PRoGS: Progressive Rendering of Gaussian Splats

Brent Zoomers, Maarten Wijnants, Ivan Molenaers, Joni Vanherck, Jeroen Put, Lode Jorissen, Nick Michiels

Hasselt University - Flanders Make - Digital Future Lab, Diepenbeek, Belgium
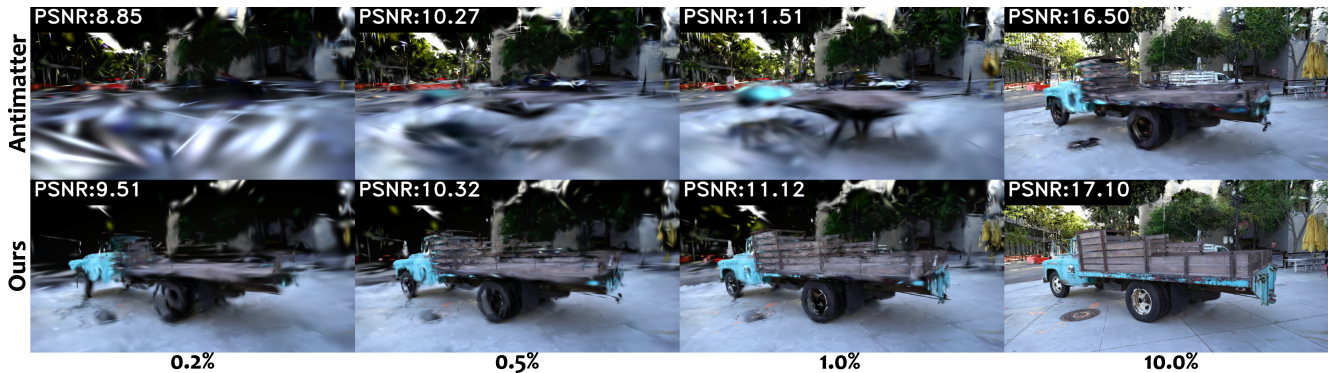
`firstname.lastname@uhasselt.be`

Figure 1. Progressive Rendering of 3DGS using our approach versus using an existing web-viewer by Antimatter. From left to right, we show 0.2%, 0.5%, 1%, and 10% of the total number of splats, respectively. Our approach results in a faster visualization of a representative version of the scene. At 0.2%, we have loaded in a basic level of the truck while it is completely missing in the alternative approach.

Abstract

Over the past year, 3D Gaussian Splatting (3DGS) has received significant attention for its ability to represent 3D scenes in a perceptually accurate manner. However, it can require a substantial amount of storage since each splat's individual data must be stored. While compression techniques offer a potential solution by reducing the memory footprint, they still necessitate retrieving the entire scene before any part of it can be rendered. In this work, we introduce a novel approach for progressively rendering such scenes, aiming to display visible content that closely approximates the final scene as early as possible without loading the entire scene into memory. This approach benefits both on-device rendering applications limited by memory constraints and streaming applications where minimal bandwidth usage is preferred. To achieve this, we approximate the contribution of each Gaussian to the final scene and construct an order of prioritization on their inclusion in the rendering process. Additionally, we demonstrate that our approach can be combined with existing compression methods to progressively render (and stream) 3DGS scenes, optimizing bandwidth usage by focusing on the most important splats within a scene. Overall, our work establishes a foundation for making remotely hosted 3DGS content more quickly accessible to end-users in over-the-top consumption scenarios, with our results showing significant improvements in quality across all metrics compared to existing methods.

1. Introduction

In recent years, radiance fields have attracted significant attention due to their ability to accurately portray real-life scenes while requiring only a limited number of input images. Neural Radiance Fields [22] (NeRFs) played an important role in this resurgence as they allowed scenes to be represented implicitly by using a multilayer perceptron (MLP) to predict the volume density and view-dependent emitted radiance based on an input position and look-at direction. After the introduction of NeRF, follow-up work has focused on improving different aspects of the approach ranging from training time [23] and editing [21] to other downstream tasks such as meshing [27]. Most of these approaches follow the same principles as NeRF and thus fall under the category of implicit approaches. Generally, they require less storage than explicit approaches at the expense of making downstream tasks less intuitive and often more difficult than explicit approaches, as there is a lack of direct manipulability.

Figure 2. Our approach works by rendering all training views and returning the top 20 contributing splats per pixel per image. This information is used to create an ordering which can then be used to dynamically select how many splats are used. For progressive rendering/streaming, we can change the number of splats per sent chunk to start rendering as soon as possible without the user needing to wait. With little overhead in total time, we drastically reduce both time and bandwidth until time to First Paint.

3D Gaussian Splatting (3DGS) [16] takes a different approach by shifting from an implicit to an explicit representation using 3D Gaussians. By optimizing a set of 3D Gaussians to represent a scene, 3DGS allows both fast training and inference, which has been pivotal in allowing a more widespread adoption of these techniques. Another important factor is that the format used to store these scenes is an extension of a regularly used point cloud format. This factor plays an important role in facilitating downstream tasks as it simplifies the process of experimenting with existing techniques as inspiration to bring them to the context of 3DGS. Our work will cover two downstream examples, *progressive rendering* and *progressive streaming*. Historically, progressive rendering has played an important role in content visualization, as visualizing the required content to the user as soon as possible is crucial in achieving a proper user experience. Progressive streaming focuses predominantly on the network distribution of media content, such as audio and video, where it is crucial that the user is able to consume the content as soon as possible. Despite this seemingly perfect fit, transferring existing progressive rendering methods to 3DGS is not trivial, as these do not fully utilize the information provided by 3DGS. Besides position, we additionally have access to the opacity, covariance, and view-dependent color information, which can all be used to achieve better results.

In our work, we thus bring progressive rendering (and, by extension, streaming) to the context of 3D Gaussian Splatting by utilizing a contribution-based prioritization method to determine a rendering order in post-training. We then integrate this with other contextual information, such as the current viewport, to ensure that relevant content is visualized to the user promptly. Concretely, using our approach allows us to reduce rendering delay and achieve faster time-to-first-paint times, which is expected to improve user experience. An overview of our approach is given in Figure 2. By combining different commonly used graphics techniques together with our contribution-based ordering, we facilitate the visualization of the approx-

imated scene using only a fraction of the data required to store the complete scene. We then incrementally use the remaining data to continuously improve quality over time (Fig. 1). Our work is the first to bring progressive rendering of 3DGS into an academic context. We show that our proposed method outperforms available web-based approaches regarding quality per percentage splats used and show possibilities for further reductions in bandwidth utilization, using existing compression methods to bring progressive rendering to 3DGS scenes. Combining both compression and progressive rendering allows us to send an initial representative scene using only fraction storage required to store the complete uncompressed scene. Furthermore, our approach works for both complete scenes and on individual objects within a scene, paving the way for further application-specific research and applications.

In Sec. 2, we will discuss a selection of relevant works concerning progressive streaming, rendering, and compression of 3DGS scenes and sketch where our approach is situated within the domain. Sec. 3 will discuss the different aspects of our approach and how they contribute to the final result. Sec. 4 illustrates our results and performs a comparative study with existing web viewer-based approaches currently used within the community, an ablation study, the results for per-object ordering, and compression. Lastly, Sec. 5 outlines potential future trajectories and summarizes our key conclusions.

Concretely, our contributions can be summarized as follows:

- We propose an importance-based sampling method to decide the order of transmission that works on both the full scene and individual objects within a scene and is viewpoint-independent.

- We show an integration of progressive rendering into an existing compression method and show the potential for further bandwidth savings by using a combination of both.

- We show that our approach can be extended with frustum culling and an octree-based prioritization method to boost our performance further.

2. Related Work

This section will briefly cover the related works of our approach. We will first discuss progressive rendering and streaming as they form the core contribution of our approach. We will then highlight several compression papers as both fields focus on quickly bringing content to the user. While compression achieves this by lowering the memory footprint, progressive approaches focus on sending the relevant content first.

Progressive Rendering of 3D content   Progressive rendering of scenes has been an interest of research for decades as it plays a crucial role in facilitating the efficient rendering of 3D scenes. The content used has increased in both size and amount of detail over the years, necessitating more efficient usage of bandwidth and memory. Progressive rendering allows applications to start rendering a scene without the need for it to be fully loaded into memory or retrieved from a remote server first. Hoppe [11] first introduced the term *progressive* in 1996, specifically in the context of triangle meshes. He proposed to iteratively simplify a mesh using the edge collapse operation, which unifies two adjacent vertices into a single vertex. The simplest version is sent first, after which the inverse operation, vertex split, can be applied to fully reconstruct the original mesh. These inverse operations can be applied incrementally, allowing the application to progressively load the mesh. At a later stage, Hoppe [10] revisited his earlier work and integrated view-dependent aspects into it. This updated approach used the viewing frustum, surface orientation, and screen-space projected error to take more context into account while reconstructing the mesh. More recently, Chen et al. [3] have explored the use of neural networks to learn a progressively compressed representation for meshes. Their approach converts a given mesh into a datastream that can be progressively transmitted. The client can then decode this datastream to reconstruct a simplified mesh, which can be iteratively improved upon using subsequent transmissions. This process can continue until the original mesh is reconstructed or the required quality is reached.

Over the years, progressive rendering has been used for other representations besides meshes. Schütz et al. [29] focus on progressively rendering point clouds. They achieve this by projecting the previous set of points onto the viewing frustum and only transmitting points that appear in unoccluded areas of the new frame. In the context of radiance fields, BungeeNeRF [35] allows progressive rendering of NeRFs for multi-scale scene rendering. The network progressively scales with the scale of the learned scene, aiming to divide quality into different layers. Reconstruction quality can then be decided by querying a different head that corresponds to a certain quality.

Progressive Streaming of 3D content   The distinctive difference between progressive rendering and progressive streaming is that the latter also involves the network delivery of the media content from a remote server to the consuming client device. As such, with progressive streaming, the client progressively renders the content at the rate it receives from the content server. This concept is widely adopted in the over-the-top delivery of conventional audio-visual content via the HTTP Adaptive Streaming (HAS) paradigm and its standardized MPEG-DASH implementation [13, 31]. Recently, the use of progressive streaming is also being explored for more immersive media formats. Noteworthy examples are the HAS-like progressive streaming of textured geometry (e.g., [6, 7, 18, 19, 30, 36]), light fields (both static [20, 34] and dynamic [12, 15]) and point clouds (e.g., via the MPEG V-PCC specification [14, 33]). With respect to the progressive streaming of radiance fields, the attention of the academic community has focused mostly on the streaming of NeRFs representing scenes that are static (e.g., Cho et al. [4]). Of specific interest is the NeRFHub approach by Chen et al. [2], which aims to minimize the network transfer latency of NeRF models while satisfying lower limits regarding rendering smoothness and perceptual quality. NeRFHub does so by (amongst others) shrinking the number of hidden MLP layer channels (in combination with selective model training) and by quantizing the feature grid's floating point values. While NeRFHub shares our objectives of reducing start-up latency during over-the-top radiance field consumption, no progressive streaming nor rendering is applied, as NeRFHub always deals with (quality-variant) integral NeRF models. Finally, to date, progressive 3DGS streaming remains largely unexplored in academic literature; in effect, this paper represents a fundamental step in that direction.

Compression of 3D Gaussian Splatting   Compression and progressive streaming share a similar goal of reducing the bandwidth needed to render a scene. While progressive streaming focuses on sending relevant content first, the goal of compression is to minimize the total memory footprint. As this goal is very similar, and both approaches can be used in conjunction with each other, we will briefly overview the relevant works in this field.

We argue that there are two main ways of performing compression of 3DGS. The first way focuses on compressing a scene while maintaining all existing parameters. Codebooks and vector quantization are often used in these approaches as they allow existing data to be stored with less memory, albeit with some loss in accuracy. The second way focuses on adapting the actual parameters themselves to a more efficient format. Niedermayr et al. [25] performed compression by utilizing a codebook to store Gaussian pa-

rameters. After converting an existing scene into its codebook representation, they fine-tune all parameters using the training images to regain lost quality due to discretization. Further compression is achieved by ordering the Gaussians and using run-length encoding to create the final representation. Compact3D [24] works under the assumption that large groups of Gaussians will be likely to share parameters. They then use K-means clustering to group similar Gaussians and use a codebook to store data per cluster. They also promote fewer Gaussians in a scene by adjusting the opacity values to be close to one or zero. EAGLES [8] uses a technique similar to our own as they also use contribution as a metric to filter splats. Besides this, they use vector quantization to reduce storage further. The previously mentioned approaches all fall under the first category and focus on compressing the existing parameters into a smaller format. Papantokankis et al. [26] propose different methods in their work, such as resolution-aware pruning and adaptively adjusting the number of coefficients used to model directional radiance. LightGaussian [5] also focuses on the directional radiance and distills spherical harmonics to a lower degree, thereby compressing the largest contributor to memory usage. Our approach can be used together with any compression technique that allows for some factorization based on the contribution of a Gaussian to a given pixel.

3. Methododology

Our approach allows progressive rendering of 3DGS by determining an ordering among Gaussians. By prioritizing Gaussians that have contributed significantly to the scene, we reduce the number of bytes needed for a qualitative approximate reconstruction. By rendering each training viewpoint and calculating the contribution across all views, we create an initial ordering of the Gaussians. We further refine this by inserting all Gaussians into an octree and selecting the top contributing Gaussians per leaf node (mean inside leaf node). This results in the splats being more evenly spread across the scene, with a small hit to the quality of the foreground. Additionally, we can enable frustum culling to further increase the perceptual quality of the visible scene. Besides creating an order at the scene level, our approach also functions at an object level. This allows us to prioritize splats within individual objects or to prioritize objects within the scene. Our final contribution is the integration of our technique into an existing compression method to fully demonstrate the capabilities of this combination when it comes to reducing storage and bandwidth requirements. In the following subsections, we will discuss each aspect of our approach in more detail.

3.1. Contribution-based ordering

The fundamental idea behind our approach comes from using the contribution of a Gaussian to all rendered views

as a metric of its importance to the scene. Large Gaussians with a high opacity value will most likely contribute more than smaller splats with a low opacity. Using this intuition, we render all training views and calculate the contribution of a Gaussian across them to use as a contribution score. Sorting based on this contribution score leads to an ordering among Gaussians that estimates how important each Gaussian is for reconstructing the final scene. To retrieve the actual contribution value, we render the image and store the weighting factor of the top K contributing Gaussians per pixel. The color C of a pixel is determined as:

$$C = \sum_{i=1}^{N} T_i \alpha_i c_i$$

with

$$T_i = \prod_{j=1}^{i-1} (1 - \alpha_i)$$

where T_i represents the transmittance, α_i the opacity value, c_i the color value, and N loops over the Gaussians overlapping the current pixel. The contribution is then taken as $T_i \alpha_i$. For each pixel, we return a pairing of contribution and Gaussian ID to measure the final contribution of a specific Gaussian as:

$$B_{id} = \sum_{v \in \mathcal{V}} \sum_{p \in \mathcal{P}} (T_{id} \alpha_{id})$$

where $\mathcal{V}$ represents the set of viewpoints and $\mathcal{P}$ represents all pixels per viewpoint. This contribution directly translates into an ordering that reflects the importance of a Gaussian to the scene. It can be used to create chunks of Gaussians that can be rendered or sent independently of each other. EAGLES [8] uses the same metric during training to prune splats that are less impactful during training. Instead, our approach opts to use it in post-training as a way to select impactful splats that are advantageous to send first.

3.2. Refinement of ordering using octree

Using the global ordering created by the initial step, the chosen splats tend to focus on areas densely captured in the input data (see Sec. 4). As we sum up all pixels, Gaussians that have been seen more often will be more likely to have a higher contribution score. This can result in sparsely captured parts being unfairly under-reconstructed. To counter this effect, we utilize an octree to create a spatial subdivision in the scene based on the density of splats. By taking the highest contributing splats per leaf node(using), we effectively spread out the Gaussians over the scene and thus make sure every part is partly reconstructed. Changing the maximum depth of the octree changes how much denser areas are prioritized. Density, in most cases, corresponds to areas of higher detail. This, however, does not always correspond perfectly to the distinction between background
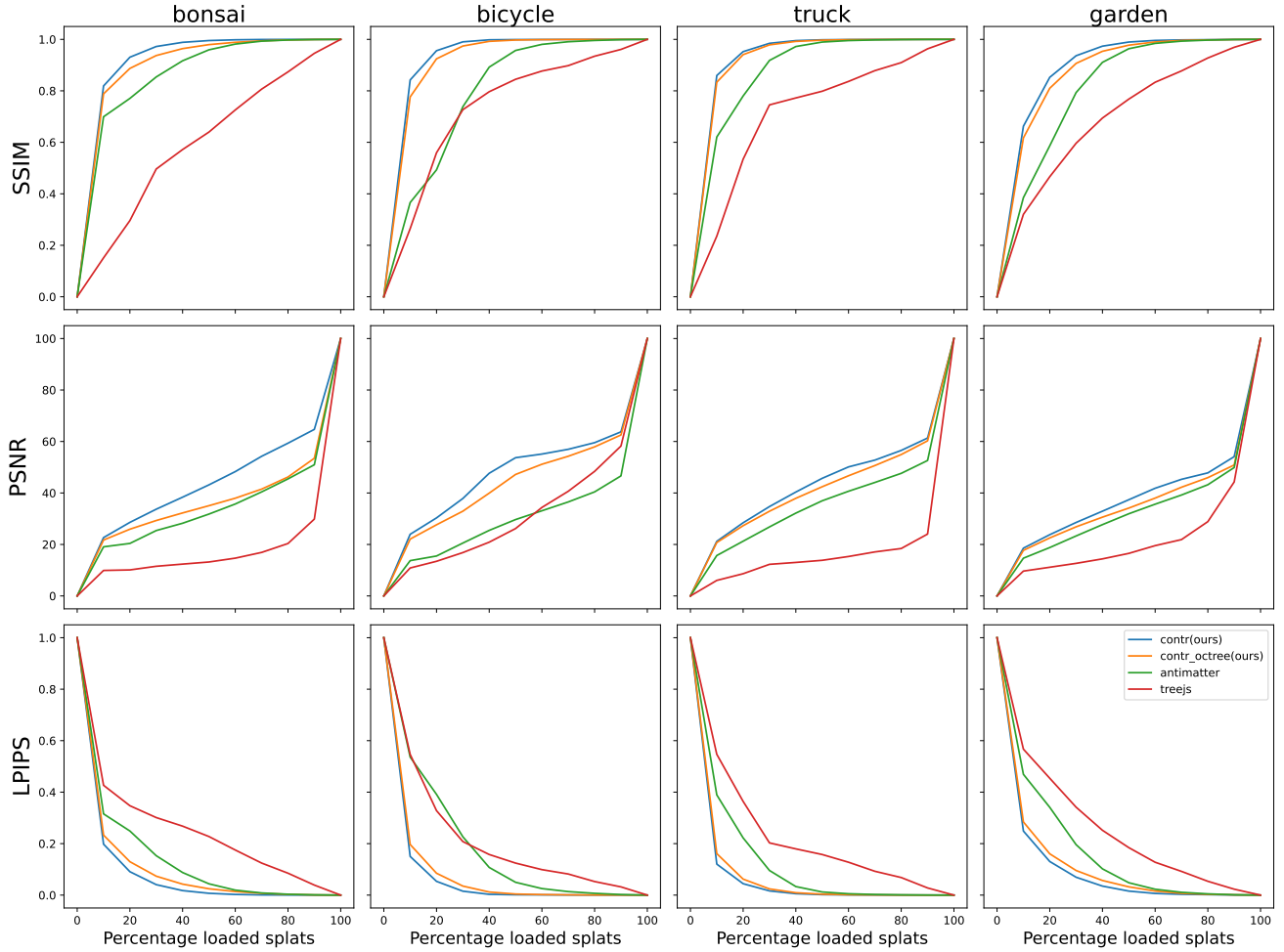
Figure 3. We show the results of different scenes for all three metrics and that our two approaches consistently outperform the previous methods by a significant margin. These scenes are taken from the MipNeRF360 [1] indoor and outdoor datasets and the Tanks&Temples [17] dataset. PSNR is measured compared to the original reconstruction and is capped at 100 when the complete scene is loaded in. (PSNR↑, SSIM↑, LPIPS↓)

and foreground; thus, caution is needed when setting this depth to not over or underprioritize these parts of a scene. In Sec. 4, we show the impact of this parameter in more detail.

3.3. In-frustum prioritization

As the initial pose of the user is known beforehand, we can use it to fine-tune our initial ordering further. Using our initial list of Gaussians, we can select a percentage of splats chosen from within this frustum to allow rapid movement while focusing mainly on what is visible to the user. To determine if a splat lies within the viewing frustum, we project all splats onto the image plane and filter the splats that fall outside it. As Gaussians cover areas and not points on the image plane, there is a chance that a Gaussian of which the mean falls outside the frustum still contributes to the image.

We thus follow 3DGS and allow a small margin around the frustum in which we still count Gaussians as being inside.

3.4. Object-level rendering

By segmenting splats that belong to a specific object, we can create an ordering within an object or compare objects at a scene level based on the sum of their contributions. Within an object, we apply the same principles as at the scene level and prioritize splats that have a significant contribution to the object. Within a given scene, we can prioritize objects based on their contribution to the scene and send higher-quality representations of important objects while sending lower-quality, low-contributing objects. In the context of progressive rendering, both these cases are important to have as they allow more advanced progressive rendering/streaming techniques to be implemented. For

now, we achieve this by manually segmenting objects or using existing 2D segmentation methods [28] and finding the contribution of splats within this segmentation to determine the importance of objects for a given scene. This can, however, also be used for scenes that are composed of different trained models.

3.5. Integration into compression methods

As our approach is agnostic to any Gaussian-specific parameter and can use solely contribution to calculate an ordering, our approach can be applied to other 3DGS-inspired approaches. We have to point out that our approach only has benefits when per-Gaussian data is stored, which despite the frequent use of codebooks, is the case for the majority of compression papers. To show this claim holds up, we integrate our approach into the approach of Niedermayr et al. [25] and demonstrate that our approach achieves similar orders of progressive rendering gains. We simply adapted the renderer to include passing back the contribution values and IDs per Gaussian and calculating our ordering afterward. Caution has to be taken as this approach sorts the Gaussians in Morton order before saving them, which completely voids our previous ordering. Furthermore, when vector quantization is done using the minimum and maximum values, they have to be passed as well to allow for dequantization.

4. Experiments

Dataset. We used the MiPNeRF360 [1] dataset (inside & outside), two scenes from the Tanks&Temples [17] dataset, and two scenes from the Deep Blending [9] dataset to validate our approach. We used the original code of 3DGS [16] to split the data into train and test sets to promote consistency.

Evaluation metrics. We use community-standard image loss metrics Peak-signal-to-noise ratio (PSNR), Learned Perceptual Image Patch Similarity (LPIPS), and Structural Similarity Index Measure (SSIM) to measure the visual quality. We compare each approach to the final trained approach and not the ground truth to show how well it approximates sending the complete trained scene.

Implementation details. For our experiments, we used the contribution of the 20 highest contributing Gaussians per pixel to decide on the ordering, which is an overestimation in most cases. In experiments using an octree (Sec. 3.2), we use a depth of 3 unless otherwise stated. For frustum culling, we allow some margin by increasing the frustum boundaries by 30%. For experiments with specific objects within a scene, we manually segmented them from the trained scene. We used the approach by Niedermayr et al. [25] for all experiments concerning compression and adapted their approach to allow for chunking of the compressed files into different independent files based on contribution to the training views. Concretely, we insert the minimum and maximum splats to allow for dequantization when loading the .ply files.

4.1. Quantitative Evaluation

As our work is the first to bring progressive rendering of 3DGS into the academic context, we compare ourselves with widely used online web viewers. We also show several variants of our own work and verify that our approach consistently outperforms other approaches. Concretely, we compare our implementation to *3D Gaussian Splatting with Three.js*[1] and *3D Gaussian Splatting viewer by antimatter15*[2]. Most other available viewers are closed-source or require the complete .ply file to be downloaded before rendering starts, which can take from multiple seconds up to several minutes. This further demonstrates the need for open-source progressive rendering approaches. Antimatter's viewer prioritizes splats based on a combination of their opacity and scale:

$$G_{contr} = \frac{-e^{scale.x+scale.y+scale.z}}{1+e^{-opacity}}$$

while the three.js implementation opts to load in splats starting from the center. We can simply use the Euclidean dis-

---

[1]https://github.com/mkkellogg/GaussianSplats3D
[2]https://github.com/antimatter15/splat

(a) Ground Truth   (b) Ours   (c) Antimatter   (d) Three.js

Figure 4. A concrete comparison between the three aforementioned methods on the *bonsai* scene taken from Mip-NeRF360 indoor scenes being rendered at 10% splats compared to the ground truth. Notice how both the bonsai and cloth are being rendered at a higher quality in Ours(b) than in Antimatter(c). Tree.js(d) will almost always be outperformed using standard metrics as the background is completely missing.

3123

(a) Base   (b) Base+Octree

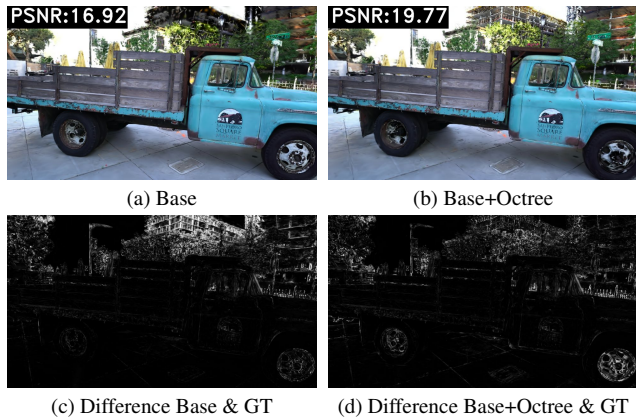(c) Difference Base & GT   (d) Difference Base+Octree & GT

Figure 5. Despite the right image (b) performing better than the left image (a), we can clearly see there are more errors in the foreground when comparing (c) with (d).

tance from the center as a metric of contribution:

$$G_{contr} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

where the center is taken as (0,0,0). Our approach is calculated as in Sec. 3.1. In Fig. 3, we show that our approach outperforms previous methods by a significant margin across different datasets and metrics. The images in Fig. 4 also demonstrate a clear perceptual improvement of our approach. The *bonsai* and *tablecloth* are loaded in at a higher quality while using the same amount of splats. Large splats that contribute to multiple pixels will inherently get a higher score and, thus, are more likely to be sent first. As each pixel's contribution is capped at 1, smaller splats are inherently disadvantaged and will need a large contribution to receive a higher overall score. We argue that these attributes are wanted behavior which is proven by Fig. 4 clearly showing a large jump in quality over the previous methods.

4.2. Ablation Study

The base implementation (see Sec. 4.1), which solely used the contribution across pixels and views as a metric, achieves state-of-the-art results. We will now show the impact of the aforementioned additions to this base implementation and their impact on the final rendering quality. Our first addition is the octree, which forces Gaussians to spread out more over the scene. This results in the sparsely captured parts of the scene being loaded in sooner, as can be seen in Fig. 5. On average, including the octree approach results in a loss in PSNR as the foreground often contributes more to PSNR. Perceptually, however, we observe a clear difference between both images. Considering the difference images in Fig. 5, the background shows fewer errors while the foreground becomes slightly worse. This deterioration of the foreground results in lower PSNR scores on average.

(a) Depth 1   (b) Depth 3
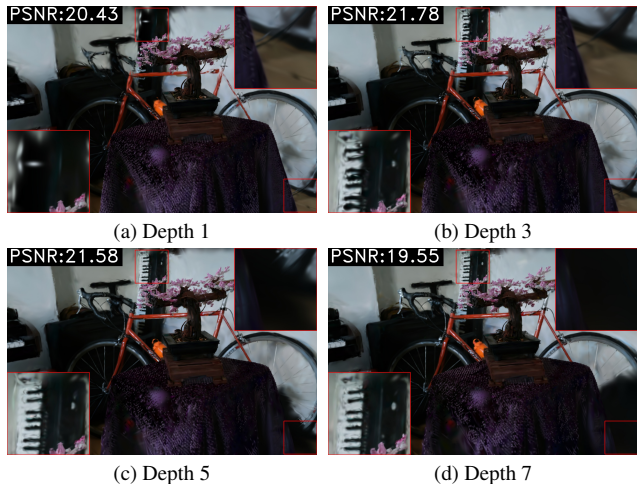
(c) Depth 5   (d) Depth 7

Figure 6. We show the impact of the octree depth parameter on the rendering quality. Using a depth that is either too low or too high will result in not enough attention being given to the background. Using 3 as the depth parameter consistently yields good results across all scenes.

For now, we leave this as a subjective observation, and future user studies will be needed to verify our intuition that using the octree does, in fact, lead to a better user experience. Fig. 6 then shows the impact of the depth parameters, clearly illustrating the negative impact of selecting a depth that is too high. Lower depth values have little to no impact, as can be seen by keys on the keyboard not being present. Medium depth values, such as 3 and 5, perform well in both the foreground and the background. A value of 7 starts introducing errors, as can be seen by the black blur on the bottom right. As this value gets larger, it comes closer to the base approach. The next addition is frustum culling, where we focus on loading in splats that are visible to the user. Fig. 7c shows that the foreground becomes marginally better. As more than 10% of all splats are visible within the current frustum, the small background splats are still not selected. This partly occurs as frustum culling occurs after the global ordering has been determined. We thus do not take into account occlusions from the current viewpoint. Using our approach with the octree results in a good representation of the scene without requiring any user-specific knowledge, while the complete approach, with frustum culling, will perform better when we have access to such information.

4.3. Object level

We manually segmented several objects and verified our approach on each object. Fig. 8 shows that our approach enables progressive object rendering, loading each object solely based on its contribution. We tested this on several objects segmented from the aforementioned datasets and consistently achieved better scores per percentage loaded

3124

(a) Base   (b) Base+Octree

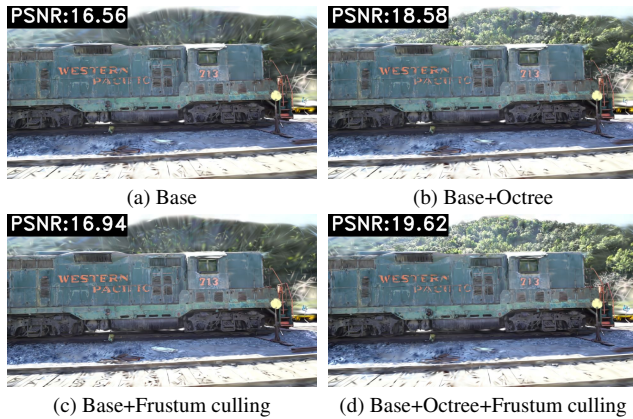(c) Base+Frustum culling   (d) Base+Octree+Frustum culling

Figure 7. This figure shows the progression from our base approach to the complete approach going from top left to bottom right. Notice how the background is noticeably better when using the octree to spread out Gaussians. The impact of frustum culling becomes more noticeable when we use the octree as we enhance (b) further by forcing even more splats to lie inside the frustum.

splats compared to other methods.

4.4. Integration with compression

Our final contribution is the integration of our approach into an existing compression method to show the impact it can have on the quality at the time to First Paint. We adapted the technique of Niedermayr et al. [25] to include our contribution-based ordering technique. To integrate our approach, the save-functionality has to be slightly rewritten to consider alterations from the base approach. Concretely, to perform vector quantization, the minimum and maximum values are needed for all values within said chunk. If every chunk needs to be rendered independently from the other chunks, the minimum and maximum need to be included in every chunk. Furthermore, the Gaussians are sorted in Morton order in the original approach to allow for further compression. This step happens right before saving the scene, requiring us to apply this sorting on the selected indices as well. We disabled this for now as this has no impact on the selected splats and only on the final storage requirements. Fig. 9 shows that we achieve similar results comparing quality per percentage of splats compared to our base approach on the standard 3DGS implementation. In actually required memory, there is a smaller difference as the shared codebook reduces the amount of Gaussian-specific data.

5. Conclusion and Future Work

We proposed a novel approach that facilitates progressive rendering of 3DGS and 3DGS-inspired methods by utilizing the contribution of Gaussians to determine an order of importance among them. Our approach allows us to provide
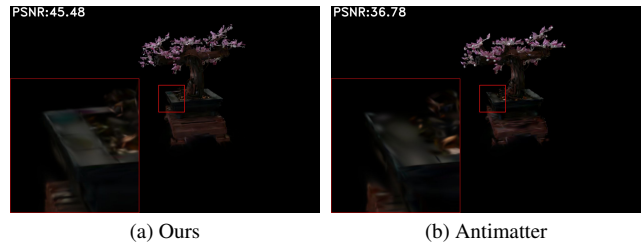

(a) Ours   (b) Antimatter

Figure 8. Comparison of our approach vs Antimatter on a segmentation of the bonsai tree from the *bonsai* scene. The zoomed-in crop shows that our approach is able to load in a more detailed version, which is also reflected by the masked PSNR.

a significantly better visualization with the same amount of splats. We have also shown that our approach can work in tandem with existing compression techniques to reduce the required bandwidth further. Our approach serves as the groundwork for the progressive streaming of 3DGS, which will play a vital role in bringing 3DGS content to end-users. In the future, we can explore other perspectives to optimize the streaming order by utilizing reinforcement learning, as shown in [32]. Other potential research directions include measuring user experience and application-guided progressive rendering, i.e., using object-level ordering to achieve adaptive streaming. When the domain evolves to work with larger and larger areas, progressive rendering and streaming will become more and more important.

6. Acknowledgements
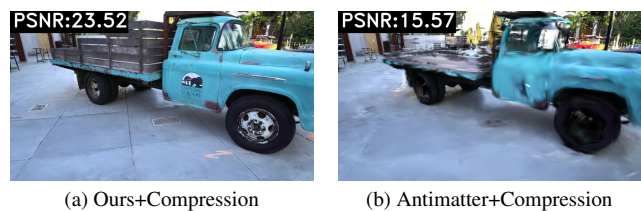
(a) Ours+Compression   (b) Antimatter+Compression

Figure 9. Comparison of the compressed versions of our approach (left) and Antimatter (right), both containing only 10% of the splats after compression.

References

[1] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *CVPR*, 2022. 5, 6

[2] Bo Chen, Zhisheng Yan, Bo Han, and Klara Nahrstedt. Nerfhub: A context-aware nerf serving framework for mobile immersive applications. In *Proceedings of the 22nd Annual International Conference on Mobile Systems, Applications and Services*, MOBISYS '24, page 85–98, New York, NY, USA, 2024. Association for Computing Machinery. 3

[3] Yun-Chun Chen, Vladimir G. Kim, Noam Aigerman, and Alec Jacobson. Neural progressive meshes, 2023. 3

[4] Junwoo Cho, Seungtae Nam, Daniel Rho, Jong Hwan Ko, and Eunbyung Park. Streamable neural fields. In *Computer Vision – ECCV 2022*, pages 595–612, Cham, 2022. Springer Nature Switzerland. 3

[5] 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. 4

[6] Jean-Philippe Farrugia, Luc Billaud, and Guillaume Lavoué. Adaptive streaming of 3d content for web-based virtual reality: an open-source prototype including several metrics and strategies. In *Proceedings of ACM Multimedia Systems Conference*, MMSys '23, June 2023. 3

[7] Thomas Forgione, Axel Carlier, Géraldine Morin, Wei Tsang Ooi, Vincent Charvillat, and Praveen Kumar Yadav. Dash for 3d networked virtual environment. In *Proceedings of the 26th ACM International Conference on Multimedia*, MM '18, page 1910–1918, New York, NY, USA, October 2018. Association for Computing Machinery. 3

[8] Sharath Girish, Kamal Gupta, and Abhinav Shrivastava. Eagles: Efficient accelerated 3d gaussians with lightweight encodings, 2024. 4

[9] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. 37(6):257:1–257:15, 2018. 6

[10] Hugues Hoppe. View-dependent refinement of progressive meshes. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, page 189–198, USA, 1997. ACM Press/Addison-Wesley Publishing Co. 3

[11] Hugues Hoppe. *Progressive Meshes*. Association for Computing Machinery, New York, NY, USA, 1 edition, 2023. 3

[12] Xinjue Hu, Yuxuan Pan, Yumei Wang, Lin Zhang, and Shervin Shirmohammadi. Multiple description coding for best-effort delivery of light field video using gnn-based compression. *IEEE Transactions on Multimedia*, 25:690–705, 2023. 3

[13] ISO/IEC 23009-1. Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats, 2022. 3

[14] ISO/IEC 23090-5. Information technology – Coded representation of immersive media – Part 5: Visual volumetric video-based coding (V3C) and video-based point cloud compression (V-PCC), 2023. 3

[15] Peter A. Kara, Aron Cserkaszky, Maria G. Martini, Attila Barsi, Laszlo Bokor, and Tibor Balogh. Evaluation of the concept of dynamic adaptive streaming of light field video. *IEEE Transactions on Broadcasting*, 64(2):407–421, 2018. 3

[16] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023. 2, 6

[17] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics*, 36(4), 2017. 5, 6

[18] Wouter LEMOINE and Maarten WIJNANTS. Progressive network streaming of textured meshes in the binary gltf 2.0 format. In *Proceedings of the 28th International ACM Conference on 3D Web Technology*, pages 1–11, New York, NY, USA, 2023. Association for Computing Machinery. The 28th International Conference on 3D Web Technology. 3

[19] Hendrik Lievens, Maarten Wijnants, Mike Vandersanden, Peter Quax, and Wim Lamotte. Adaptive web-based vr streaming of multi-lod 3d scenes via author-provided relevance scores. In *Proceedings of the IEEE Conference on Virtual Reality and 3D User Interfaces*, VR '21, pages 488–489, 2021. 3

[20] Hendrik Lievens, Maarten Wijnants, Brent Zoomers, Jeroen Put, Nick Michiels, Peter Quax, and Wim Lamotte. Adaptive streaming and rendering of static light fields in the web browser. In *2021 International Conference on 3D Immersion (IC3D)*, pages 1–8, 2021. 3

[21] Steven Liu, Xiuming Zhang, Zhoutong Zhang, Richard Zhang, Jun-Yan Zhu, and Bryan Russell. Editing conditional radiance fields. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021. 1

[22] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 1

[23] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022. 1

[24] KL Navaneet, Kossar Pourahmadi Meibodi, Soroush Abbasi Koohpayegani, and Hamed Pirsiavash. Compact3d: Compressing gaussian splat radiance field models with vector quantization. *arXiv preprint arXiv:2311.18159*, 2023. 4

[25] Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann. Compressed 3d gaussian splatting for accelerated novel view synthesis, 2023. 3, 6, 8

[26] 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), May 2024. 4

[27] Marie-Julie Rakotosaona, Fabian Manhardt, Diego Martin Arroyo, Michael Niemeyer, Abhijit Kundu, and Federico Tombari. Nerfmeshing: Distilling neural radiance fields into geometrically-accurate 3d meshes, 2023. 1

[28] Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollár, and Christoph Feichtenhofer. Sam 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*, 2024. 6

[29] Markus Schütz, Gottfried Mandlburger, Johannes Otepka, and Michael Wimmer. Progressive real-time rendering of one billion points without hierarchical acceleration structures. *Computer Graphics Forum*, 39:51–64, 07 2020. 3

[30] Carter Slocum, Jingwen Huang, and Jiasi Chen. Via: Visibility-aware web-based virtual reality. In *Proceedings of the 26th International Conference on 3D Web Technology*, Web3D '21. Association for Computing Machinery, 2021. 3

[31] Iraj Sodagar. The MPEG-DASH standard for multimedia streaming over the internet. *IEEE MultiMedia*, 18(4):62–67, April 2011. 3

[32] Naima Souane, Malika Bourenane, and Yassine Douga. Deep reinforcement learning-based approach for video streaming: Dynamic adaptive video streaming over http. *Applied Sciences*, 13(21), 2023. 8

[33] Jeroen Van Der Hooft, Tim Wauters, Filip De Turck, Christian Timmerer, and Hermann Hellwagner. Towards 6DoF HTTP Adaptive Streaming Through Point Cloud Compression. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19. Association for Computing Machinery, October 2019. 3

[34] Maarten Wijnants, Hendrik Lievens, Nick Michiels, Jeroen Put, Peter Quax, and Wim Lamotte. Standards-Compliant HTTP Adaptive Streaming of Static Light Fields. In *Proceedings of the 24th ACM Symposium on Virtual Reality Software and Technology*, VRST '18, 2018. 3

[35] Yuanbo Xiangli, Linning Xu, Xingang Pan, Nanxuan Zhao, Anyi Rao, Christian Theobalt, Bo Dai, and Dahua Lin. Bungeenerf: Progressive neural radiance field for extreme multi-scale scene rendering. In Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner, editors, *Computer Vision – ECCV 2022*, pages 106–122, Cham, 2022. Springer Nature Switzerland. 3

[36] Markos Zampoglou, Kostas Kapetanakis, Andreas Stamoulias, Athanasios G. Malamos, and Spyros Panagiotakis. Adaptive Streaming of Complex Web 3D Scenes based on the MPEG-DASH Standard. *Multimedia Tools and Applications*, 77(1):125–148, Jan 2018. 3