-35.38%
100859	229,308	109,789	47.9%	148,576	107,207	72.2%	-35.21%
100861	264,017	128,745	48.8%	170,439	125,885	73.9%	-35.44%
bonsai	319,339	209,366	65.6%	284,781	204,559	71.8%	-10.82%
crispy	264,978	191,948	72.4%	237,725	191,353	80.5%	-10.29%
desk	425,546	244,866	57.5%	339,417	237,974	70.1%	-20.24%
dinosaur	221,706	144,505	65.2%	207,475	143,451	69.1%	-6.42%
fall	276,574	147,584	53.4%	202,262	144,641	71.5%	-26.87%
house	273,216	168,544	61.7%	240,408	167,239	69.6%	-12.01%
oak	257,670	205,816	79.9%	246,392	205,118	83.2%	-4.38%
pine	264,686	156,355	59.1%	202,004	152,392	75.4%	-23.68%
donut	44,106	26,267	59.6%	29,964	26,052	86.9%	-32.06%

Table 4. Per-asset statistics. We display the number of Gaussians passed to the rasterizer, as well as the Gaussians that actually contribute to the final rendering, both in absolute numbers and as percentages relative to the total number of Gaussians passed. We then show the difference in Gaussians passed between the ground truth and our method in the last column. We divide the assets into their three respective datasets. The first 16 entries are from the MVHumanNet dataset [26], the next eight from the RTMV [23] dataset, and the donut from [24]. We prune a significant number of Gaussians across all assets. For oak and dinosaur, the lower gains can be dedicated to the higher-frequency visibility functions, which comprise the majority of the asset. While this could be improved by using frequency encodings, we argue that the slowdown in general for these specific assets is not worth the slight increase for a small subset of assets.

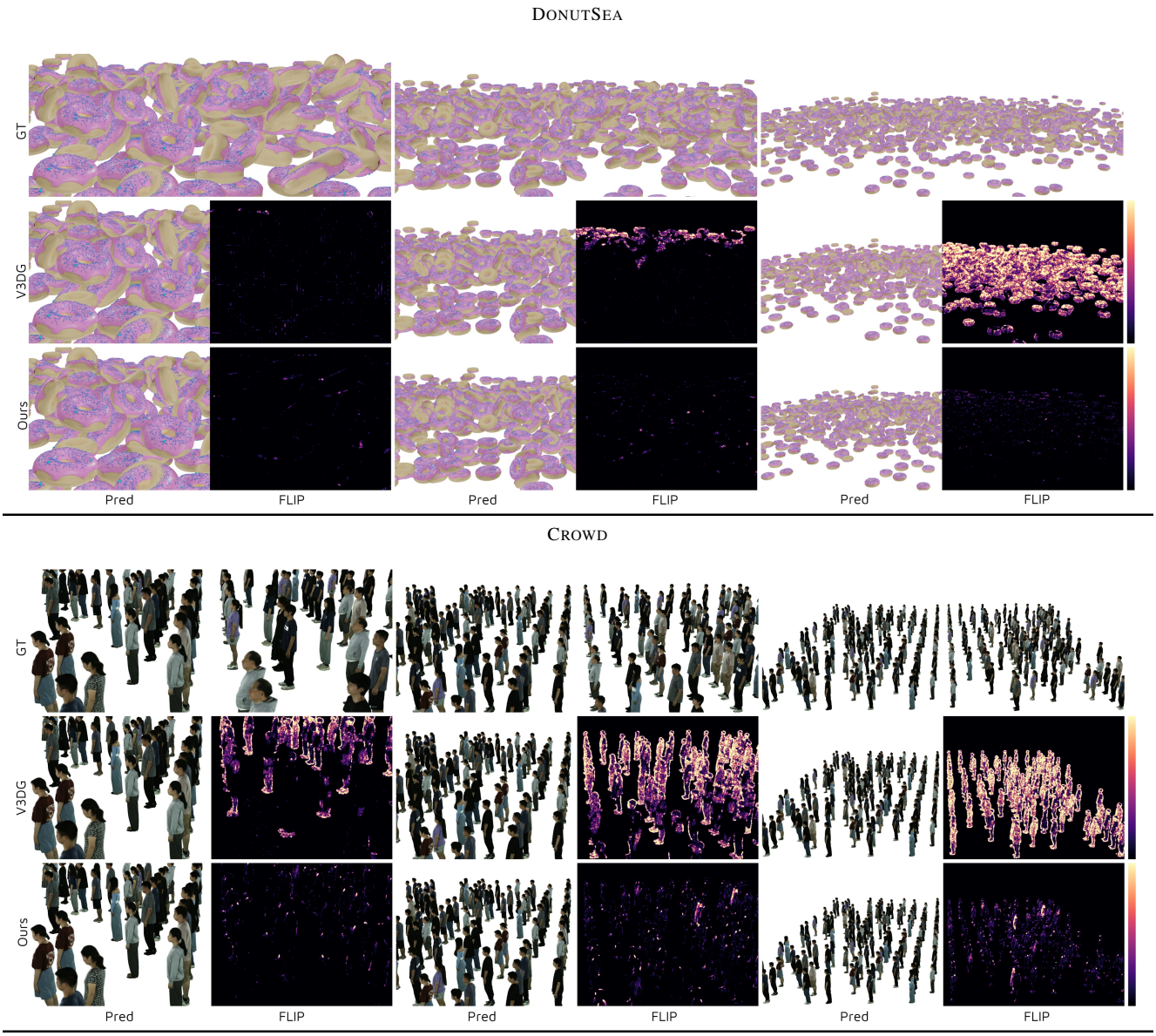


Figure 12. Shows GT render using gsplat, V3DG, and our method for close, medium, and far distances on the DONUTSEA and CROWD layouts. For V3DG and ours, we show the prediction on the left and FLIP on the right. We scaled FLIP by a factor of five to facilitate better visual comparison. These scenes have been downsampled by a factor of two compared to the original layouts provided by V3DG [27] to ensure that all methods can be tested without exceeding VRAM limits.

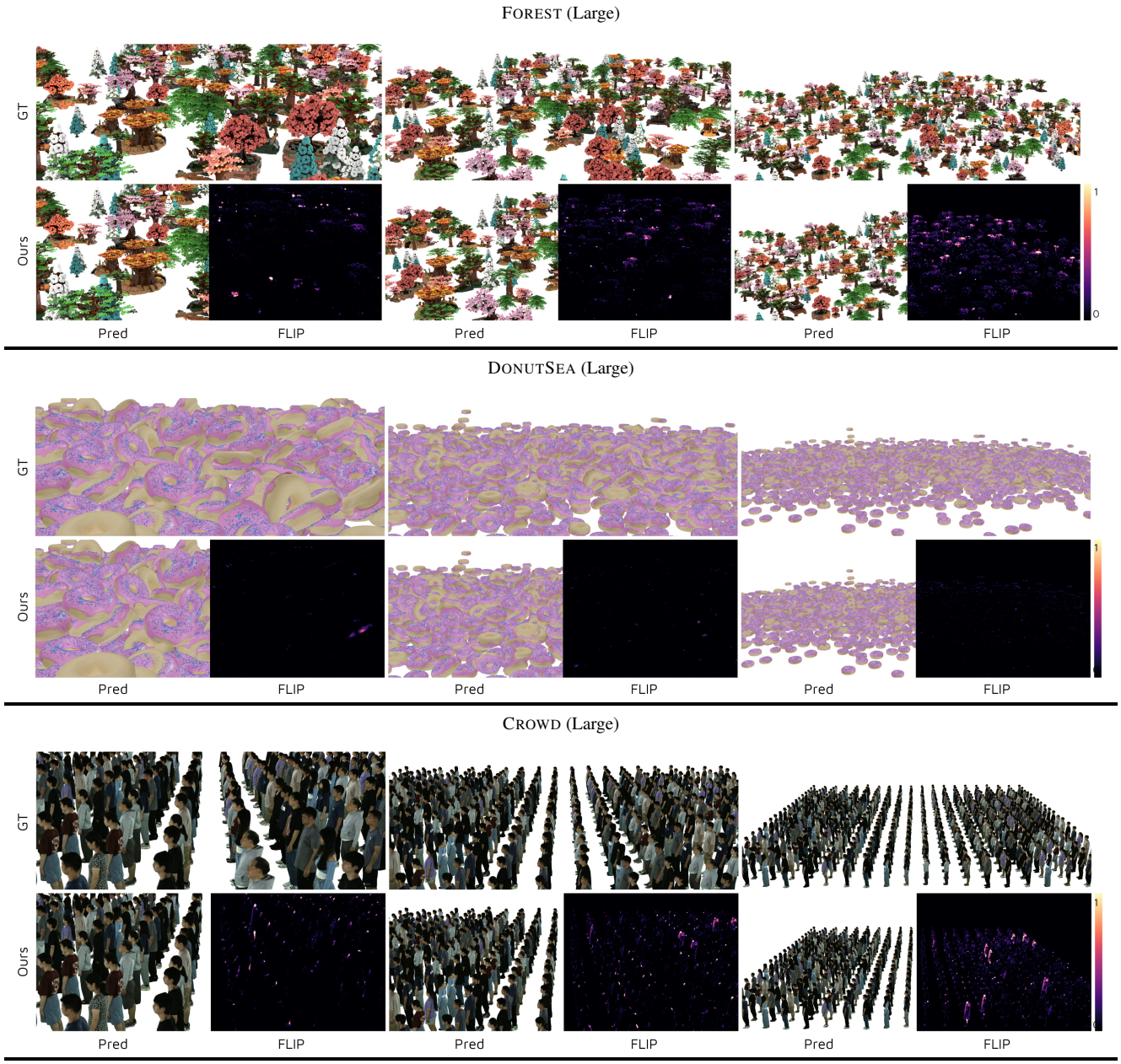


Figure 13. Shows GT render using our method with and without MLP(solely using instanced rasterizer) for close, medium, and far distances on the FOREST, DONUTSEA and CROWD layouts. For ours, we show the prediction on the left and FLIP on the right. These renderings show the original layout files without downscaling(individual assets have still been centered after removing all Gaussians with an opacity lower than $\frac{1}{255}$). We scaled FLIP by a factor of five to facilitate better visual comparison.