Radiant Foam: Real-Time Differentiable Ray Tracing

Shrisudhan Govindarajan*[1], Daniel Rebain*[2], Kwang Moo Yi[2], Andrea Tagliasacchi[1,3,4]

[1]Simon Fraser University, [2]University of British Columbia,
[3]University of Toronto, [4]Google DeepMind, *equal contributions

radfoam.github.io

arXiv:2502.01157v1 [cs.CV] 3 Feb 2025

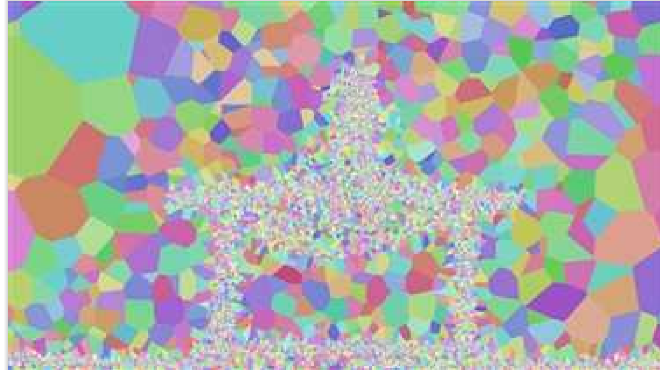

Figure 1. **Teaser –** We introduce the Radiant Foam differentiable 3D representation, which can be used to learn accurate radiance fields for any novel view synthesis applications (left). If we slice our foam along the plane highlighted by the red "laser", we expose (right) the internal structure of our representation: a polyhedral mesh that provides an injective parameterization of the 3D domain. Our representation is a *foam*, as the polyhedral cell structure is analogous to a closed-cell foam which partitions space into regions physically separated by thin, flat walls. It is *radiant*, as each foam bubble emits a view-dependent radiance that can be used to model the plenoptic function.

Abstract

Research on differentiable scene representations is consistently moving towards more efficient, real-time models. Recently, this has led to the popularization of splatting methods, which eschew the traditional ray-based rendering of radiance fields in favor of rasterization. This has yielded a significant improvement in rendering speeds due to the efficiency of rasterization algorithms and hardware, but has come at a cost: the approximations that make rasterization efficient also make implementation of light transport phenomena like reflection and refraction much more difficult. We propose a novel scene representation which avoids these approximations, but keeps the efficiency and reconstruction quality of splatting by leveraging a decades-old efficient volumetric mesh ray tracing algorithm which has been largely overlooked in recent computer vision research. The resulting model, which we name Radiant Foam, achieves rendering speed and quality comparable to Gaussian Splatting, without the constraints of rasterization. Unlike ray traced Gaussian models that use hardware ray tracing acceleration, our method requires no special hardware or APIs beyond the standard features of a programmable GPU.

1. Introduction

Neural radiance fields (NeRF) have revolutionized 3D computer vision by allowing the extraction of dense 3D rep-

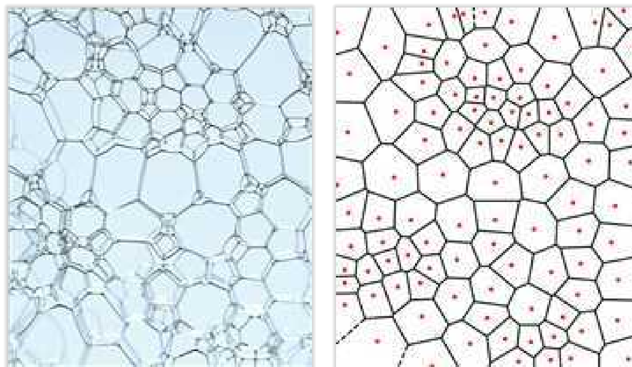

Figure 2. **Radiant Foam –** (left) In a stable foam, the pressure inside each bubble is roughly equal. The interfaces between bubbles settle into shapes that balance forces, leading to polygonal cells resembling Voronoi patterns (right). Our representation is nothing but a dense Voronoi tessellation of 3D space, where each point belongs to exactly one Voronoi cell. The position of Voronoi sites is differentiable, making it amenable to gradient-based optimization.

Figure 3. **Ray-based effects –** Ray tracing simplifies the implementation of many effects which are difficult to approximate with rasterization. To motivate our work, we show here examples of integrating reflections, refractions, and non-linear camera models into our rendering pipeline. Each would be complicated to achieve with rasterization, but requires only minor modification to our rendering code.

resentations from collections of 2D images [28]. In their early days, radiance fields suffered from low training and testing/rendering speed. Since then, researchers have developed more efficient models [30], as well as techniques for distilling models which enable highly efficient rendering/testing [5]. In particular, the real-time rendering performance was made possible by leveraging the rasterization pipeline readily available on GPU hardware. Soon thereafter, researchers proposed to also incorporate rasterization into the training process, leading to the development of the now wildly popular 3D Gaussian Splatting (3DGS) [19].

By comparing the development of radiance fields to that of classical graphics, one may argue that history is repeating itself: rasterization was introduced in computer graphics to *approximate* the rendering equation [18], so to make it amenable to real-time rendering workloads. However, light effects that are trivial to implement in ray tracing (e.g. reflections, refractions, and transparency; see Fig. 3), are rather difficult to implement with rasterization.[1] Consider the "Graphics Gems" series of books[2] as a practical example, given how much of their content describes clever implementation tricks to express advanced light effects with rasterization engines. Nonetheless, since the introduction of dedicated hardware support (NVIDIA RTX) in 2018, real-time rendering engines have, at least partially, returned to simulate light transport via ray tracing.

Recently, research on 3DGS representations has flourished, and much of this research is seemingly trying to fix issues *caused* by rasterization, such as the removal of 3DGS popping artifacts [35], or the introduction of more complex camera projection models [15]. Others integrated ray tracing with 3DGS, so to accelerate rendering and enable more complex light behavior [29]. But these developments have not discouraged researchers from investigating new, better, 3D representations. The community craves a return to polygonal meshes, as meshes are the unquestionable workhorse of modern computer graphics. And while we can find very interesting attempts at employing meshes for the modeling of radiance fields [10, 23, 45], as we will later discuss, none of these has realistically been able to oust 3DGS as *the* representation for learning radiance fields.

Rather than revisiting history, and proposing clever engineering tricks that enable rasterization to work slightly better, we take a drastically different approach. In particular, we highlight how, over two decades ago, Weiler et al. [52] showed that fields represented by volumetric meshes admit a very efficient ray-tracing algorithm which requires no special hardware. This approach has been subsequently overlooked in the resurgence of volume rendering methods, and we hope to re-introduce volumetric mesh models which can benefit from it to the differentiable rendering community.[3] In this paper, we make this representation differentiable, and carefully design refinement techniques for the underlying mesh. These refinements allow us to accurately represent the surface of objects, and yet render efficiently by skipping empty portions of volume.

In a nutshell, our solution, which we name `Radiant Foam` (Fig. 2) provides 3DGS-like rendering speed and quality, but has a training modality based on rays that resembles the one from NeRF [28]. This implies that many NeRF techniques can now be seamlessly adapted to our method, with the significant advantage that the underlying geometry is *explicitly* represented by a volumetric mesh. We parameterize this volumetric mesh as a 3D Voronoi diagram (Fig. 1), which enables training of a mesh structure with dynamic connectivity through gradient descent by avoiding the discontinuities associated with discrete changes in other representations. We also propose a coarse-to-fine training approach, which enables rapid construction of mesh models with adaptive resolution.

[1] After all, is this really that surprising, given that the physics of light is taught as the propagation of light rays through an environment?

[2] `http://www.graphicsgems.org`

[3] It is also interesting to note that the volume rendering tutorial by Max and Chen [27] that is cited in conjunction with Mildenhall et al. [28] describes volumetric rendering within the context of volumetric meshes that *partition* space, which is more similar to [26] than to [28].

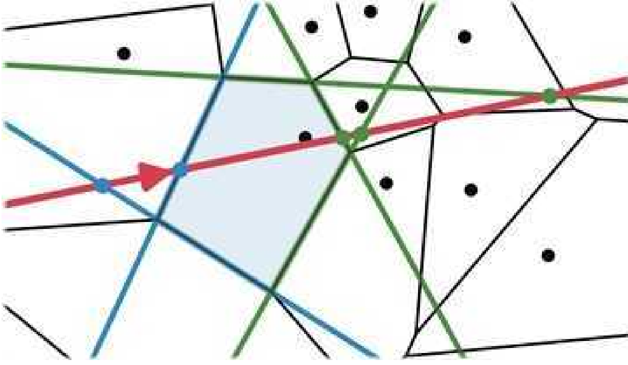

Figure 4. **Ray tracing foams –** When a ray (red) enters a cell, we iterate through all the planar cell faces to identify the face through which the ray exits. This exit intersection is the first intersection along the ray with a normal vector less than 90 degrees from the ray direction (green); other intersections are considered back-facing (blue) and ignored. As the faces each correspond to a neighboring cell, the tracing then proceeds by stepping into the cell associated with the exit intersection and repeating the process.

2. Related Work

Neural Radiance Fields (NeRFs) [28] represent 3D scenes as volumetric radiance fields encoded within coordinate-based neural networks. This representation allows querying the network at any spatial location to retrieve volumetric density and view-dependent color, facilitating the generation of photo-realistic novel views. The success of NeRFs has led to numerous follow-up works. For instance, significant effort has been devoted to enhancing training speed [30, 37], quality [1, 2], and to extract surfaces from the representation [49, 54]. Finally, several works investigated ways to speed up the inference by baking the neural fields to more performant representations [5, 7, 38, 44, 50]. While achieving high quality and fast rendering speeds, these methods often employ multi-stage training procedures.

Point-based rasterization. Point-based rendering with primitives such as circles, spheres, or ellipsoids [34, 40, 56] laid the foundations for point-based rasterization. Differentiable rendering using depth-based blending has also been extended to volumetric particles in Pulsar [24], which employs sphere-based differentiable rasterization to render scenes with millions of spheres in real time. While Pulsar provides a differentiable representation, Kerbl et al. [19] provides more expressive primitives in the form of soft, anistropic Gaussians, which can be differentiably optimized to fit the scene content. This approach has inspired several follow-up works aimed at reducing its reliance on strong initialization [21], reducing rendering time and memory footprints [8, 31, 33], enabling the reconstruction of surfaces [11, 14], and improving their ability to be trained for large spatial extents [20, 39].

Extensions of 3DGS. While significant progress has been made, these approaches still inherit the inherent limitations of rasterization. They struggle to handle camera distortions, model secondary lighting effects, or simulate sensor-specific properties like rolling shutter or motion blur. Recent efforts have aimed to address these challenges. For example Niemeyer et al. [32] distilled a Zip-NeRF [3] into a 3DGS so to model distorted cameras and rolling shutter effects. To capture secondary lighting effects, recent works have explored incorporating occlusion information into spherical harmonics for each Gaussian [9, 25], or leveraging shading models and environment maps [16]. These methods either rely on rasterization during inference [9], or during training [16, 25] hence inheriting the limitations of rasterization. For complex lens effects, Seiskari et al. [43] modeled motion blur and rolling shutter by *approximating* them in screen space through rasterization and pixel velocities. Our approach introduces a principled framework for efficient ray tracing of volumetric primitives, overcoming the aforementioned limitations and enabling the simulation of effects like reflection, refraction, and camera distortion.

2.1. Ray tracing of volumetric primitives

Ray tracing has been a cornerstone of photo-realistic rendering since its introduction [53]. It enables the accurate simulation of light interactions with scene geometry, making it indispensable for applications requiring effects such as shadows, reflections, and refractions. Modern advancements in hardware have further accelerated ray tracing, allowing for real-time applications [17]. Recently, Moenne-Loccoz et al. [29] proposed to use the NVIDIA OptiX ray tracer with 3DGS [19] for fast ray-tracing. While the method performs real-time ray tracing, it suffers from the tendency of 3DGS [19] to produce overlapping primitives which degrade the quality of the hierarchical acceleration structure and increase the number of intersections per ray. Our method represents the scene using a non-overlapping polyhedral mesh without the need for a secondary acceleration structure, thereby efficiently avoiding these limitations.

2.2. Delaunay triangulation and Voronoi diagrams

In computer graphics, Delaunay triangulations [6] and Voronoi diagrams [48] have been extensively studied for meshing (surfaces and volumes), and spatial partitioning [12, 46, 51]. Recently, they have also found novel applications in the realm of differentiable rendering. DMTet [45] introduced a deformable tetrahedral grid, generated using Delaunay triangulation, as an underlying 3D representation, while Tet-Splatting [10] implements a differentiable rasterizer for this representation. DeRF [36] leverages Voronoi diagrams to spatially decompose a scene, resulting in faster training and inference, while Tetra-NeRF [23] employs a tetrahedral Delaunay mesh to model scenes more effectively. While these methods inherit the ray-marching ap-

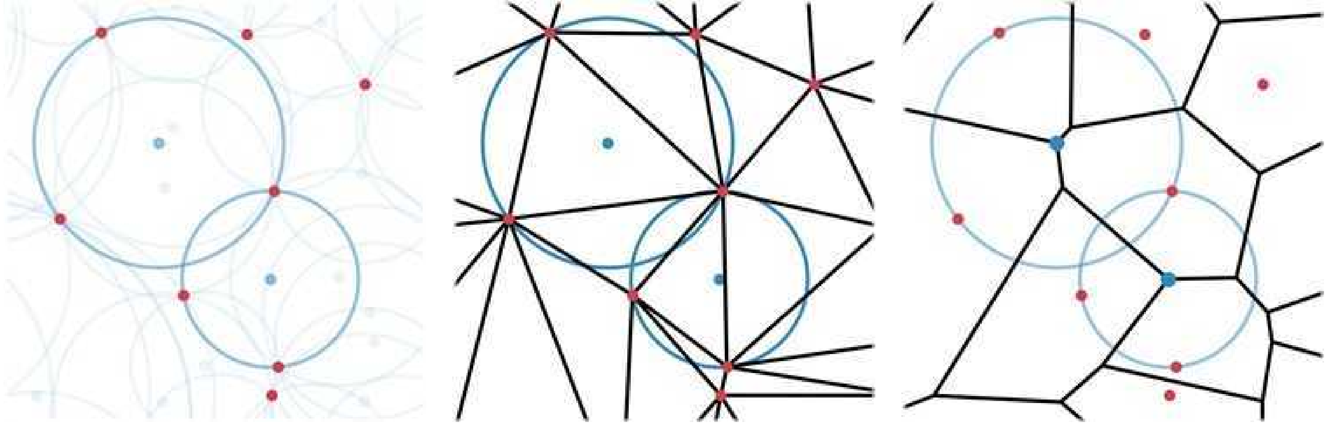

Figure 5. **Delaunay and its dual Voronoi –** (left) Given a set of points (red) in $\mathbb{R}^N$, we can find circumspheres (blue) which each pass through $N+1$ points. (center) The set of all circumspheres which contain no points on their interior defines the Delaunay triangulation, where the $N+1$ points tangent to each circumsphere form a simplex. (right) These circumspheres also describe the Delaunay triangulation's dual, the Voronoi diagram: the centers of circumspheres tangent to a point become the vertices of the Voronoi cell containing that point.

proach of [28] for volume rendering, they share a common limitation: slow rendering speeds caused by many MLP evaluations required per ray. In contrast, with our method we propose to *optimize* the mesh topology generated by the Voronoi diagram as a *differentiable* polygonal mesh. Our efficient ray tracing of volumetric particles also significantly accelerates the rendering speed.

3. Method

Our method addresses the now-familiar problem of constructing representations of scenes from image collections. As with NeRF [28] and its numerous successors, we achieve this reconstruction by gradient-based optimization of a differentiable scene representation. In the following sections we propose a volumetric mesh-based differentiable representation, and explain how we are able to effectively optimize it from image supervision.

3.1. Volume rendering

Volume rendering has become the workhorse of modern differentiable scene reconstruction methods. Volume rendering allows all points in space to make a continuously varying contribution to the observed color of viewing rays which pass through those points. The effect of this contribution is controlled by a density field, which creates occlusions, and a radiance field, which determines the brightness of light observed. Unlike many alternative rendering formulations, volume rendering of continuous fields is fully continuous with respect to all degrees of freedom, including both the viewpoint, as well as the density and radiance field values. This property makes it very amenable to gradient-based optimization. Volume rendering is defined by an integral over the segment $(t_{\text{min}}, t_{\text{max}})$ of a viewing ray. Specifically the observed color $\mathbf{c_r}$ for a ray $\mathbf{r}$ is:

$$\mathbf{c_r} = \int_{t_{\text{min}}}^{t_{\text{max}}} T(t) \cdot \sigma(\mathbf{r}(t)) \cdot \mathbf{c}(\mathbf{r}(t)) \, dt, \tag{1}$$

$$T(t) = \exp\left(- \int_{t_{\text{min}}}^{t} \sigma(\mathbf{r}(u)) \, du \right), \tag{2}$$

where $\mathbf{r}(t)$ denotes the point in 3D space at distance t along ray $\mathbf{r}$, and $\sigma(\cdot)$ and $\mathbf{c}(\cdot)$ denote the density and radiance fields respectively; see [47] for more details.

Piecewise constant volumes. In the case of *piecewise constant* $\sigma(\cdot)$ and $\mathbf{c}(\cdot)$, the integral can be expressed as a sum over all N ray segments with constant field values:

$$\mathbf{c_r} = \sum_{n=1}^{N} T_n \cdot (1 - \exp(-\sigma_n \delta_n)) \cdot \mathbf{c}_n \, dt, \tag{3}$$

$$T_n = \prod_{j=1}^{n} \exp(-\sigma_j \delta_j), \tag{4}$$

where δ_n is the width of segment n. This formulation admits a simple implementation as a loop over the segments in order of depth. NeRF-based methods typically use (3) as an *approximation* to (1), with segments sampled according to some importance sampling scheme, e.g. see [2].

Conversely, in our model these forms are exactly equivalent. We propose to leverage the algorithm by Weiler et al. [52] along with a model of constant field values within the cells of a volumetric mesh (see Fig. 4), arriving at a representation for which (3) gives the *exact* volume rendering result. While this choice is highly advantageous in avoiding any complicated or expensive sampling schemes, we must take great care to not interfere with the continuity of the representation, which is critical for gradient-based optimizers.

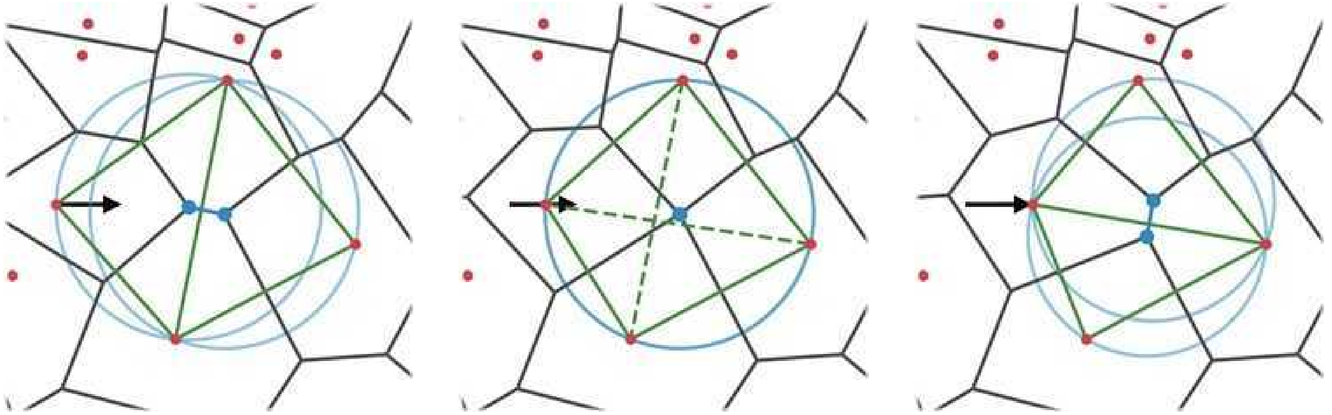

Figure 6. **Edge flips –** The connectivity of the Delaunay graph (green) is sensitive to small positional perturbations, leading to "edge flips" in the triangulation. These discrete changes occur at configurations where the circumspheres (blue) of two neighbouring simplices become identical. In the dual Voronoi diagram, this configuration also corresponds to a discrete change, but the cell boundary which changes has an area of zero at the moment of the flip (center). Consequently, while there are still discrete changes in the adjacency structure of the Voronoi diagram, the *shapes of the cells* vary continuously with the positions of the points (red), which enables gradient-based optimization.

In particular, the volumetric mesh itself receives gradients only through the segment widths δ_n, which are determined by the locations of ray intersections with cell boundaries. It is therefore critical that these boundary intersections *vary continuously* with the optimizable parameters of the model.

3.2. Differentiable mesh representation

Constructing mesh representations of volumes or surfaces through gradient-based optimization is hardly a new problem, and many methods tackle it using a variety of strategies; see Shen et al. [45] and citations therein. Our method, however, encounters and must solve a challenge that most mesh optimization methods either avoid, or solve with discrete optimization techniques. Generally, meshes are determined by degrees of freedom in two groups: vertex locations $\mathcal{V}$ and cells $\mathcal{C}$ (i.e. connectivity, or mesh topology). Assuming all other parameters are fixed, optimizing for vertex locations is typically straightforward, as the intersections between rays and cell boundaries vary smoothly with vertex locations. The true challenge arises if one desires to also optimize the connectivity $\mathcal{C}$ of the representation.

The Delaunay triangulation. Because mesh connectivity is inherently discrete, we can not optimize it directly with gradient-based methods. To avoid this problem, we can define connectivity in terms of the vertex locations, such that each possible configuration of vertices corresponds to a *unique* set of cells. The most obvious choice for this mapping is the *Delaunay triangulation* [6], which in 3D comprises the set of all tetrahedral cells $\mathbf{c}_i \in \mathcal{C}$ which may be formed from four vertices such that their circumspheres contain no additional vertices (the so-called "Delaunay criterion"). This construction is unique for point sets in general position (those lacking groups of 5 or more co-spherical points), and is easily computed using well-known and efficient algorithms; see Section 3.4. This strategy encounters two significant problems if we wish to use it as a basis for differentiable volume rendering: (1) the discrete nature of the mesh connectivity is not entirely avoided, as the boundaries of Delaunay cells undergo discrete "flips" whenever a vertex enters the circumsphere of another cell. These flips introduce *discontinuities* into the optimization landscape, which interfere with the convergence of gradient descent; (2) the number of tetrahedra in this model is not fixed in this model, and therefore it is not straightforward to associate each cell with optimizable σ and $\mathbf{c}$ values as required for volume rendering. The latter issue could be solved by associating field values with vertices (vs. tets), and interpolating those values within the cell, but this complicates volume rendering.

The Voronoi diagram. Rather than utilize the Delaunay mesh directly and suffer its limitations, we instead look to the dual graph of the Delaunay triangulation: the *Voronoi diagram* [48]. As shown in Figure 5, this is constructed by placing a vertex at the circumcenter of each Delaunay tetrahedron; it partitions space into convex polyhedral cells $\mathbf{c}_i \in \mathcal{C}$ consisting of points which share a nearest neighbour among the primal vertices[4] $\mathbf{p}_i$ of the Delaunay triangulation:

$$\mathbf{c}_i = \{x \in \mathbb{R}^3 : \arg\min_j ||x - \mathbf{p}_j|| = i\}. \tag{5}$$

This may at first seem like a strange choice... after all, *if the Delaunay Triangulation is unavoidably discontinuous in optimization, will not its dual also suffer this issue?* The answer to this question lies in the fact that any discrete change in the connectivity of the Voronoi Diagram exactly

[4]Also known as Voronoi "sites" or "seeds".

coincides with the point that the affected cell faces attain zero surface area. As shown in Figure 6, the discrete flips are effectively *hidden* within these zero-volume regions of space, and the resulting field representation remains completely continuous with respect to the primal vertex locations. The number of cells in the Voronoi diagram is also constant regardless of configuration, which makes the association of σ and $\mathbf{c}$ values to cells... trivial. The resulting model is then effectively a *learnable point cloud*, not dissimilar to the formulation of 3D Gaussian Splatting [19], though lacking the per-point covariance matrix. The only remaining issue is that our ray tracing algorithm [26] expects *tetrahedral* cells. We therefore modify the algorithm to handle the more general case of convex cells. This modified tracing process pre-fetches the neighboring vertices of each vertex, thereby allowing an efficient iteration over the cell faces to find ray intersections. For more detail on this algorithm, see Figure 7.

Algorithm 1 Ray tracing algorithm

```
 1: procedure RENDER(o, d)                          ▷ ray parameters
 2:     t_0 ← 0
 3:     i ← nn(o)                ▷ initial cell (nearest neighbour)
 4:     T ← 1
 5:     C ← 0
 6:     while T > ε do
 7:         x ← v_i                    ▷ v_i: primal vertex of cell i
 8:         t_1 ← ∞
 9:         i′ ← ∅
10:         for all j ∈ N(i) do ▷ N(i): neighbours of cell i
11:             x′ ← v_j               ▷ v_i: primal vertex of cell j
12:             (t_j, front) ← INTERSECT(o, d, x, x′)
13:             if front and (t_j < t_1) then
14:                 t_1 ← t_j
15:                 i′ ← j
16:             end if
17:         end for
18:         c ← c_i                            ▷ c_i: color of cell i
19:         σ ← σ_i                          ▷ σ_i: density of cell i
20:         α ← 1 − e^{−σ(t_1 − t_0)}
21:         C ← C + T α c
22:         T ← T (1 − α)
23:         t_0 ← t_1
24:         i ← i′
25:     end while
26:     return C
27: end procedure
```

Figure 7. **Ray tracing –** Our ray tracing algorithm is simple, and based on the method proposed by Weiler et al. [52]. Unlike common algorithms for tracing triangle meshes and other unstructured scene representations, we do not require a hierarchical acceleration structure, and thus avoid the associated $O(\log(n))$ query operation.

3.3. Optimization

Similarly to Kerbl et al. [19], the (mostly) local nature of Voronoi cells renders the optimization landscape more prone to local minima. We follow a similar strategy, by first carefully initializing the optimization, and then adaptively *densifying* and *pruning* Voronoi sites. Additionally, to promote the formation of surface-like densities, we also employ a regularization objective similar to the *distortion* loss [2] commonly used by NeRF methods.

Densification. Similarly to Kerbl et al. [19], to initialize training we start with a sparse point cloud obtained from COLMAP [42]. Over training, we perform densification and pruning operations to control the number of Voronoi sites and their density, allowing the model to adaptively re-allocate representational capacity to areas of space with more geometric and/or photometric detail. We observe that gradients of the reconstruction loss with respect to Voronoi site locations can be used to identify cells which are underfitting the training signal. We therefore use the norm of this gradient multiplied by the approximate radius of the cell as a measure of which cells require further densification. Inspired by Kheradmand et al. [21], we select the candidates for densification by sampling a Multinomial distribution with probability mass function proportional to this measure.

Pruning. Towards building a parsimonious representation, we remove cells from the Voronoi diagram that do not contribute to rendering. However, it is not sufficient to simply delete "empty" cells (i.e. zero density), as the geometry of Voronoi cells is determined by the positions of adjacent sites, even when the density in those cells is zero. To accurately represent object boundaries, it is therefore essential to retain cells with near-zero density that define the boundary. For this reason, our pruning strategy removes Voronoi sites that have very low density, *and* are surrounded by neighbors with very low density. This pruning ensures that we eliminate sites that neither contribute to nor define the surfaces, thereby maintaining the accuracy of the object boundaries.

Training objectives. Similarly to the *distortion* loss of Mip-NeRF 360 [2], we apply a regularization on the distribution of contribution to the volume rendering integral along the ray. This additional loss function encourages the density to concentrate at surfaces and reduces visible "floater" artifacts. This objective is computed as

$$\mathcal{L}_{\text{quantile}} = \mathbb{E}_{t_1,t_2\sim\mathcal{U}[0,1]}[|W^{-1}(t_1) - W^{-1}(t_2)|], \quad (6)$$

where $W^{-1}(\cdot)$ denotes the quantile function (inverse CDF) of the volume rendering weight distribution along the ray. This form has the same effect as the distortion loss [2], but avoids the need for a quadratic nested sum which would increase the computational cost and memory footprint of training. Denoting with $\mathcal{L}_{\text{rgb}}$ the typical L2 photometric

reconstruction loss, our overall training objective is

$$\mathcal{L} = \mathcal{L}_{\text{rgb}} + \lambda \mathcal{L}_{\text{quantile}} \tag{7}$$

3.4. Implementation details

We implemented our method in Python and C++ within the PyTorch framework, with custom CUDA kernels for ray tracing, Delaunay triangulation, and other operations requiring high efficiency. This implementation includes an interactive viewer and a (very) low overhead renderer, which we used to measure frame rates. Importantly, nothing in our implementation is dependent on dedicated ray tracing hardware, or the OptiX library, which is required by methods like [29]. Therefore, with some engineering effort, our entire rendering loop could easily be implemented in a portable rendering framework like WebGL.

Training. Our training pipeline uses the Adam [22] optimizer, and similarly to 3DGS, directly optimizes per-point position, density and view-dependent color via spherical harmonics of degree three. We use the softplus activation function with β=10 for density to constraint it within the $[0, \infty)$ range, while keeping smooth gradients. For the location of points, we start at a learning rate of $2e^{-4}$ and decay it using a cosine annealing scheduler to a final learning rate of $2e^{-6}$. For point density and spherical harmonics, we start at a learning rate of $1e^{-1}$ and $5e^{-3}$ respectively and decay it with a cosine annealing scheduler by a factor of 0.1. Following 3DGS [19], we optimize only the zero-order component of SH coefficients and the high-order coefficients with a warmup for the first 25% of the total training iterations. Similarly to [4, 21], after initialization and warm-up training, we gradually grow the number of Voronoi sites so that points are placed at useful locations. We progressively increase the number of points up to half the total training iterations, linearly increasing the number of points until the maximum desired number of points is obtained.

Voronoi optimization. We maintain an adjacency data structure throughout training, which defines the Voronoi cells for rendering. Whenever the primal vertex positions are changed we must update the adjacency by performing an incremental Delaunay triangulation. While much faster than a complete rebuild, the incremental update is still computationally expensive for large point sets, so we allow the optimizer to take multiple steps between mesh rebuilds. We start at a 1:1 ratio after each densification and increase to 1:100, as the frequency of discrete changes to the mesh decreases over time with the converging optimization. This strategy balances the overall speed of training with maintaining a relatively accurate mesh structure. All our experiments are optimized for 20k iterations, with the last 2k only updating radiance and density attributes while positions are frozen. This process takes, as an example, 70 minutes on the "bonsai" scene with an RTX 4090 GPU.

	MipNeRF 360 [2]				Deep Blending [13]			
	PSNR↑	SSIM↑	LPIPS↓	FPS↑	PSNR↑	SSIM↑	LPIPS↓	FPS↑
Rasterization								
3DGS* [19]	28.69	0.87	0.22	293	29.41	**0.90**	**0.32**	319
Mip-Splatting [55]	29.39	0.88	0.20	241	29.47	**0.90**	**0.32**	260
3DGS-MCMC [21]	**29.72**	**0.89**	**0.19**	**302**	**29.71**	**0.90**	**0.32**	**662**
Ray Tracing / Marching								
Plenoxels [41]	23.63	0.67	0.44	< 30	23.06	0.80	0.51	< 30
iNGP-Big [30]	26.75	0.75	0.30	< 30	24.96	0.82	0.39	< 30
MipNeRF360 [2]	**29.23**	**0.84**	**0.21**	<1	**29.40**	**0.90**	**0.25**	<1
3DGRT** [29]	28.71	0.85	0.25	78	29.23	**0.90**	0.32	119
RadiantFoam (v1)	28.47	0.83	**0.21**	**200**	28.95	0.89	0.26	**301**

Table 1. **Novel view synthesis –** We evaluate our method's accuracy in reconstructing held-out views for two standard datasets. Our method has similar performance to 3DGS and 3DGSRT, while providing *significantly* higher frame rates than the latter. FPS is measured as an average on the test set for each scene. *For 3DGS, we report corrected LPIPS scores provided by Bulò et al. [4] **Note that 3DGRT FPS results were measured with an RTX 6000 Ada GPU. We report the results from the original publication, as the code is not open source.

4. Experiments

We evaluated our algorithm on a total of 9 real-world scenes sourced from two publicly available datasets. Specifically, we utilized the complete set of scenes from the Mip-NeRF 360 dataset [2] except for two private scenes (flowers and treehill) and two scenes from the Deep Blending dataset [13]. These datasets contain scenes with a diverse range of capture styles, including bounded indoor scenes and expansive, unbounded outdoor environments. For Mip-NeRF 360, to make our results compatible with [19, 29], we downsample images for the indoor scenes by a factor of two, and the outdoor scenes by four. For Deep Blending scenes, we use the original image resolutions. All frame rates were measured on a consumer-grade RTX 4090 GPU.

Metrics. We assess each method using three widely recognized image quality metrics: Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index (SSIM), and Learned Perceptual Image Patch Similarity (LPIPS). In the supplementary web page we include rendered video paths for selected scenes, showcasing views significantly different from the input images.

Quantitative results. We report our quantitative results for the Mip-NeRF 360 [2] and Deep Blending [13] in Table 1. In terms of quality, our method achieves results comparable to, or slightly below, those of 3DGS [19] and 3DGRT [29] (the state-of-the-art differentiable ray tracing method). However, our method excels in rendering speed. As shown in Table 1, our efficient ray-tracing implementation achieves in some cases over 300 FPS, more than twice as fast as 3DGRT (119 FPS), while maintaining a similar rendering speed to rasterization methods.

SfM	Densify	Prune	Quantile	Bonsai	Garden	Playground	Mean
✗	✓	✓	✓	29.65	25.83	26.34	27.00
✗	✗	✓	✓	20.23	18.88	19.55	19.36
✓	✓	✗	✓	**32.25**	**26.58**	29.46	**29.15**
✓	✓	✓	✗	29.62	25.35	**29.59**	27.90
✓	✓	✓	✓	32.15	**26.58**	**29.59**	**29.15**

Table 2. **Ablation table –** We evaluate the impact of various components in our method by systematically excluding them and analyzing the reconstruction quality (PSNR↑) on the Bonsai and Garden scenes from MipNeRF 360 [2], as well as the Playground scene from Deep Blending [13].

4.1. Ablation

We isolated the different contributions and choices we made and constructed a set of experiments to measure their effect. Specifically, we test the following aspects of our algorithm: initialization from SfM, our densification and pruning strategies, and regularization loss. The quantitative effect of each choice is summarized in Table 2.

Initialization from SFM. We assess the importance of initializing our Voronoi sites from the SfM point cloud. For this, we initialize our representation with 2^{17} points from a normal distribution with a standard deviation of 10. We observe that our method performs relatively well even without the SfM points. Instead, it degrades mainly in the background and tends to have more floaters in regions that are sparsely covered in the training views, see Figure 8.

Densification and pruning. Due to numerical approximations, our triangulation algorithm can fail when processing very close (or identical) points. This limitation prevents us from initializing our representation directly with the final set of points obtained by duplicating and perturbing the Structure from Motion (SfM) points, as SfM typically produces numerous closely spaced points. Consequently, for the ablation study on densification, we utilize the random initialization strategy described earlier to initialize our representation with the intended number of points. We observe that our method significantly underperforms without densification, resulting in an under-represented scene where resources are not adequately allocated to regions with complex geometry or texture. In the ablation study of our pruning strategy, we cease pruning those Voronoi sites that are neither surface nor boundary points. We notice that this pruning approach has minimal impact on the quality of the renderings because the number of prunable points is very low. By starting with a sparse point set and progressively densifying only in under-represented areas, we ensure that points are allocated exclusively to necessary regions, resulting in few prunable points.

Quantile loss. We disable the quantile regularization while training, which results in floater artifacts that degrade rendering quality for novel views. Some scenes are more affected by this than others, as the floaters result from data-dependent ambiguities, but on average we find that the quantile regularization improves reconstruction metrics.

5. Conclusions

We have introduced Radiant Foam, a novel representation that allows real-time differentiable ray tracing. The core of our method is a foam structure of polyhedral cells, which allows efficient volumetric mesh ray tracing algorithms to be applied *without* relying on dedicated hardware such as NVIDIA RT cores. We allow these cells to be continuously optimized by parameterizing them as a Voronoi diagram, which we show to be differentiable under volume rendering. By doing so, we have shown that one can achieve similar modeling quality as 3D Gaussian Splatting, but without sacrificing the benefits of a true ray tracing-based volume renderer, nor the fast rendering speed of rasterization-based renderers.

Limitations and future work. While the Voronoi-based representation we have proposed is very effective at constructing foam models through continuous optimization, the space of possible foam models which could be used in our rendering pipeline is much larger than what is parameterized by Voronoi. Most notably, our current model always requires that cell boundaries be equidistant between neighbouring points, which leads to many small, empty cells being needed to define a surface. Future work could potentially relax this requirement by generalizing beyond Voronoi diagrams. Other open research questions include how to compose multiple foam models together efficiently and accounting for varying illumination, how to model dynamic content instead of static scenes, how to enable editing of scenes, and how to integrate generative modeling with our representation. Progress in these directions could make foam models relevant in real-time ray tracing applications currently dominated by triangle meshes, as we have already found that foam-based ray tracing can exceed the performance of dedicated ray tracing hardware.

Acknowledgements. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant [2023-05617], NSERC Collaborative Research and Development Grant, the SFU Visual Computing Research Chair, Google Research, Digital Research Alliance of Canada, and Advanced Research Computing at the University of British Columbia. We would like to thank George Kopanas, Lily Goli, Alex Evans, Thomas Muller, Bernhard Kerbl, Vincent Sitzmann, Forrester Cole, Or Litany, and David Fleet for their feedback and/or early research discussions.

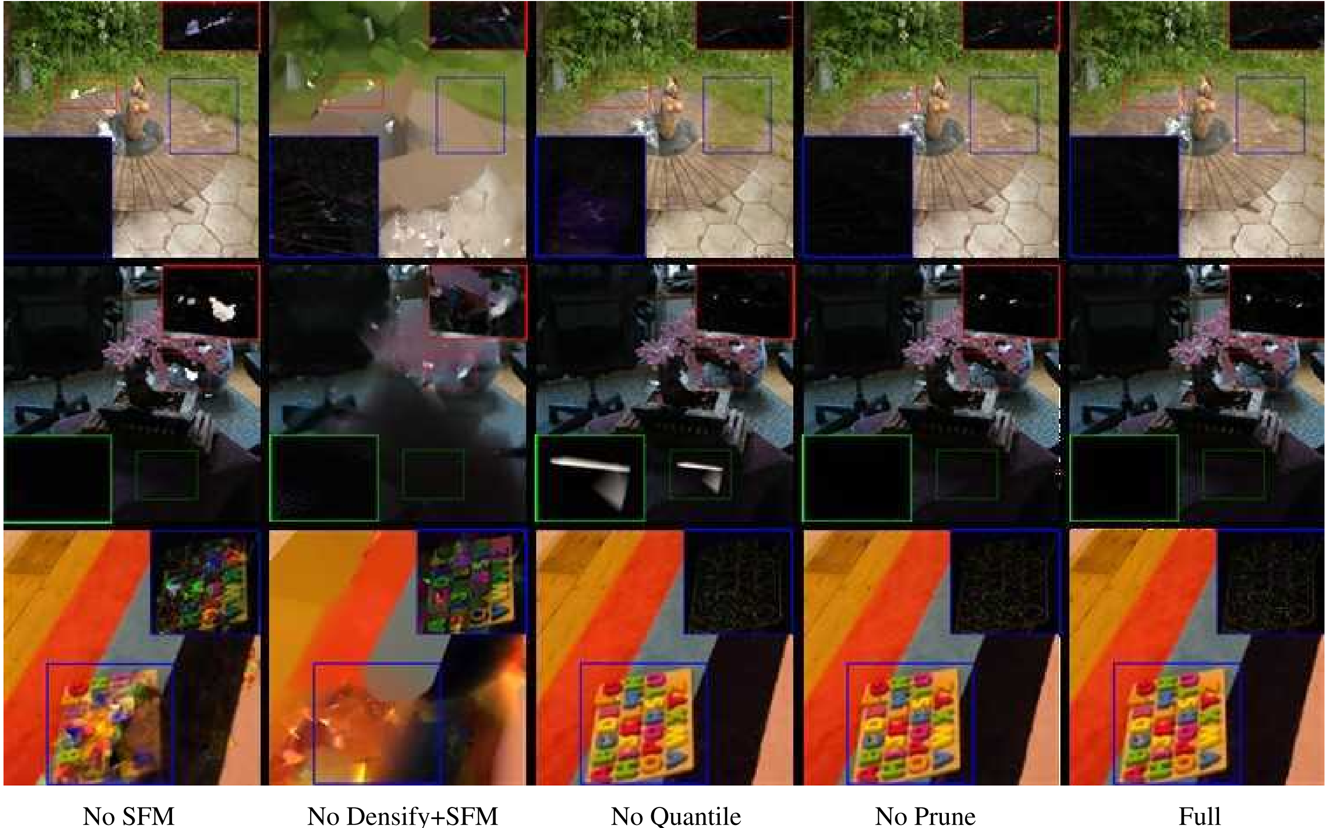

Figure 8. **Ablations –** We qualitatively analyze the impact of excluding various components from our method. For better visualization, we zoom in on specific image regions and present their corresponding error maps. *No SFM*: While the method performs relatively well without initialization, it degrades in sparsely covered regions of the training views. *No Densify+SFM*: Without densification, the method significantly underperforms, leading to under-represented scenes. *No Quantile*: Disabling the quantile loss introduces floaters, degrading rendering quality. *No Pruning*: Ceasing point pruning has minimal impact due to the low number of prunable points.

References

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

[3] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Zip-nerf: Anti-aliased grid-based neural radiance fields. *ICCV*, 2023.

[4] Samuel Rota Bulò, Lorenzo Porzi, and Peter Kontschieder. Revising densification in gaussian splatting, 2024.

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

[6] Boris Delaunay. Sur la sphere vide. *Bulletin of the Academy of Sciences of the USSR Classe des Sciences Mathematiques et Naturelles*, 1934.

[7] 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 (TOG)*, 43(4):1–13, 2024.

[8] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps. *arXiv preprint arXiv:2311.17245*, 2023.

[9] Jian Gao, Chun Gu, Youtian Lin, Hao Zhu, Xun Cao, Li Zhang, and Yao Yao. Relightable 3d gaussian: Real-time point cloud relighting with brdf decomposition and ray tracing. *arXiv preprint arXiv:2311.16043*, 2023.

[10] Chun Gu, Zeyu Yang, Zijie Pan, Xiatian Zhu, and Li Zhang. Tetrahedron splatting for 3d generation. In *NeurIPS*, 2024.

[11] Antoine Guédon and Vincent Lepetit. Sugar: Surface-

aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5354–5363, 2024.

[12] Leonidas Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi. *ACM Trans. Graph.*, 4(2):74–123, 1985.

[13] 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.

[14] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2d gaussian splatting for geometrically accurate radiance fields. In *ACM SIGGRAPH 2024 Conference Papers*, New York, NY, USA, 2024. Association for Computing Machinery.

[15] Letian Huang, Jiayang Bai, Jie Guo, Yuanqi Li, and Yanwen Guo. On the error analysis of 3d gaussian splatting and an optimal projection strategy. *arXiv preprint arXiv:2402.00752*, 2024.

[16] Yingwenqi Jiang, Jiadong Tu, Yuan Liu, Xifeng Gao, Xiaoxiao Long, Wenping Wang, and Yuexin Ma. Gaussianshader: 3d gaussian splatting with shading functions for reflective surfaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5322–5332, 2024.

[17] Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. Harmonic coordinates for character articulation. *ACM transactions on graphics (TOG)*, 26(3):71–es, 2007.

[18] James T Kajiya. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, 1986.

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

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

[21] 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. *NeurIPS*, 2024.

[22] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[23] Jonas Kulhanek and Torsten Sattler. Tetra-nerf: Representing neural radiance fields using tetrahedra. In *ICCV*, 2023.

[24] Christoph Lassner and Michael Zollhofer. Pulsar: Efficient sphere-based neural rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1440–1449, 2021.

[25] Zhihao Liang, Qi Zhang, Ying Feng, Ying Shan, and Kui Jia. Gs-ir: 3d gaussian splatting for inverse rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21644–21653, 2024.

[26] Gerd Marmitt and Philipp Slusallek. Fast ray traversal of tetrahedral and hexahedral meshes for direct volume rendering. In *Proceedings of the Joint Eurographics/IEEE VGTC conference on Visualization*, pages 235–242, 2006.

[27] Nelson Max and Min Chen. Local and global illumination in the volume rendering integral. Technical report, Lawrence Livermore National Lab. (LLNL), Livermore, CA (United States), 2005.

[28] 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.

[29] 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. *ACM Transactions on Graphics and SIGGRAPH Asia*, 2024.

[30] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *SIGGRAPH*, 2022.

[31] Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann. Compressed 3d gaussian splatting for accelerated novel view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10349–10358, 2024.

[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.org*, 2024.

[33] 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):1–17, 2024.

[34] Hanspeter Pfister, Matthias Zwicker, Jeroen Van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 335–342, 2000.

[35] Lukas Radl, Michael Steiner, Mathias Parger, Alexander Weinrauch, Bernhard Kerbl, and Markus Steinberger. Stopthepop: Sorted gaussian splatting for view-consistent real-time rendering. *ACM Transactions on Graphics (TOG)*, 2024.

[36] Daniel Rebain, Wei Jiang, Soroosh Yazdani, Ke Li, Kwang Moo Yi, and Andrea Tagliasacchi. Derf: Decomposed radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14153–14161, 2021.

[37] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 14335–14345, 2021.

[38] Christian Reiser, Stephan Garbin, Pratul Srinivasan, Dor Verbin, Richard Szeliski, Ben Mildenhall, Jonathan Barron, Peter Hedman, and Andreas Geiger. Binary opacity grids: Capturing fine geometric detail for mesh-based view synthesis. *ACM Transactions on Graphics (TOG)*, 43(4):1–14, 2024.

[39] Kerui Ren, Lihan Jiang, Tao Lu, Mulin Yu, Linning Xu, Zhangkai Ni, and Bo Dai. Octree-gs: Towards consistent real-time rendering with lod-structured 3d gaussians. *arXiv preprint arXiv:2403.17898*, 2024.

[40] Liu Ren, Hanspeter Pfister, and Matthias Zwicker. Object space ewa surface splatting: A hardware accelerated approach to high quality point rendering. In *Computer Graphics Forum*, pages 461–470. Wiley Online Library, 2002.

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

[42] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[43] Otto Seiskari, Jerry Ylilammi, Valtteri Kaatrasalo, Pekka Rantalankila, Matias Turkulainen, Juho Kannala, Esa Rahtu, and Arno Solin. Gaussian splatting on the move: Blur and rolling shutter compensation for natural camera motion. In *European Conference on Computer Vision*, pages 160–177. Springer, 2025.

[44] Gopal Sharma, Daniel Rebain, Kwang Moo Yi, and Andrea Tagliasacchi. Volumetric rendering with baked quadrature fields. *ECCV*, 2024.

[45] Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. *NeurIPS*, 2021.

[46] Jonathan Richard Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry*, 22(1):21–74, 2002. 16th ACM Symposium on Computational Geometry.

[47] Andrea Tagliasacchi and Ben Mildenhall. Volume rendering digest (for nerf), 2022.

[48] Georges Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 1908.

[49] 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 preprint arXiv:2106.10689*, 2021.

[50] 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. *arXiv preprint arXiv:2311.10091*, 2023.

[51] D. F. Watson. Computing the n-dimensional delaunay tessellation with application to voronoi polytopes. *The Computer Journal*, 24(2):167–172, 1981.

[52] M. Weiler, M. Kraus, M. Merz, and T. Ertl. Hardware-based view-independent cell projection. *IEEE Transactions on Visualization and Computer Graphics*, 9(2): 163–175, 2003.

[53] Turner Whitted. An improved illumination model for shaded display. In *ACM Siggraph 2005 Courses*, pages 4–es. 2005.

[54] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. *Advances in Neural Information Processing Systems*, 34:4805–4815, 2021.

[55] Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. Mip-splatting: Alias-free 3d gaussian splatting. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[56] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. Surface splatting. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, page 371–378, New York, NY, USA, 2001. Association for Computing Machinery.

	3DGS [19]	Mip-Splatting [55]	3DGS-MCMC [21]	Plenoxels [41]	iNGP-Big [30]	MipNerf360 [2]	Ours
	PSNR↑ / SSIM↑ / LPIPS↓	PSNR↑ / SSIM↑ / LPIPS↓	PSNR↑ / SSIM↑ / LPIPS↓	PSNR↑ / SSIM↑ / LPIPS↓	PSNR↑ / SSIM↑ / LPIPS↓	PSNR↑ / SSIM↑ / LPIPS↓	PSNR↑ / SSIM↑ / LPIPS↓
Room	30.63 / 0.91 / 0.27	31.74 / **0.93** / 0.27	**32.30** / **0.93** / **0.25**	27.59 / 0.84 / 0.42	29.69 / 0.87 / 0.26	**31.63** / **0.91** / 0.21	30.87 / **0.91** / **0.19**
Counter	28.70 / 0.91 / 0.24	**29.16** / **0.92** / 0.24	**29.16** / 0.91 / **0.23**	23.63 / 0.76 / 0.44	26.69 / 0.82 / 0.31	**29.55** / **0.89** / 0.20	28.59 / 0.88 / **0.19**
Bonsai	31.98 / 0.94 / 0.24	32.31 / **0.95** / **0.23**	**32.67** / **0.95** / **0.23**	24.67 / 0.81 / 0.40	30.69 / 0.91 / 0.21	**33.46** / **0.94** / 0.18	32.15 / 0.93 / **0.17**
Kitchen	30.32 / 0.92 / **0.14**	31.55 / 0.93 / 0.15	**32.23** / **0.94** / **0.14**	23.42 / 0.65 / 0.45	29.48 / 0.86 / 0.20	**32.23** / **0.92** / **0.13**	31.40 / 0.91 / **0.13**
Bicycle	25.25 / 0.77 / 0.23	25.72 / **0.78** / **0.19**	**26.06** / **0.78** / **0.19**	21.92 / 0.50 / 0.51	22.17 / 0.51 / 0.45	**24.37** / **0.69** / **0.30**	24.19 / 0.68 / 0.31
Garden	27.41 / 0.87 / 0.12	27.76 / **0.88** / **0.11**	**27.99** / 0.87 / **0.11**	23.49 / 0.61 / 0.39	25.07 / 0.70 / 0.26	**26.98** / 0.81 / 0.17	26.58 / **0.82** / **0.16**
Stump	26.55 / 0.78 / 0.24	26.94 / **0.79** / 0.21	**27.67** / 0.78 / **0.20**	20.66 / 0.52 / 0.50	23.47 / 0.59 / 0.42	**26.40** / **0.74** / **0.26**	25.48 / 0.71 / 0.29

Table 3. PSNR, SSIM, and LPIPS scores for Mip-NeRF360 [2] scenes.

	3DGS [19]	Mip-Splatting [55]	3DGS-MCMC [21]	Plenoxels [41]	iNGP-Big [30]	MipNerf360 [2]	Ours
	PSNR↑ / SSIM↑ / LPIPS↓	PSNR↑ / SSIM↑ / LPIPS↓	PSNR↑ / SSIM↑ / LPIPS↓	PSNR↑ / SSIM↑ / LPIPS↓	PSNR↑ / SSIM↑ / LPIPS↓	PSNR↑ / SSIM↑ / LPIPS↓	PSNR↑ / SSIM↑ / LPIPS↓
Dr Johnson	28.77 / **0.90** / 0.33	28.76 / **0.90** / **0.32**	**29.05** / 0.89 / 0.33	23.14 / 0.79 / 0.52	28.26 / 0.85 / 0.35	**29.14** / **0.90** / **0.24**	28.33 / /0.88 / 0.27
Playroom	30.04 / **0.91** / 0.32	30.17 / **0.91** / 0.33	**30.37** / 0.90 / **0.31**	22.98 / 0.80 / 0.50	21.67 / 0.78 / 0.43	**29.66** / **0.90** / **0.25**	29.56 / 0.89 / 0.26

Table 4. PSNR, SSIM, and LPIPS scores for Deep Blending [13] scenes.

6. Per Scene metrics

Tables 3 and 4 summarize the error metrics collected for our evaluation of all considered techniques. These include results for both Mip-NeRF360 [2] and Deep Blending [13] scenes. However, 3DGRT [29] is excluded from per-scene comparisons as these values are not reported in the original paper, and the code is not publicly available.