EVER: Exact Volumetric Ellipsoid Rendering for Real-time View Synthesis

Alexander Mai[1]
amai@ucsd.edu

Peter Hedman[2]
hedman@google.com

George Kopanas[2]
gkopanas@google.com

Dor Verbin[2]
dorverbin@google.com

David Futschik[2]
dfutschik@google.com

Qiangeng Xu[2]
qiangenx@google.com

Falko Kuester[1]
fkuester@ucsd.edu

Jonathan T. Barron[2]
barron@google.com

Yinda Zhang[2]
yindaz@google.com

[1]University of California, San Diego [2]Google

Figure 1. An overview of the quality benefits of our EVER technique. Left: On the Zip-NeRF dataset, our model produces sharper and more accurate renderings than 3DGS and successor splatting-based techniques. Middle: Because our method correctly blends primitive colors according to the physics of volume rendering, it produces fewer "foggy" artifacts than splatting. Right: Our method correctly blends primitive colors, which is not possible using a splatting regardless of how primitives are sorted (globally or ray-wise).

Abstract

We present Exact Volumetric Ellipsoid Rendering (EVER), a method for real-time differentiable emission-only volume rendering. Unlike recent rasterization based approach by 3D Gaussian Splatting (3DGS), our primitive based representation allows for exact volume rendering, rather than alpha compositing 3D Gaussian billboards. As such, unlike 3DGS our formulation does not suffer from popping artifacts and view dependent density, but still achieves frame rates of $\sim$ *30 FPS at 720p on an NVIDIA RTX4090. Since our approach is built upon ray tracing it enables effects such as defocus blur and camera distortion (e.g. such as from fisheye cameras), which are difficult to achieve by rasterization. We show that our method is more accurate with fewer blending issues than 3DGS and follow-up work on view-consistent rendering, especially on the challenging large-scale scenes from the Zip-NeRF dataset where it achieves sharpest results among real-time techniques.*

1. Introduction

The field of 3D reconstruction for novel view synthesis has explored a variety of scene representations: point based [22], surface based [26, 44, 47], and volume based [29, 31]. Since their introduction in NeRF [29], differentiable rendering of volumetric scene representations have become popular due to their ability to yield photorealistic 3D reconstructions. More recently, 3D Gaussian Splatting (3DGS) combined the speed of point based models with the differentiability of volume based representations by representing the scene as a collection of millions of Gaussian primitives that can be rendered via rasterization in real-time.

Unlike NeRF, 3DGS lacks a true volumetric density

field, and uses Gaussians to describe the *opacity* of the scene rather than of density. As such, 3DGS's scene representation may violate volumetric rendering, in that an anisotropic Gaussian primitive in 3DGS will appear to have the same opacity regardless of the viewing direction of the camera. This lack of a consistent underlying density field prohibits the use of various algorithms and regularizers (such as the distortion loss from Mip-NeRF360 [2]), but the biggest downside of this model is popping. Popping in 3DGS is due to two assumptions made by the model: that primitives do not overlap, and that (given a camera position) primitives can be sorted accurately using only their centers. These assumptions are almost always violated in practice, which causes the rendered image to change significantly as the camera moves due to the sort-order of primitives changing. This popping may not be noticeable when primitives are small, but describing a large scene using many small primitives requires a prohibitively large amount of memory.

In this work we build on the primitive based representation of 3DGS, but we define a physically accurate, constant density ellipsoid based representation. This formulation lets us compute the exact form of the volume rendering integral efficiently, which eliminates the inconsistencies of 3DGS while still maintaining real-time framerates. Because our system is built around ray tracing (rather than 3DGS's rasterization) it can straightforwardly model various optical effects like radial distortion lenses including fisheye and defocus blur. Our method guarantees 3D consistent real-time rendering while also improving the image quality of our 3DGS baselines. This is particularly pronounced in the challenging large-scale Zip-NeRF scenes [4] where our method matches the quality of state-of-the-art offline rendering methods.

2. Related Work

Neural Volume Rendering Neural Radiance Fields (NeRF) [29] introduced the paradigm of representing a scene as an emissive volume, using a neural network with position encoding that is rendered differentiably using quadrature [14, 28] and optimized using gradient descent. While the original formulation used a neural network, follow up work has used voxel grids [16, 39], hash grids [31], tri-planes [9], primitives [27], and points [46]. These papers all use numerical quadrature to approximately integrate the volume rendering equation.

While NeRF reconstructions are slow to render they can be converted into faster representations, such as triangle meshes [12, 35, 37, 38, 47], sparse volumes [15, 17, 36, 49], mesh-volume hybrids [40, 42, 45] and 3D Gaussians [32]. While these facilitate real-time rendering, creating them is a slow two-step process that first trains a NeRF and then converts it to a faster representation. In the experiments we compare with SMERF [15], the current state-of-the-art for real-time NeRF rendering.

Differentiable Point-based Rendering Like NeRF, 3D Gaussian Splatting (3DGS) [22] models the scene as a radiance field, but 3DGS represents radiance as a set of Gaussians that are rendered via splatting [52] while NeRF represents radiance using a field that is rendered via ray-marching. While 3DGS approximates volume rendering with view-independent opacity and view-dependent radiance, in practice it often yields highly accurate renderings, and these approximations allow it to be trained and rendered quickly, hence its popularity [11]. Since its inception, improvements have been made to 3DGS in terms of aliasing [18], camera models [30], heuristic densification [7, 8, 23, 41, 48, 50], and view-consistency (i.e. "popping").

Popping results from how 3DGS sorts Gaussians once per-frame using their mean, and it has been partly addressed by StopThePop [34] during rasterization and 3DGRT [30] during ray-tracing. However, StopThePop only approximately sorts per-ray, and both StopThePop and 3DGRT ignore overlap between the primitives. This introduces a blend order artifact shown in Fig. 7. One way of addressing popping is to compute the volumetric rendering equation with fewer (or zero) approximations. Concurrently with our work, a number of preprints attempt this [5, 13, 51] but they all achieve significantly slower performance and are subject to significant limitations regarding how much primitives may overlap, or use a different approximation [43]. By representing the scene with constant density primitives, our model is able to quickly and exactly compute the volumetric rendering equation, even with unlimited overlap.

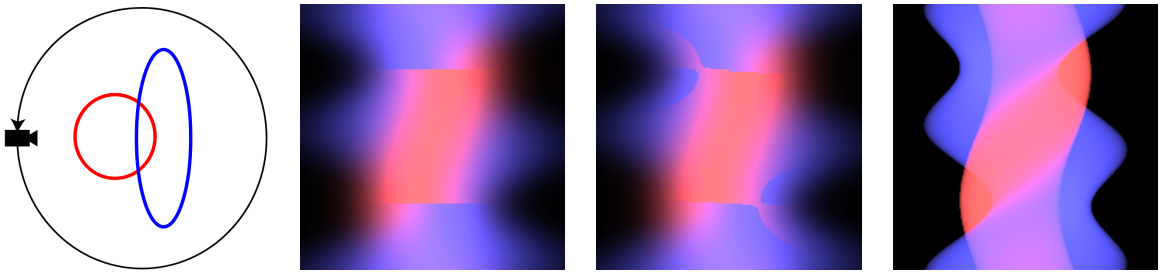
3. Motivation

Neural Radiance Fields A radiance field is comprised of two spatially-varying fields as a function of spatial coordinate $\mathbf{x}$: density $\sigma(\mathbf{x})$ and color $c(\mathbf{x}, \mathbf{d})$, where the color field may depend on viewing direction $\mathbf{d}$ in addition to $\mathbf{x}$. A ray with origin $\mathbf{o}$ and direction $\mathbf{d}$ is rendered by integrating these fields using standard volume rendering integral (also known as the radiative transfer equation [10]):

$$C = \int_0^\infty c_\mathbf{r}(t)\sigma_\mathbf{r}(t) \exp\left(-\int_0^t \sigma_\mathbf{r}(s)\, ds\right) dt\,, \quad (1)$$

where t is distance along a ray, and $\sigma_\mathbf{r}(t) = \sigma(\mathbf{o} + t\mathbf{d})$ and $c_\mathbf{r}(t) = c(\mathbf{o}+t\mathbf{d}, \mathbf{d})$ are the density and color along the ray, respectively.

The parameterization of the density and color fields can vary from small MLPs [29] to large hierarchies of hashes and grids grids [31]. Though this approach can produce highly realistic renderings, accurately integrating Equation 1 requires a large number of calls to the underlying



(a) Top Down View (b) 3DGS [22] (c) StopThePop [34] (d) Our Model

Figure 2. (a) Here we show a simple "flatland" scene containing two primitives (one red, one blue) with a camera orbiting them, viewed from above. We render this orbit using three different techniques, where each camera position yields a one-dimensional "image" (a scanline) which are stacked vertically to produce these epipolar plane image (EPI) visualizations. (b, c) The approximations made by approximate splatting-based techniques result in improper blending due to discontinuities, which are visible as horizontal lines across the EPI. In contrast, (d) our method's exact rendering yields a smooth EPI, with bands of purple from color blending.

field, which means that training and rendering may be slow.

3D Gaussian Splatting Like NeRF, 3DGS uses alpha compositing to render an image from volumetric primitives, but unlike NeRF it does not directly model a density field. Instead, a collection of Gaussians are projected onto frontoparallel billboards, multiplied by their opacities, sorted, and alpha composited together. Because this rendering process is data-parallel, 3DGS can be implemented to yield extremely high framerates. However, 3DGS is not 3D consistent; when the order in which the Gaussians are composed changes, the color abruptly changes as well. This lack of blending is visible as popping artifacts, as can be seen in Figure 2.

Constant Density Ellipsoids Our model lies in between NeRF and 3DGS: like NeRF we perform 3D consistent volume rendering of a radiance field, but like 3DGS we parameterize the scene using a collection of anisotropic primitives. Instead of Gaussians as primitives we use constant density ellipsoids, which we can volume render *exactly* without any numerical quadrature (and *efficiently* thanks to decades of progress in ray tracing). To highlight the difference between our method, 3DGS, StopThePop, and traditional NeRF, we render the same 3 colorful ellipsoids with all 4 methods on the right side of Figure 1. When the primitives are sorted per a view, the color between the primitives do not mix to produce purple, yellow, and blue green, but the appearance is smooth. When the primitives are sorted per a ray, the order changes throughout the image, causing lines to appear. Because our rendering model is exact, the rendering from our model exactly matches the ground truth.

4. Method

Like 3DGS, our method takes as input a set of posed images and a sparse point cloud. We optimize a mixture of ellipsoids (each with a constant density and color) to reproduce the appearance of the input images, where the initial positions of ellipsoids are determined by the input point cloud. We build upon the 3DGS framework, and reuse its Adaptive Density Control (ADC), with some modifications for handling density based primitives. We begin by describing how we are able to perform exact volume rendering of our representation, then describe how we convert opacity to density to allow the optimization of our density based primitives.

4.1. Exact Primitive-based Rendering

We use a simple primitive-based rendering model, where each primitive has a constant density and a constant (view-dependent) color. We choose our primitive's shape to be an ellipsoid, which, similar to the Gaussians in 3DGS, is fully characterized by a rotation and scale matrix. We illustrate these primitives and their intersections in Figure 3.

Selecting density and color models that are piecewise-

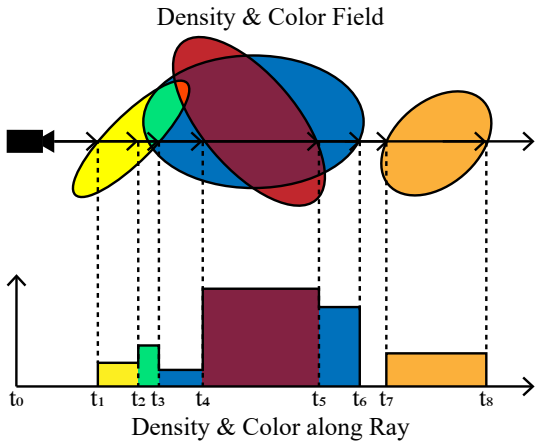


Figure 3. A visualization of our rendering procedure. The volume rendering equation can be integrated a field of constant density ellipsoids (depicted at the top) by tracing 9 rays. When the ray enters each primitive, the density along the ray increases. When it exits, the density drops back down a corresponding amount.

constant in 3D means that the density and color along any ray are also piecewise constant, and they can be written in the following step function form with coefficients σ_i, c_i respectively:

$$\sigma(t) = \sum_{i=1}^{N} \sigma_i \mathbb{1}[t_i \leq t < t_{i+1}], \tag{2}$$

$$\mathbf{c}(t) = \sum_{i=1}^{N} \mathbf{c}_i \mathbb{1}[t_i \leq t < t_{i+1}], \tag{3}$$

where $\mathbb{1}[\cdot]$ is an indicator function whose value is 1 when the condition in the brackets is met and 0 otherwise, and $\{t_i\}_{i=1}^{N}$ are the density discontinuities, which are found by intersecting the ray with the primitives. The density and color coefficients for the ith interval, $t \in [t_i, t_{i+1})$, can be determined by combining the contributions of each primitive along the ray:

$$\sigma_i = \sum_{k=1}^{i} \Delta\sigma_k, \quad \mathbf{c}_i = \frac{1}{\sigma_i} \sum_{k=1}^{i} \Delta\sigma_k \Delta\mathbf{c}_k, \tag{4}$$

where $\Delta\sigma_k$ and $\Delta\mathbf{c}_k$ are the changes in density and color at the kth intersection point of the ray with the primitives in the scene, respectively. Note that $\Delta\sigma_k$ is positive when entering a primitive and negative when exiting one. Now, the volume rendering integral from Equation 1 can be computed *exactly*:

$$C = \sum_{i=1}^{N} \mathbf{c}_i (1 - \exp(-\sigma_i \Delta t_i)) \prod_{j=1}^{i-1} \exp(-\sigma_j \Delta t_j) \tag{5}$$

Unsurprisingly, the expression in Equation 5 is identical to the quadrature rule derived by Max [28] that also makes a piecewise-constant approximation to density and color — though in our case this piecewise-constant property is an intentional design decision, not an approximation.

Though our scene representation looks superficially similar to that of 3DGS, it is different in two key ways: First, while 3DGS Gaussians are treated as 2D "billboards", our ellipsoids interact with each other so as to constitute a proper and consistent 3D radiance field. Second, while the 2D opacity profile of the primitives used in 3DGS always has a smooth Gaussian falloff, the 2D opacity profile of one of our ellipsoidal primitives can range from extremely smooth to a perfect step function in the limit of infinite density. See Figure 4 for examples of our primitive's opacity profiles for different density values. Our ellipsoidal primitives are also beneficial due to their computational efficiency in intersection calculations, low memory footprint, and smooth surface geometry.

4.2. Density Parameterization

As described in Section 4.1, our method assigns a density value to each primitive. Optimizing the density of all primitives directly presents a challenge: as the density of a primitive grows and its opacity approaches 1, the gradient used for updating the parameters of the primitive approaches 0 and would require special treatment [25]. Intuitively, this is because the outside of the primitive becomes opaque.

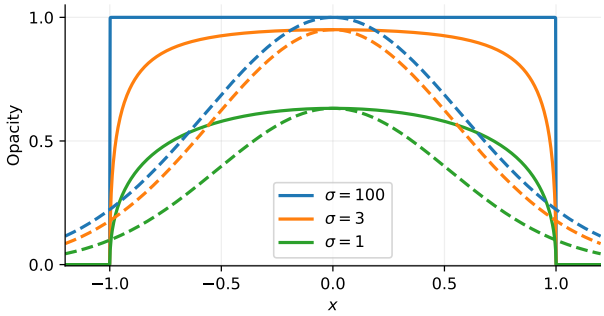


Figure 4. Here we show 1D screen-space slices of rendered opacity profiles for different rendering methods, to demonstrate one advantage of our density-based parameterization. We show 3 different peak opacities and their profiles, where solid lines represent our density based ellipsoid primitives and dashed lines represent 3DGS. For our model we show densities $\sigma \in \{1, 3, 100\}$, while for 3DGS we compute the corresponding peak opacity such that each slice has the same maximum opacity as our model. While Gaussians always have smooth opacity profiles that limit their ability to reproduce edges in image space, our opacity profiles can be smooth ($\sigma = 1$) or sharp ($\sigma = 100$).

We avoid this problem by optimizing over a parameter α and using the following density during rendering:

$$\sigma(\alpha) = -\frac{\log(1 - 0.99 \cdot \alpha)}{\min(s_x, s_y, s_z)}. \tag{6}$$

With this, a primitive with density $\sigma(\alpha)$ when viewed along it's shortest axis will have a peak opacity equal to α.

5. Implementation

Our model was implemented using PyTorch, CUDA, OptiX [33], and Slang [20]. We use OptiX to ray trace through the primitives and to provide a per-ray sorting of distances. During training we rebuild our BVH at every step, which takes tens of milliseconds. All shaders are written in Slang [20], a language that supports automatic differentiation with the Slang.D extension [1]. We use adjoint rendering to propagate the gradient, which means we skip storage of all function values and simply recompute them during the backwards pass. To optimize the representation, we use our differentiable renderer in the 3DGS [22] codebase, and make a few adjustments to handle density based primitives (Section 5.2 and Appendix C).

5.1. Ray Tracing (BVHs) for Sorting

We use ray tracing to sort primitives since it offers more flexibility than rasterization. Ray tracing allows for effects

(a) Fisheye training and rendering

(b) Shallow depth of field rendering

Figure 5. A depiction of two benefits of ray tracing.

like fisheye projection and defocus blur (see Figure 5) as well as random pixel offsets during training, which we find improve performance.

We use the recent approach of 3DGRT [30], which uses a BVH to accelerate an exact per ray sort of the primitives to realtime speeds. We start by providing the Nvidia OptiX [33] framework with an Axis-Aligned Bounding Box (AABB) and an intersection function that indicates where along the ray it intersects each ellipsoid. A tight AABB for an ellipsoid is straightforward to compute, and we interesect a ray with an anisotropic ellipsoid first making it isotropic through shifting, rotating, then scaling the ray origin. Then, then we compute the stable isotropic ray/sphere intersection described Haines et al [19].

BVH efficiency depends strongly on how often the bounding boxes of primitives overlap. Rotated anisotropic primitives tend to have large axis-aligned bounding boxes that slow down rendering. To ameliorate this, we impose a loss during training to discourage overly anisotropic primitives:

$$\text{stopgrad}(1-\alpha)(\max(s_x, s_y, s_z) - \min(s_x, s_y, s_z)), \quad (7)$$

where $\alpha \in [0, 1]$ is opacity and $\mathbf{s} \in \mathbb{R}^3$ is axis-aligned scaled for each primitive. This regularizer is applied only to primitives that are visible in the current batch.

5.2. Changes to Adaptive Density Control

The Adaptive Density Control (ADC) used by 3DGS is critical to its success. During training 3DGS uses a variety of strategies for periodically cloning, splitting, and pruning 3D Gaussians. This is essential to avoid local minima by creating primitives where they are necessary, and removing primitives where they are invisible. Though these heuristics were developed specifically for 3DGS, they work well for our method with some minor necessary adjustments.

To decide where to create primitives, 3DGS accumulates the xy gradients of each primitive, then splits or clones them if they are above a certain threshold every n steps. The spatial gradients of our model tend to be smaller, so the thresholds used by these splitting and cloning heuristics must be modified (splitting=2.5×10^{-7}, cloning=0.1). We use a similar splitting method as 3DGS: the primitive size is reduced and the new position is perturbed by some random amount sampled from a normal distribution, but we additionally we also divide the density in half, similar to [7].

Finally, as a consequence of Section 4.2, we add an additional splitting condition: We split primitives if the opacity is near 1 across the entire primitive when viewed from the major axis to see if the gradient has vanished using the following test:

$$0.99 < 1 - \exp(\sigma \max(s_x, s_y, s_z)) \quad (8)$$

6. Results

We evaluate on the 9 scenes introduced in Mip-NeRF 360 [3] and the 4 scenes introduced in Zip-NeRF [4]. We use the same parameters across all scenes, except one: the size threshold for splitting and cloning is adjusted for large scenes to avoid total densification failure for both 3DGS and our method. For the *alameda* scene, we utilize the exposure trick introduced in [15] to allow for evaluation on datasets with multiple exposures. We also re-tuned 3DGS to work slightly better than the tune done by Duckworth et al. [15] on the Zip-NeRF.

There have been many follow ups to 3DGS, each introducing different kinds of improvements, so we limit the scope of our comparison to best isolate our contribution. There are many recent approaches for densification [7, 8, 41, 48, 50], but we chose to change densification only in ways required to handle density-based parameterization. It is likely that recent progress on improved 3DGS heuristics may similarly improve the performance of our model. There are also many different ways to change rendering [5, 18, 21, 23, 34, 51], so we chose to only compare rendering methods that specifically address popping.

6.1. Reconstruction Quality

We measure image quality through the standard image metrics PSNR, SSIM and LPIPS. Note however that we recom-

	Mip-NeRF360				Zip-NeRF			
	PSNR ↑	SSIM ↑	LPIPS ↓	FPS ↑	PSNR ↑	SSIM ↑	LPIPS ↓	FPS ↑
3DGS [22]	27.48	.816	.257	224	25.84	.842	.418	559
StopThePop [34]	27.33	.816	.251	180	25.92	.819	.411	403
3DGRT [30]	27.20	.818	.248	78†	-	-	-	-
SMERF [15]	27.99	.818	.238	204†	27.28	.830	.389	217†
Our model	27.51	.825	.233	36	26.60	.845	.368	24
ZipNeRF [4]	28.54	.828	.219	0.5†	27.37	.836	.364	0.5†

Table 1. Results on the Mip-NeRF360 [2] and Zip-NeRF [4] datasets. The softness of the Gaussian primitives helps with PSNR, which is a metric that prefers blurrier images because of imprecise image alignment. However, we can see from the other metrics that our method works as well as the latest 3DGS methods on the Mip-NeRF 360 datasets, and outperforms them on the Zip-NeRF datasets. Note that, unlike the prior works, our LPIPS values have been recomputed to properly normalize images to $[-1, 1]$. Zip-NeRF outperforms all other models on all metrics, but it is prohibitively slow to render for real-time applications (~ 0.5 FPS). FPS results are computed at test resolution, and results with a † were not recomputed and use different high end GPUs.

pute LPIPS for all methods to ensure they use the same back-end (VGG) and input color range ($[-1, 1]$). We compare our method with 3DGS [22] and Zip-NeRF [4], the current offline rendering state-of-the-art for these datasets. We also compare with StopThePop since it improves view-consistency for 3DGS, 3DGRT [30] because our work uses their ray tracing algorithm, and we SMERF [15] because even though it is a distillation based method, it is a comparable real-time 3D consistent volume rendering method.

Quantitatively, we outperform the other 3DGS-based methods across the board, and set a new state of the art in the Zip-NeRF dataset in terms of sharpness as measured by LPIPS and SSIM — even when compared to offline and distillation-based methods. Qualitatively, we notice two main factors: improved blending, and improved sharpness. The combination of more flexible primitive appearance, combined with color blending, allows our method to represent gradients better, which can be seen in Figure 4. This effect can also be seen when comparing how these methods model light falling on flat walls, as shown by the bottom two rows of Figure 6. While our method is able to reproduce the smooth image gradients in the shadowed areas, all of the Gaussian based methods struggle (similarly to how our model struggles when our blending model is ablated, as seen in Figure 8). The improved sharpness is more apparent on the larger and more difficult Zip-NeRF dataset, both in the metrics and in qualitative examples (see the second two rows of Figure 6).

6.2. Popping

As established in Figure 2, both 3DGS and StopThePop exhibit artifacts due to their view-dependent density. We show what these artifacts look like using epipolar plane images [6] in Figure 7. Because our rendering model is exact, and because the volume rendering integral is a continuous function of ray direction, our renderings exhibit no popping. Please see the supplemental video for a clear visualization.

6.3. Performance

At 720p, we achieve average framerates of 36 FPS on the mip-NeRF 360 outdoor scenes (min=33), 66 FPS on the mip-NeRF 360 indoor scenes (min=57), and 30 FPS on the Zip-NeRF scenes (min=25) on an NVIDIA RTX4090. The test set resolution is high on a couple of the Zip-NeRF and Mip-NeRF360 scenes, which causes the drop in FPS in Table 1. Training takes around 1-2 hours, also on an NVIDIA RTX4090. The majority of the time is spent tracing through the scene for rendering, which takes 20-71 ms, depending on the BVH. For back propagation, 16 ms are spent storing intersections, and 13-20 ms are spent loading the primitives and atomically adding up the gradients. The BVH is rebuilt every training iteration, which takes 2-20 ms.

6.4. Ablations

To see if it was possible to train our model using ray tracing, then render using splatting, we implemented a splatting version of our 3D ellipsoids, labeled "Splatted". The result can be seen in Figure 8. We also tested what would happen if we sorted the primitives per ray like StopThePop , which is labeled "No Mixing". Both of these resulted in visual degradation where the method had been mixing primitive

	PSNR ↑	SSIM ↑	LPIPS ↓	FPS ↑
Splatted	26.63	.845	.329	-
No Mixing	26.79	.849	.305	-
3DGS	27.02	.853	.301	-
3DGS + Our Changes	26.94	.832	.323	-
No Anisotropic	27.25	.865	.272	27.3
No Points	27.19	.863	.277	41.2
No Rand Center	27.08	.859	.278	41.4
Ours	27.17	.862	.277	41.9

Table 2. An ablation study reporting average performance on the *bicycle* and *counter* scenes from MipNeRF 360 [2] and the *berlin* scene from ZipNeRF [4].

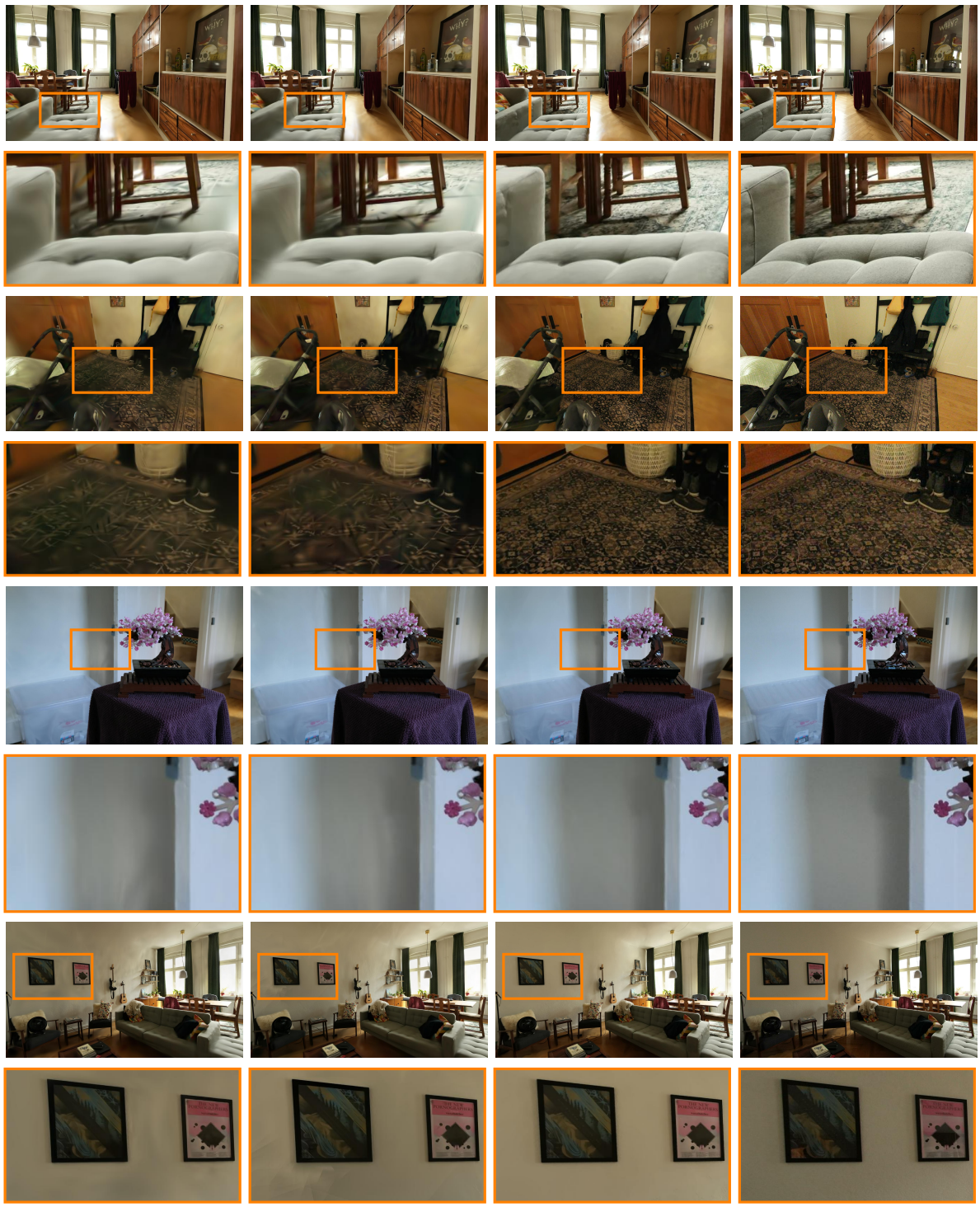


Figure 6. Visual comparison of our method to other 3DGS-based methods on the Mip-NeRF [2] and Zip-NeRF [4] datasets. Rows 1 and 2 show regions where our results are sharper. Rows 3 and 4 show how proper blending of primitives helps with handling lighting on textureless surfaces.

Figure 7. Here we show epipolar plane images [6] to visualize popping in 3DGS [22] and StopThePop [34]. We recorded sequences of images while rotating the camera in the *berlin* [4] (left) and *train* [24] (right) scenes. The region in the middle are the epipolar plane images, with green arrows highlighting discontinuities in the color caused by popping/blend order artifacts. Our method lacks these horizontal discontinuities. Contrast and brightness have been boosted here for visibility's sake. See the supplemental video for more.

Figure 8. To visualize the difference between rendering methods, we optimize our ellipsoids for our method, then visualize how different rendering methods affect the final image. Both "Splatted" and "No Mixing" assume that Gaussians do not overlap, with the former ordering them globally for the image (like 3DGS) , and the latter sorting them independently per pixel (like StopThePop). Both approaches cause hard edges to appear in areas where different primitives were blended to produce a color gradient.

colors together, which can be seen on the fridge and candle. These ablations also perform worse when trained for their specific rendering styles, as can be seen in Table 2. "Splatting" performs worse than "No Mixing", which does not match the comparison between 3DGS and StopThePop, both of which have similar performance.

To ablate the effect of our densification changes, along with the other changes, we use the 3DGS renderer with the other minor changes we've made to densification, which we label "3DGS+Our Changes". These changes do not help 3DGS because they are aimed at handling density primitives. Finally, we run ablations on each of the changes we performed. "No anisotropic" refers to ablating anisotropic loss, which results in slightly increased quality, but much lower framerate. "No points" refers to no additional points added using the inverse contraction."No Rand Centers" refers to ablating randomizing the pixel centers, which helps with thin structures on the bicycle scene.

7. Conclusion

We have presented Exact Volumetric Ellipsoid Rendering (EVER), which bridges the gap between slow but accurate radiance field methods such as Zip-NeRF, and fast but inaccurate radiance field methods like 3DGS. By exactly ray-tracing a volumetric collection of constant density ellipsoids, EVER is able to produce high-quality renderings that are guaranteed to be 3D-consistent and therefore exhibit no popping, and does so at 30 FPS @ 720p on a single consumer-level GPU. By combining the flexibility of ray-tracing with the speed of primitive-based radiance field methods, EVER enables highly-flexible, high-quality, real-time, radiance field reconstruction.

Acknowledgements We would like to thank Delio Vicini, Stephan Garbin and Ben Lee for helpful discussions. AM is funded in part by the USACE Engineer Research and Development Center Cooperative Agreement W9132T-22-2-0014.

References

[1] Sai Bangaru, Lifan Wu, Tzu-Mao Li, Jacob Munkberg, Gilbert Bernstein, Jonathan Ragan-Kelley, Fredo Durand, Aaron Lefohn, and Yong He. SLANG.D: Fast, Modular and Differentiable Shader Programming. *SIGGRAPH Asia*, 2023. 4

[2] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields. *ICCV*, 2021. 2, 6, 7

[3] 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, 1

[4] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Zip-NeRF: Anti-Aliased Grid-Based Neural Radiance Fields. *ICCV*, 2023. 2, 5, 6, 7, 8

[5] Hugo Blanc, Jean-Emmanuel Deschaud, and Alexis Paljic. Raygauss: Volumetric gaussian-based ray casting for photorealistic novel view synthesis, 2024. 2, 5

[6] Robert C Bolles, H Harlyn Baker, and David H Marimont. Epipolar-plane image analysis: An approach to determining structure from motion. *IJCV*, 1987. 6, 8

[7] Samuel Rota Bulò, Lorenzo Porzi, and Peter Kontschieder. Revising densification in gaussian splatting. *arXiv:2404.06109*, 2024. 2, 5

[8] Junli Cao, Vidit Goel, Chaoyang Wang, Anil Kag, Ju Hu, Sergei Korolev, Chenfanfu Jiang, Sergey Tulyakov, and Jian Ren. Lightweight predictive 3d gaussian splats. *arXiv:2406.19434*, 2024. 2, 5

[9] Eric R Chan, Connor Z Lin, Matthew A Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas J Guibas, Jonathan Tremblay, Sameh Khamis, et al. Efficient geometry-aware 3d generative adversarial networks. *CVPR*, 2022. 2

[10] Subrahmanyan Chandrasekhar. *Radiative transfer*. Courier Corporation, 1960. 2

[11] Guikun Chen and Wenguan Wang. A Survey on 3D Gaussian Splatting. *arXiv:2401.03890*, 2024. 2

[12] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. MobileNeRF: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. *CVPR*, 2023. 2

[13] Jorge Condor, Sebastien Speierer, Lukas Bode, Aljaz Bozic, Simon Green, Piotr Didyk, and Adrian Jarabo. Volumetric primitives for modeling and rendering scattering and emissive media. *arXiv:2405.15425*, 2024. 2

[14] Robert A Drebin, Loren Carpenter, and Pat Hanrahan. Volume rendering. *SIGGRAPH*, 1988. 2

[15] Daniel Duckworth, Peter Hedman, Christian Reiser, Peter Zhizhin, Jean-François Thibert, Mario Lučić, Richard Szeliski, and Jonathan T Barron. SMERF: Streamable Memory Efficient Radiance Fields for Real-Time Large-Scene Exploration. *ACM Transactions on Graphics*, 2024. 2, 5, 6

[16] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. *CVPR*, 2022. 2

[17] Stephan J. Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. FastNeRF: High-fidelity neural rendering at 200fps. *ICCV*, 2021. 2

[18] Andreas Geiger, Torsten Sattler, Binbin Huang, Anpei Chen, and Zehao Yu. Mip-Splatting: Alias-free 3D Gaussian Splatting. *CVPR*, 2024. 2, 5

[19] Eric Haines, Johannes Günther, and Tomas Akenine-Möller. *Precision Improvements for Ray/Sphere Intersection*. 2019. 5

[20] Yong He, Kayvon Fatahalian, and Tim Foley. Slang: language mechanisms for extensible real-time shading systems. *ACM Transactions on Graphics*, 2018. 4

[21] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2d gaussian splatting for geometrically accurate radiance fields. *SIGGRAPH*, 2024. 5

[22] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 2023. 1, 2, 3, 4, 6, 8

[23] Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Weiwei Sun, Jeff Tseng, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. 3d gaussian splatting as markov chain monte carlo. *arXiv:2404.09591*, 2024. 2, 5

[24] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics*, 2017. 8

[25] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Transactions on Graphics (TOG)*, 37(6):1–11, 2018. 4

[26] Zhaoshuo Li, Thomas Müller, Alex Evans, Russell H Taylor, Mathias Unberath, Ming-Yu Liu, and Chen-Hsuan Lin. Neuralangelo: High-fidelity neural surface reconstruction. *CVPR*, 2023. 1

[27] Stephen Lombardi, Tomas Simon, Gabriel Schwartz, Michael Zollhoefer, Yaser Sheikh, and Jason Saragih. Mixture of volumetric primitives for efficient neural rendering. *ACM Transactions on Graphics (ToG)*, 40(4):1–13, 2021. 2

[28] Nelson Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995. 2, 4

[29] 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. *ECCV*, 2020. 1, 2

[30] Nicolas Moenne-Loccoz, Ashkan Mirzaei, Or Perel, Riccardo de Lutio, Janick Martinez Esturo, Gavriel State, Sanja Fidler, Nicholas Sharp, and Zan Gojcic. 3d gaussian ray tracing: Fast tracing of particle scenes. *arXiv:2407.07090*, 2024. 2, 5, 6

[31] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics*, 2022. 1, 2

[32] 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. *arXiv:2403.13806*, 2024. 2

[33] Steven G Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, et al. Optix: a general purpose ray tracing engine. *ACM Transactions on Graphics*, 2010. 4, 5

[34] Lukas Radl, Michael Steiner, Mathias Parger, Alexander Weinrauch, Bernhard Kerbl, and Markus Steinberger. Stopthepop: Sorted gaussian splatting for view-consistent real-time rendering. *arXiv:2402.00525*, 2024. 2, 3, 5, 6, 8

[35] 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. *3DV*, 2023. 2

[36] Christian Reiser, Rick Szeliski, Dor Verbin, Pratul Srinivasan, Ben Mildenhall, Andreas Geiger, Jon Barron, and Peter Hedman. Merf: Memory-efficient radiance fields for real-time view synthesis in unbounded scenes. *ACM Transactions on Graphics (TOG)*, 42(4):1–12, 2023. 2

[37] Christian Reiser, Stephan Garbin, Pratul P. Srinivasan, Dor Verbin, Richard Szeliski, Ben Mildenhall, Jonathan T. Barron, Peter Hedman, and Andreas Geiger. Binary opacity grids: Capturing fine geometric detail for mesh-based view synthesis. *SIGGRAPH*, 2024. 2

[38] Sara Rojas, Jesus Zarzar, Juan C. Pérez, Artsiom Sanakoyeu, Ali Thabet, Albert Pumarola, and Bernard Ghanem. Rerend: Real-time rendering of nerfs across devices. *ICCV*, 2023. 2

[39] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. 2022 ieee. *CVPR*, 2021. 2

[40] Haithem Turki, Vasu Agrawal, Samuel Rota Bulò, Lorenzo Porzi, Peter Kontschieder, Deva Ramanan, Michael Zollhöfer, and Christian Richardt. HybridNeRF: Efficient neural rendering via adaptive volumetric surfaces. *CVPR*, 2024. 2

[41] Evangelos Ververas, Rolandos Alexandros Potamias, Jifei Song, Jiankang Deng, and Stefanos Zafeiriou. Sags: Structure-aware 3d gaussian splatting. *arXiv:2404.19149*, 2024. 2, 5

[42] Ziyu Wan, Christian Richardt, Aljaž Božič, Chao Li, Vijay Rengarajan, Seonghyeon Nam, Xiaoyu Xiang, Tuotuo Li, Bo Zhu, Rakesh Ranjan, and Jing Liao. Learning neural duplex radiance fields for real-time view synthesis. *CVPR*, 2023. 2

[43] Angtian Wang, Peng Wang, Jian Sun, Adam Kortylewski, and Alan Yuille. Voge: a differentiable volume renderer using gaussian ellipsoids for analysis-by-synthesis. *arXiv preprint arXiv:2205.15401*, 2022. 2

[44] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *arXiv:2106.10689*, 2021. 1

[45] Zian Wang, Tianchang Shen, Merlin Nimier-David, Nicholas Sharp, Jun Gao, Alexander Keller, Sanja Fidler, Thomas Müller, and Zan Gojcic. Adaptive shells for efficient neural radiance field rendering. *SIGGRAPH Asia*, 2023. 2

[46] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-nerf: Point-based neural radiance fields. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5438–5448, 2022. 2

[47] Lior Yariv, Peter Hedman, Christian Reiser, Dor Verbin, Pratul P Srinivasan, Richard Szeliski, Jonathan T Barron, and Ben Mildenhall. Bakedsdf: Meshing neural sdfs for real-time view synthesis. *SIGGRAPH*, 2023. 1, 2

[48] Zongxin Ye, Wenyu Li, Sidun Liu, Peng Qiao, and Yong Dou. Absgs: Recovering fine details in 3d gaussian splatting. *ACM Multimedia 2024*, 2024. 2, 5

[49] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. 2021. 2

[50] Zehao Yu, Torsten Sattler, and Andreas Geiger. Gaussian opacity fields: Efficient and compact surface reconstruction in unbounded scenes. *arXiv:2404.10772*, 2024. 2, 5

[51] Yang Zhou, Songyin Wu, and Ling-Qi Yan. Unified gaussian primitives for scene representation and rendering. *arXiv:2406.09733*, 2024. 2, 5

[52] Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. Ewa volume splatting. *Visualization*, 2001. 2

EVER: Exact Volumetric Ellipsoid Rendering for Real-time View Synthesis

Supplementary Material

A. Backpropagation

We use an adjoint rendering approach, starting from the last state of each ray, which we store, and reconstruct each ray state backwards using $p^{-1}(x, i)$. We can then use this reconstructed state to backpropagate the error to each state, then propagate this error to each primitive. The gradient with respect to mean, scale, and orientation, all come from the derivative of the ray-ellipsoid intersection function. To retrieve the list of primitives intersected, we store the list of intersections on the forward pass. Since rays tend to terminate before 300 total intersections, this turns out to be relatively cheap, at 1.2 KB per a ray. Although we experimented with using ray tracing to retrieve the list of surfaces in reverse order, we found the instability too high.

B. Full Results

PSNR	berlin	nyc	alameda	london
3DGS	26.83	26.90	24.14	25.48
StopThePop	26.81	27.14	24.12	25.61
SMERF	28.52	28.21	25.35	27.05
Our model	27.24	27.93	24.72	26.49
ZipNeRF	28.59	28.42	25.41	27.06
SSIM	berlin	nyc	alameda	london
3DGS	.899	.861	.776	.830
StopThePop	.885	.844	.748	.801
SMERF	.887	.844	.758	.829
Our model	.900	.863	.779	.837
ZipNeRF	.891	.850	.767	.835
LPIPS	berlin	nyc	alameda	london
3DGS	.406	.380	.441	.446
StopThePop	.402	.373	.433	.438
SMERF	.391	.361	.416	.390
Our model	.371	.337	.389	.374
ZipNeRF	.378	.331	.387	.360

Table 3. Full results for Zip-NeRF dataset

C. Hyperparameters, Etc

For learning rates (LR), we change the opacity LR to 0.0125, the initial position LR to 4×10^{-5} and the final position LR to 4×10^{-7}. We change the parameter known as "percent dense" in 3DGS to 0.001785, which controls the size threshold above which primitives are split instead of cloned. Speaking of splitting and cloning, we perform this every 200 iterations, instead of 100, and set the splitting gradient threshold to 2.5×10^{-7} and the clone gradient threshold to 0.1. We also stop splitting and cloning at 7 million primitives, or at 16000 iterations, which ever comes first, and start at 1500 iterations.

For color, we apply softplus ($\beta = 10$) to the output of the spherical harmonics, which we find avoids certain local minima where primitives get lock into a color, which happens with $\max(0, x)$ in 3DGS. Speaking of color, we increase the spherical harmonic degree every 2000 iterations, instead of 1000. One final note: we set the max primitive size to 25 units, which seems to help performance.

C.1. Inverse Contraction Initialization

To help initialize the primitives in a scene, we supplement the SfM initialization with 10000 additional primitives. We generate these primitives by sampling their means uniformly from a radius-2 sphere. The radius is set to a constant value based the max radius at which the spheres could be packed into the radius-2 sphere, and colors are set to a constant value of 0.5. These primitives are then transformed by "uncontracting" the resulting means and covariances using the inverse of the contraction used in mip-NeRF 360 [3]. We found that highly anisotropic primitives at initialization can cause issues, so we scale the primitives to be isotropic.

To review, the mip-NeRF 360 contraction function $\mathcal{C}$ that maps from a 3D coordinate in Euclidean space $\mathbf{x}$ to a 3D coordinate in contracted space $\mathbf{z}$ is:

$$\mathcal{C}(\mathbf{x}) = \mathbf{x} \cdot \frac{2\sqrt{\max(1, ||\mathbf{x}||^2)} - 1}{\max(1, ||\mathbf{x}||^2)} \tag{9}$$

The inverse of $\mathcal{C}(\mathbf{x})$ can be defined straightforwardly:

$$\mathcal{C}^{-1}(\mathbf{z}) = \frac{\mathbf{z}}{\sqrt{\max(1, ||\mathbf{z}||^2)}(2 - \min(2, \sqrt{\max(1, ||\mathbf{z}||^2)}))} \tag{10}$$

To apply this inverse contraction to a Gaussian instead of a point, we use the same Kalman-esque approach as was used in mip-NeRF 360: we linearize the contraction around $\mathbf{z}$ into a Jacobian-vector product, which we apply twice to the input covariance matrix.

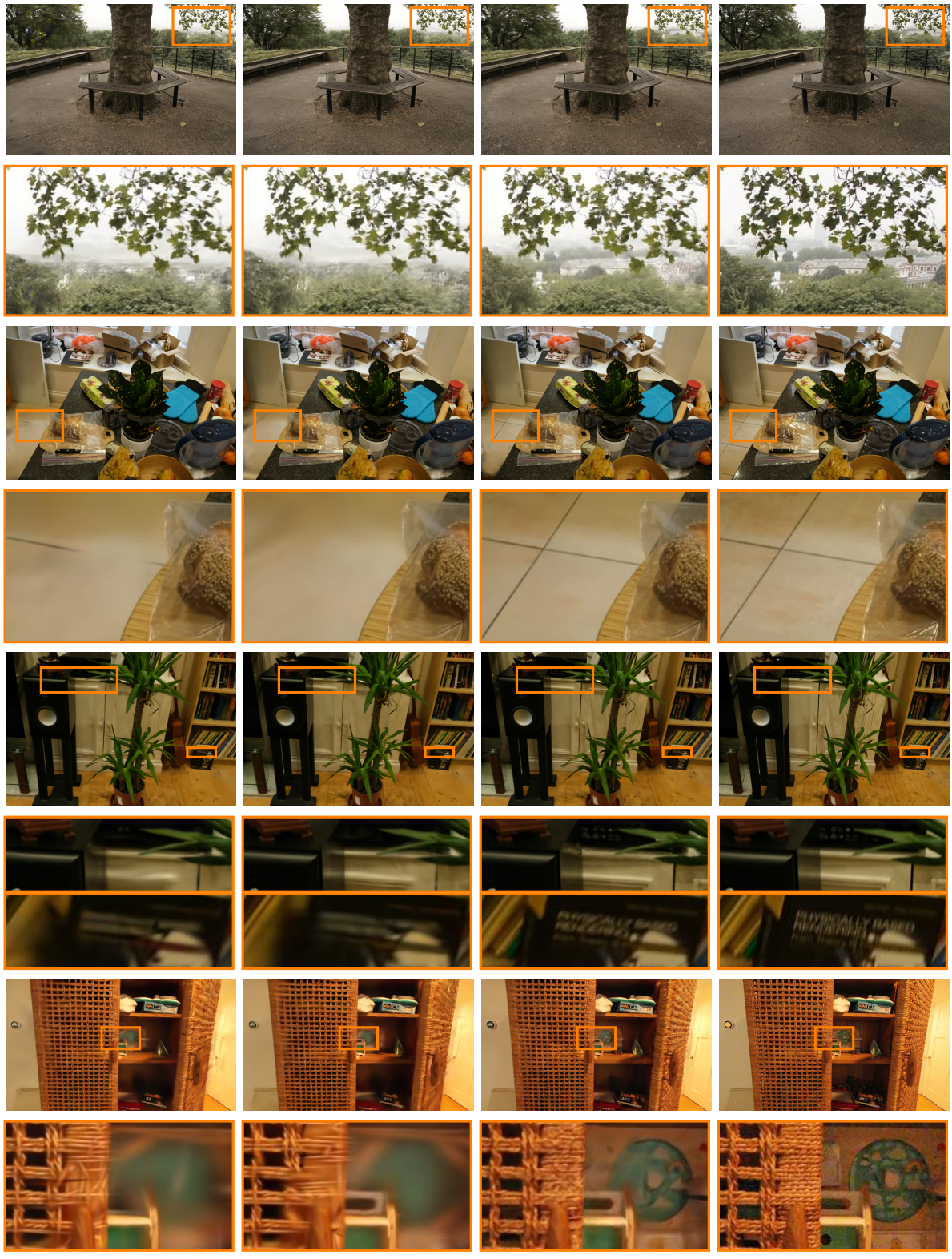


Figure 9. Additional visual comparison of our method to other Gaussian based methods on the Mip-NeRF 360 dataset [4].

PSNR	bicycle	flowers	garden	stump	treehill	room	counter	kitchen	bonsai
3DGS	25.24	21.55	27.38	26.56	22.43	31.53	29.00	31.45	32.21
StopThePop	25.23	21.62	27.33	26.65	22.44	30.91	28.79	31.13	31.85
3DGRT	25.13	21.58	26.99	26.57	22.40	30.92	28.78	30.60	31.85
SMERF	25.58	22.24	27.66	27.19	23.93	31.38	29.02	31.68	33.19
Our model	25.34	21.70	27.46	26.41	22.74	31.39	28.91	31.36	32.24
ZipNeRF	25.80	22.40	28.20	27.55	23.89	32.65	29.38	32.50	34.46
SSIM	bicycle	flowers	garden	stump	treehill	room	counter	kitchen	bonsai
3DGS	.766	.606	.866	.771	.633	.919	.909	.928	.942
StopThePop	.768	.607	.866	.775	.635	.919	.907	.927	.941
3DGRT	.770	.624	.858	.779	.636	.917	.908	.924	.942
SMERF	.760	.626	.844	.784	.682	.918	.892	.916	.941
Our model	.776	.639	.869	.781	.656	.922	.910	.926	.943
ZipNeRF	.769	.642	.860	.800	.681	.925	.902	.928	.949
LPIPS	bicycle	flowers	garden	stump	treehill	room	counter	kitchen	bonsai
3DGS	.240	.367	.123	.251	.376	.287	.258	.155	.252
StopThePop	.233	.362	.120	.244	.366	.281	.253	.154	.249
3DGRT	.226	.335	.134	.243	.364	.280	.248	.156	.242
SMERF	.239	.317	.147	.243	.302	.259	.256	.155	.222
Our model	.220	.307	.120	.230	.318	.275	.240	.155	.236
ZipNeRF	.228	.309	.127	.236	.281	.238	.223	.134	.196

Table 4. Full results for Mip-NeRF 360 dataset