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[3] fkuester@ucsd.edu
Jonathan T. Barron[2] barron@google.com
Yinda Zhang[2] yindaz@google.com

[1]University of California, San Diego [2]Google [*]work done while at Google

Figure 1. An overview of the quality benefits of our EVER technique. Left: On the Zip-NeRF dataset, our model produces the sharpest results ever. Middle: Our method is able to blend primitive colors together to reproduce light and shadow better than other 3DGS-based methods. Right: The way our approach blends colors is physically accurate over any number of primitives, matching the ground truth computed via quadrature.

Abstract

We present Exact Volumetric Ellipsoid Rendering (EVER), a method for real-time 3D reconstruction. EVER accurately blends an unlimited number of overlapping primitives together in 3D space, eliminating the popping artifacts that 3D Gaussian Splatting (3DGS) and other related methods exhibit. EVER represents a radiance field as a set of constant-density volumetric ellipsoids, which are raytraced by intersecting each primitive twice (once upon ray entrance and another on ray exit) and accumulating the derivatives of the densities and colors along the ray. Because EVER is built around ray tracing, it also enables effects such as defocus blur and fish-eye camera distortion, while still achieving frame rates of $\sim$30 *FPS at 720p on an NVIDIA RTX4090. We show that our method is more accurate on the challenging large-scale scenes from the Zip-NeRF dataset, where it achieves state of the art SSIM, even higher than Zip-NeRF.*

1. Introduction

The field of 3D reconstruction for novel view synthesis has explored a variety of scene representations: point based [23], surface based [28, 46, 49], and volume based [31, 33]. Since their introduction in NeRF [31], 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 isn't volumetric rendering, as 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 introduce a method that allows for the physically accurate blending of an infinite number of overlapping number of primitives, specifically constant density ellipsoids. We implement this blending method in a ray-tracing framework, which allows us to model various optical effects like radial distortion lenses including fisheye and defocus blur in a straightforward way, all at real-time framerates. 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) [31] 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, 30] and optimized using gradient descent. While the original formulation used a neural network, follow up work has used voxel grids [16, 41], hash grids [33], tri-planes [9], primitives [29], and points [48]. 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, 37, 39, 40, 49], sparse volumes [15, 17, 38, 51], mesh-volume hybrids [42, 44, 47] and 3D Gaussians [34]. 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) [23] models the scene as a radiance field, but 3DGS represents radiance as a set of Gaussians that are rendered via splatting [54] 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 [32], heuristic densification [7, 8, 24, 43, 50, 52], 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 [36] during rasterization and 3DGRT [32] 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, 53] but they all achieve significantly slower performance and are subject to significant limitations regarding how much primitives may overlap, or use a different approximation [45]. 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

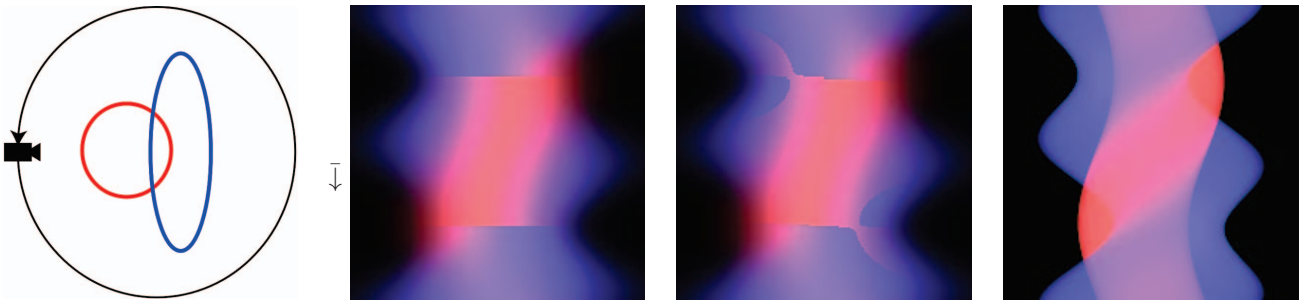
Neural Radiance Fields A radiance field is comprised of two spatially-varying fields as a function of spatial coordinate $\mathbf{x}$: density $\sigma(\mathbf{x})$ and color $c(\mathbf{x}, \mathbf{d})$, where the color field may depend on viewing direction $\mathbf{d}$ in addition to $\mathbf{x}$. A 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 [31] to large hierarchies of hashes and grids [33]. Though this approach can produce highly realistic renderings, accurately integrating Equation 1 requires a large number of calls to the underlying field, which means that training and rendering may be slow.

3D Gaussian Splatting Like NeRF, 3DGS uses alpha



(a) Top Down View (b) 3DGS [23] (c) StopThePop [36] (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.

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.

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. Our method enables *exact* volume rendering. In Section 4.1 we describe our scene representation and the algorithm for volume rendering it, and in Section 4.2 we describe how we optimize it to the input images.

4.1. Exact Primitive-based Rendering

We use a simple primitive-based rendering model, where each primitive has a constant density and a single color (in 3D space). We choose our primitive's shape to be an ellipsoid, which, similar to the Gaussians in 3DGS, is fully characterized by a position vector, rotation quaternion, and scale vector. To model view-dependent color, the ellipsoid's color changes with the view direction, which we store as a spherical function represented by the coefficients for the first two degrees of spherical harmonics.

To render a given ray, we iterate over all front and back surfaces of all primitives intersected by the ray. When we step through the front of a primitive, we add its density $\Delta\sigma_k$ and premultiplied color $\Delta\sigma\Delta\mathbf{c}_k$ to two running totals, $\sigma_i, \hat{\mathbf{c}}_i$. When we step through the back, we subtract the same density and pre-multiplied color. Using these running totals, we can calculate the color $\mathbf{c}_i$ and density σ_i for the ith interval:

$$\sigma_i = \sum_{k=1}^{i} \Delta\sigma_k \,, \quad \mathbf{c}_i = \frac{1}{\sigma_i}\hat{\mathbf{c}}_i = \frac{1}{\sigma_i}\sum_{k=1}^{i} \Delta\sigma_k \Delta\mathbf{c}_k \,. \tag{2}$$

The key here is that this decoupling of the primitive into the front and back allow us to rearrange the surfaces along the ray, as seen in Figure 3. In this example, we can see two

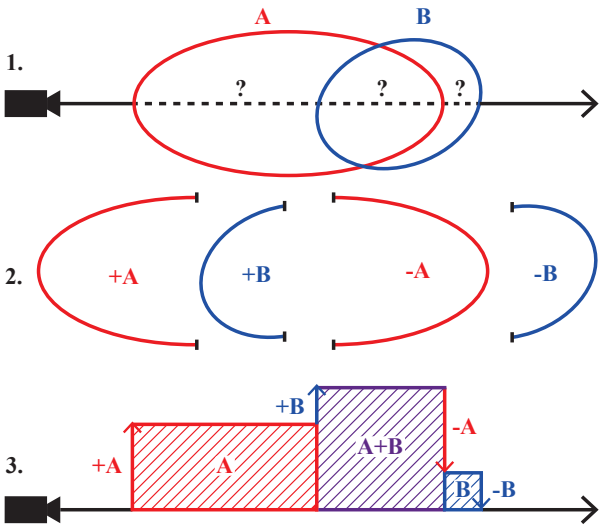


Figure 3. A visualization of our blending algorithm. 1. We start with our primitives, in this case two, and we are trying to recover the value of the intervals along the ray. 2. We take the change in density and color and sort then from closest to furthest. 3. We sum these changes as we progress through the ray, retrieving the interval values.

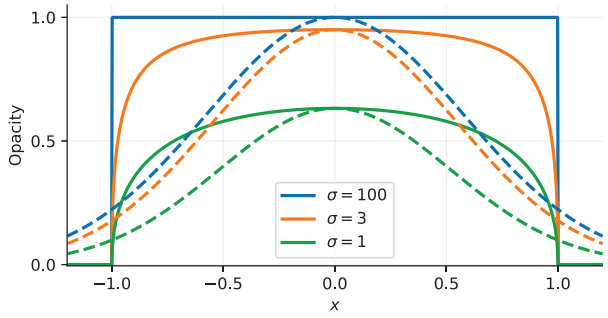


Figure 4. Here we show 1D screen-space slices of rendered opacity profiles for different rendering methods, to demonstrate one advantage of our density-based parameterization. We show 3 different peak opacities and their profiles, where solid lines represent our density based ellipsoid primitives and 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 ($\sigma = 100$).

overlapping primitives. In 3DGS, we would simply render A and B separately, ignoring their overlap, and then combine them. In contrast, our method allows rendering this representation exactly by computing the density and color at the three intervals A, A+B, and B, by tracking their changes as we progress through the ray. Since color and density are constant within each interval of length Δt_i between consecutive intersection points, there is a simple closed-form solution to the volume rendering equation:

$$\mathbf{c}_i(1 - \exp(-\sigma_i \Delta t_i)), \tag{3}$$

By tracking the running total and the distance between the intersection steps, Δt_i, we are able to integrate the volume rendering equation exactly by alpha compositing these simple segments, as follows:

$$C = \sum_{i=1}^{N} \mathbf{c}_i (1 - \exp(-\sigma_i \Delta t_i)) \prod_{j=1}^{i-1} \exp(-\sigma_j \Delta t_j). \tag{4}$$

Unsurprisingly, the expression in Equation 4 is identical to the quadrature rule derived by Max [30], which uses a piecewise-constant approximation to density and color — though in our case this piecewise-constant property is an intentional design decision, not an approximation. Our work uses this closed-form solution to enable real-time rendering of the entire collection of ellipsoids without introducing any approximation errors which may cause artifacts.

Note that although 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 blend together 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.

4.2. Optimization

Our optimization framework is built on top of 3DGS, where instead of rasterizing Gaussians we ray trace and volume render our ellipsoid-based representation using the method described in Section 4.1. In particular, we reuse the Adaptive Density Control (ADC) introduced in 3DGS, with some modifications for our density-based primitives.

The first, essential change is to convert from the 3DGS opacity framework to a density framework. The naive approach would be to assign each primitive a density instead of opacity, but this 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 [27]. Intuitively, this is because the outside of the primitive becomes opaque.

We avoid this problem by optimizing over a parameter α_k which is mapped to density value used during rendering:

$$\sigma(\alpha_k) = -\frac{\log(1 - 0.99 \cdot \alpha_k)}{\min(s_{kx}, s_{ky}, s_{kz})}. \tag{5}$$

With this, a primitive with density $\sigma(\alpha_k)$ when viewed along its shortest axis will have a peak opacity equal to α_k, ensuring at least one view will have non-zero gradient.

Finally, 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:

$$0.99 < 1 - \exp(-\sigma(\alpha_k) \cdot \max(s_{kx}, s_{ky}, s_{kz})). \tag{6}$$

5. Implementation

Our model was implemented using PyTorch, CUDA, OptiX [35], 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 [23] codebase, and make a few adjustments to handle density based primitives (Section 5.2 and Appendix B).

(a) Fisheye training and rendering

(b) Shallow depth of field rendering

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

5.1. Ray Tracing (BVHs) for Sorting

We use ray tracing to sort primitives since it offers more flexibility than a rasterization approach like StP [36]. Ray tracing allows for effects like fisheye projection and defocus blur (see Figure 5) as well as random pixel offsets during training, which we find improves performance.

We start by providing the NVIDIA OptiX [35] framework with an Axis-Aligned Bounding Box (AABB) for the ellipsoid, which is used to construct a binary tree. We then provide an intersection test to refine the search if a ray hits the bounding box, for which we adapt the stable isotropic ray/sphere intersection described by Haines et al [19]. To further accelerate rendering, we also integrate the recent approach of 3DGRT [32], which accumulates multiple hits within a queue during each call to OptiX trace.

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 the opacity and $\mathbf{s} \in \mathbb{R}^3$ is the ellipsoid axis scaling vector 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 is 2.5×10^{-7}, cloning is 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 also divide the density in half, similar to [7].

6. Results

We evaluate EVER on the 9 scenes introduced in Mip-NeRF 360 [3], the 4 scenes from DeepBlending [21] and Tanks&Temples [25], 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 use the exposure trick introduced in [15] to allow evaluation on data sets 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 dataset.

We evaluate EVER on the 9 scenes introduced in Mip-NeRF 360 [3], the 4 scenes from DeepBlending [21] and Tanks&Temples [25], 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 use the exposure trick introduced in [15] to allow evaluation on data sets 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 dataset.

There have been many follow-ups to 3DGS, each introducing different kinds of improvement, so we limit the scope of our comparison to best isolate our contribution. There are many recent approaches for densification [7, 8, 43, 50, 52], 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

	Mip-NeRF360			Zip-NeRF			Tanks&Temples & DeepBlending		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
3DGS [23]	27.48	.816	.216	25.84	.817	.358	26.65	.848	.263
StopThePop [36]	27.33	.816	.212	25.92	.819	.352	26.60	.847	.252
3DGRT [32]	27.20	.818	.248	-	-	-	26.22	.865	.254
SMERF [15]	27.99	.818	.238	27.28	.829	.339	-	-	-
Our model	27.51	.825	.194	26.58	.845	.308	26.59	.889	.234
ZipNeRF [4]	28.54	.828	.198	27.37	.836	.305	-	-	-
	GPU-hr ↓	Mem. ↓	FPS ↑	GPU-hr ↓	Mem. ↓	FPS ↑	GPU-hr ↓	Mem. ↓	FPS ↑
3DGS [23]	0.54	763MB	224	1.07	222MB	559	0.34	563MB	560
StopThePop [36]	0.60	780MB	180	0.24	223MB	403	0.39	549MB	179
3DGRT [32]	0.83†	383MB	156†	-	-	-	0.83†	388MB	190†
SMERF [15]	272	139MB	454†	528	4108MB	408†	-	-	-
Our model	1.04	1134MB	36	1.26	1694MB	24	0.86	1173MB	44
ZipNeRF [4]	32	903MB	0.5	48	903M	0.5	-	-	-

Table 1. Results on the 9 scenes from Mip-NeRF360 [2], 4 large scenes from Zip-NeRF [4] datasets, and 4 scenes from DeepBlending [21] and Tanks&Temples [25] combined. The increased sharpness and perceptual quality are reflected in the qualitative comparison in Fig. 6. "GPU-hr" indicates the number of GPU hours it takes to train a model. "Mem." indicates the amount of memory the finished model consumes on disk. Variations in memory usage between 3DGS, StopThePop, 3DGRT, and our model are due to densification differences, as memory per primitive is the same. FPS results are computed at test resolution, and results with a † were not recomputed and use different high end GPUs.

model. There are also many different ways to change rendering [5, 18, 22, 24, 36, 53], 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. We compare our method with 3DGS [23], and with 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 [32] because our work uses their ray tracing algorithm, and with SMERF [15], because even though it is a distillation based method that takes 33 hours to train on 16xA100 GPUs compared to 1-2 hours for our 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. 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.

To accompany our theoretical explanation of why our method does not suffer popping, we compare our method to StopThePop and 3DGS using the metric from StopThePop (StP) to measure the amount of popping, closely following the original author's methodology across all the original scenes measured. EVER achieves a lower (better) score at step=1, and 0.0 error (lowest) with step=7, which StP called a more reliable step size, see Table. 2.

StP Popping Metric	DeepBlending		M360 Indoor		M360 Outdoor		Tanks & Temples	
	$\exists$LIP$_1$	$\exists$LIP$_7$	$\exists$LIP$_1$	$\exists$LIP$_7$	$\exists$LIP$_1$	$\exists$LIP$_7$	$\exists$LIP$_1$	$\exists$LIP$_7$
3DGS	0.0063	0.0122	0.0072	0.0149	0.0083	0.0154	0.0107	0.0315
StopThePop	0.0052	0.0055	0.0060	0.0073	0.0083	0.0115	0.0076	0.0114
Ours	0.0031	0.0000	0.0049	0.0000	0.0067	0.0000	0.0040	0.0000

Table 2. Experimental evidence that our method does not suffer popping, using the metric from StopThePop [36]. $\exists$LIP$_7$ is considered a more reliable step size by the authors because the signal is stronger than the noise caused by the optical flow used to compute the metric.

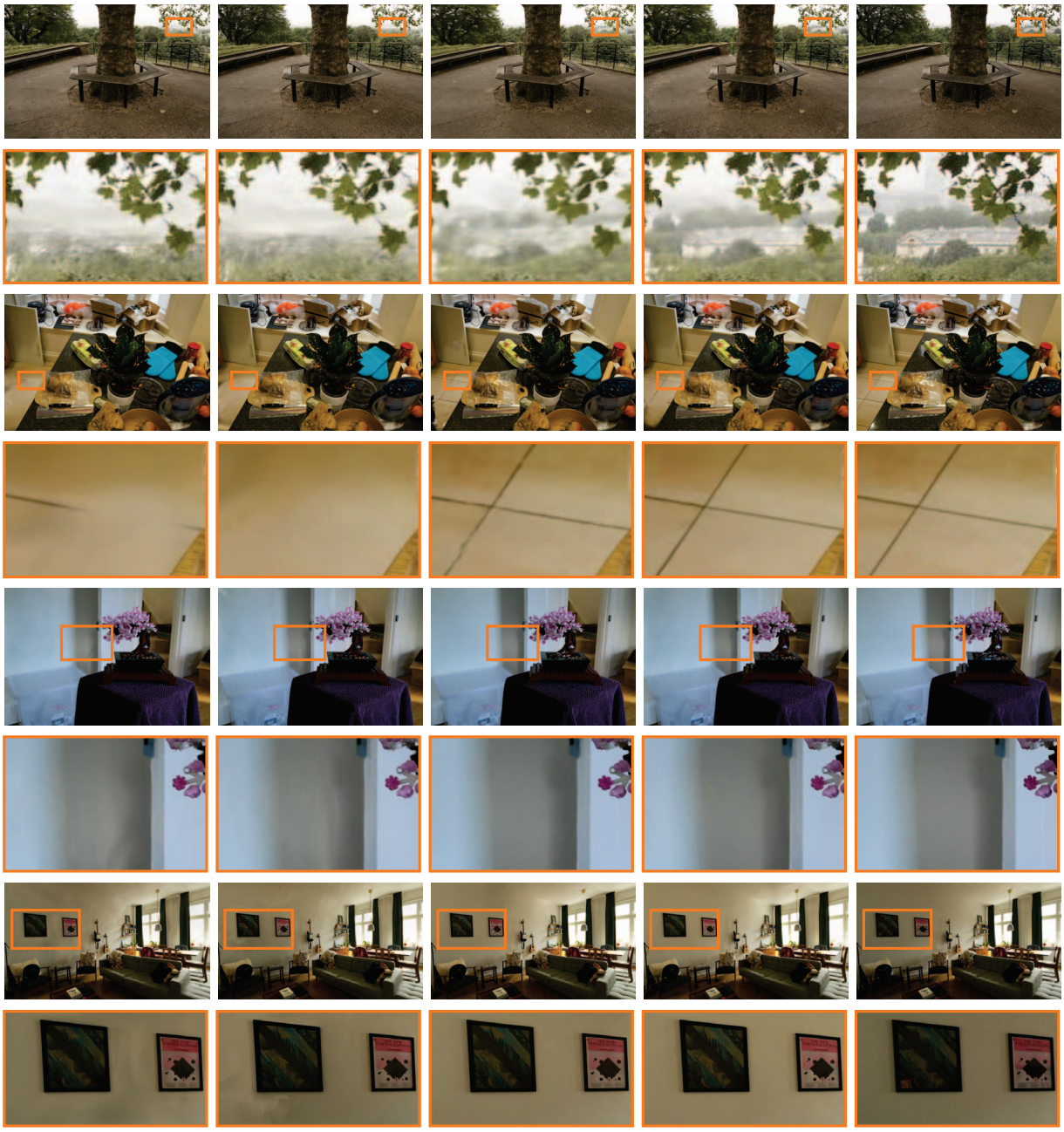


Figure 6. Visual comparison of our method on the Mip-NeRF [2] and Zip-NeRF [4] datasets. Rows 1 and 2 show regions where our results are sharper. Rows 3 and 4 show how proper blending of primitives helps with handling lighting on textureless surfaces. 3DGRT is omitted due to missing Zip-NeRF results.

6.3. Performance

At 720p, we achieve average framerates of 36 FPS on the mip-NeRF 360 outdoor scenes, 66 FPS on the mip-NeRF 360 indoor scenes, and 30 FPS on the Zip-NeRF scenes 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

3DGS Ours StopThePop Ours

Figure 7. Here we show epipolar plane images [6] to visualize popping in 3DGS [23] and StopThePop [36]. We recorded sequences of images while rotating the camera in the *berlin* [4] (left) and *train* [26] (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.

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". We also tested what would happen if we sorted the primitives per ray like StopThePop without our color blending technique, which is labeled "No Mixing". Both of these resulted in visual degradation in areas where the method had been mixing primitive 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 3. "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, which reduces the reliance on the SfM reconstruction. The inverse contraction sampling is described in the supplemental material. "No Rand Centers" refers to ablating randomizing the pixel centers, which helps with thin structures on the bicycle scene. We jitter our rays by picking a point within the pixel from a uniform random distribution, as opposed to using the center of the pixel.

	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 3. An ablation study reporting average performance on the *bicycle* and *counter* scenes from MipNeRF 360 [2] and the *berlin* scene from ZipNeRF [4]. The "No Anisotropic" ablation shows the cost of our regularizer that increases frame-rate.

7. Conclusion

Our double-intersection approach for Exact Volumetric Ellipsoid Rendering can be used to blend any number of overlapping primitive colors in a physically accurate way. We have shown that our method outperforms Zip-NeRF on the larger Zip-NeRF datasets on sharpness, while outperforming all other real-time methods on SSIM and LPIPS on the Mip-NeRF360 and Zip-NeRF datasets. By isolating the effect of our renderer, we know that this improvement comes from a combination of density based primitives and double-intersection based, physically accurate blending. This results in an approach that bridges the gap between slow but accurate radiance field methods such as Zip-NeRF, and fast but inaccurate radiance field methods like 3DGS.

We also show that sorting primitives per a pixel is not sufficient for eliminating the popping artifacts in 3DGS. It is only by combining per pixel sorting, which we achieve with ray tracing, with proper primitive blending, that we are able to eliminate these artifacts. Together, EVER is able to produce high-quality, large scale renderings that are guaranteed to be 3D-consistent, and does so at 30 FPS at 720p on a single NVIDIA RTX4090.

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, 8

[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, 6

[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, 6

[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] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Transactions on Graphics*, 2018. 5, 6

[22] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2D Gaussian Splatting for Geometrically Accurate Radiance Fields. *SIGGRAPH*, 2024. 6

[23] 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

[24] 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, 6

[25] 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

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

[27] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable Monte Carlo Ray Tracing through Edge Sampling. *ACM TOG*, 2018. 4

[28] 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

[29] Stephen Lombardi, Tomas Simon, Gabriel Schwartz, Michael Zollhoefer, Yaser Sheikh, and Jason Saragih. Mixture of volumetric primitives for efficient neural rendering. *ACM TOG*, 2021. 2

[30] Nelson Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1995. 2, 4

[31] 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

[32] 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

[33] 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

[34] 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

[35] 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

[36] 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

[37] 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

[38] Christian Reiser, Rick Szeliski, Dor Verbin, Pratul Srinivasan, Ben Mildenhall, Andreas Geiger, Jonathan T. Barron, and Peter Hedman. MERF: Memory-Efficient Radiance Fields for Real-time View Synthesis in Unbounded Scenes. *ACM TOG*, 2023. 2

[39] 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

[40] Sara Rojas, Jesus Zarzar, Juan C. Pérez, Artsiom Sanakoyeu, Ali Thabet, Albert Pumarola, and Bernard Ghanem. Re-ReND: Real-Time Rendering of NeRFs across Devices. *ICCV*, 2023. 2

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

[42] 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

[43] Evangelos Ververas, Rolandos Alexandros Potamias, Jifei Song, Jiankang Deng, and Stefanos Zafeiriou. SAGS: Structure-Aware 3D Gaussian Splatting. *arXiv:2404.19149*, 2024. 2, 5

[44] 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

[45] Angtian Wang, Peng Wang, Jian Sun, Adam Kortylewski, and Alan Yuille. VoGE: A Differentiable Volume Renderer using Gaussian Ellipsoids for Analysis-by-Synthesis. *arXiv:2205.15401*, 2022. 2

[46] 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

[47] 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

[48] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-NeRF: Point-based neural radiance fields. *CVPR*, 2022. 2

[49] 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

[50] 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

[51] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for Real-Time Rendering of Neural Radiance Fields. *ICCV*, 2021. 2

[52] Zehao Yu, Torsten Sattler, and Andreas Geiger. Gaussian Opacity Fields: Efficient Adaptive Surface Reconstruction in Unbounded Scenes. *arXiv:2404.10772*, 2024. 2, 5

[53] Yang Zhou, Songyin Wu, and Ling-Qi Yan. Unified Gaussian Primitives for Scene Representation and Rendering. *arXiv:2406.09733*, 2024. 2, 6

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