Eurographics Symposium on Rendering (2025)
B. Wang and A. Wilkie (Editors)

Content-Aware Texturing for Gaussian Splatting

Panagiotis Papantonakis[1,2], Georgios Kopanas†[3,4], Frédo Durand[5] and George Drettakis[1,2]

[1] Inria [2] Université Côte D'Azur [3] Google [4] Runway ML [5] MIT CSAIL

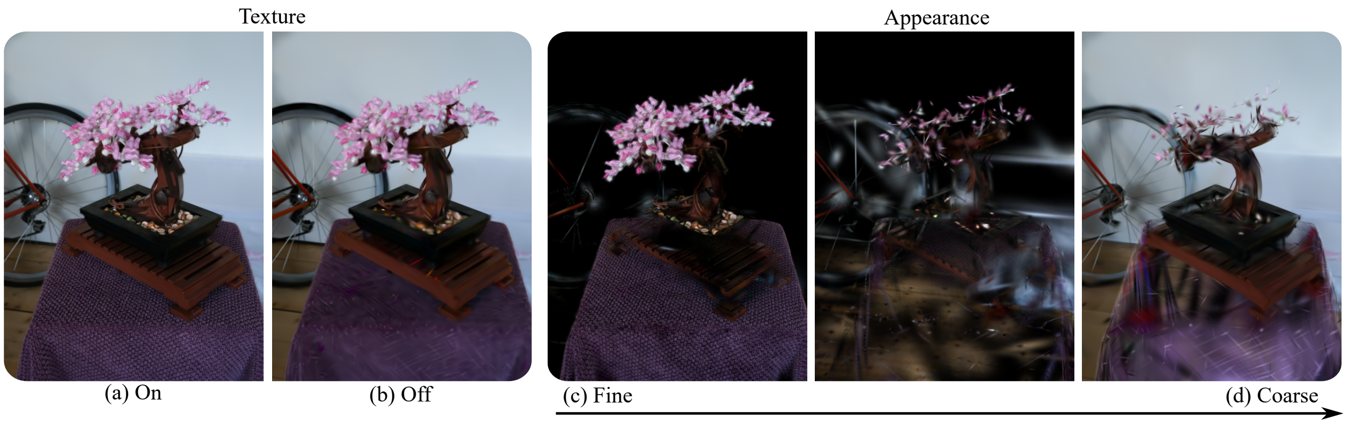

Figure 1: *We propose a content-aware texturing method for 2D Gaussian Splatting. Our textures reconstruct intricate scene detail (a). Gaussian primitives reconstruct the shape of the scene at low frequency of appearance; we show this in (b) where texturing is disabled. Our method is adaptive, allowing different primitives to have different texel sizes, depending on scene content. On the right hand panel we display primitives with progressively higher* texel-to-pixel *ratio. In regions with high frequency appearance, texels have size close that of pixels (e.g., the table cover (c)). For the low-frequency walls however (d), the ratio is high, with each texel representing a large number of input image pixels.*

Abstract

Gaussian Splatting has become the method of choice for 3D reconstruction and real-time rendering of captured real scenes. However, fine appearance details need to be represented as a large number of small Gaussian primitives, which can be wasteful when geometry and appearance exhibit different frequency characteristics. Inspired by the long tradition of texture mapping, we propose to use texture to represent detailed appearance where possible. Our main focus is to incorporate per-primitive texture maps that adapt to the scene in a principled manner during Gaussian Splatting optimization. We do this by proposing a new appearance representation for 2D Gaussian primitives with textures where the size of a texel is bounded by the image sampling frequency and adapted to the content of the input images. We achieve this by adaptively upscaling or downscaling the texture resolution during optimization. In addition, our approach enables control of the number of primitives during optimization based on texture resolution. We show that our approach performs favorably in image quality and total number of parameters used compared to alternative solutions for textured Gaussian primitives.

1. Introduction

Gaussian Splatting [KKLD23; HYC*24] has become the method of choice for the capture and real-time novel-view synthesis of real scenes. It relies on flexible, easy-to-optimize *Gaussian primitives* to represent the scene. However, in the standard approach, fine visual appearance – such as intricate texture on a surface – needs to be represented with a large number of very small primitives even when the geometric complexity is low. This wasteful limitation arises because each Gaussian primitive is associated with a single color sample, preventing the decoupling of appearance and geometric complexity. Rendering techniques, on the other hand, have traditionally

† work done at Google

built on *texture mapping* to represent such detailed appearance. We build on this tradition to propose a representation for *textured* Gaussian primitives that is guided by the scene and its content.

In recent and concurrent work [SMAB25; RCB*25; CTP*25; XCW*24] a number of methods introduce textured Gaussian primitives, providing a spatially varying color. Most of these methods use planar 2D Gaussians to match the dimensionality of traditional textures. Each primitive carries several parameters (position, rotation, scale, opacity and spherical harmonics (SH)), while textures are 2D arrays of RGB(A) texels. The fundamental challenge of such approaches is how to allocate model capacity across the scene and between the number of texels and the number of primitives used. Previous and concurrent solutions either deal with the issue by imposing a fixed texture resolution per primitive [SMAB25; XCW*24; CTP*25], or propose heuristic approaches, that are far from ideal [RCB*25]. This leaves open the question of how to share resources in a more principled manner.

To answer this question, we propose a *content-aware* texturing solution for 2D Gaussian primitives that uses the appearance complexity as observed from the input images of the scene to make informed decisions on how parameter resources are distributed. We do this by first introducing a textured Gaussian primitive representation in which texel size is *fixed* in world space. This makes the appearance texture of each primitive independent of its size, and therefore unaffected by the growing or shrinking that it undergoes during optimization. This representation separates the parameters of geometry and appearance, allowing us to refine them independently with appropriate allocation of memory capacity. More specifically, we introduce a method to increase and decrease the texel size as optimization evolves, by analyzing the frequency content that primitive textures can represent and determining the corresponding error. We pair that with a resolution-aware method to control the overall number of primitives, striking a balance between texture size and number of primitives. Our solution interacts closely with the optimization, using the downscale/upscale mechanism to reduce the error in *appearance*, while our control of the number of primitives handles *geometric* error by spawning additional Gaussians to capture geometric detail.

In summary, we propose three contributions:

- A texture representation for 2D Gaussians that defines texels with a *fixed* world space size, supporting visual reconstruction independent of the shape of the primitives.
- A progressive algorithm that adaptively determines texel size and allows for content-aware fitting of the scene.
- A resolution-based solution to control the number of primitives, adjusted to the textured representation.

Our experiments show that our method provides a good balance between visual quality and the number of parameters used for the representation, comparing favorably to other textured Gaussian primitives solutions proposed in recent and concurrent work.

2. Related Work

2.1. Traditional Representations

Texture Mapping was introduced in the early days of CG [Cat74] and is widely used to effectively map detailed appearance information onto a simpler geometric representation, such as triangles. Textures are stored as 2D image arrays with a mapping function between the 3D surface and 2D texture coordinates. Computing this 2D surface parameterization is a well-studied and difficult problem [HLS07; LKK*18; SGV*24]. Intrinsic 2D parameterizations pose challenges even in traditional graphics pipelines [YLT19] because they introduce difficulties in content creation and produce visual artifacts in rendering. In the context of optimization for inverse problems – such as novel view synthesis – intrinsic parameterizations are also challenging since the geometry is not known in advance [SGV*24]. An alternative, non-parametric way to store localized information on the surface is by attaching it on the primitives themselves, i.e., attaching colors to the vertices of a mesh, requiring a very fine mesh subdivision to represent details. Mesh Colors [YKH10; MSY20] extend this idea by using more color samples along the edges and the faces of the triangles. This significantly improves the representational capacity of non-parametric appearance textures. While non-parametric appearance models are limited in many ways, they have significant advantages in the context of optimization since they overcome the need to jointly solve for texture parameterization and the surface. Our representation is also a non-parametric texture representation for Gaussian Splatting.

2.2. Representations for 3D Reconstruction

Scene reconstruction and novel view synthesis has recently utilized differentiable rendering to recover 3D representations from a set of images or video. Early methods utilized Multi-Plane Images (MPIs) [MSO*19; ZTF*18], a collection of planes with RGBA textures that are re-projected and rendered very efficiently using homography transformations. The semi-transparent and planar nature of the RGBA textures allowed the optimization to converge to useful 3D representations, but the planar assumption for the geometry is insufficient in most realistic cases. Neural Radiance Fields (NeRF [MST*21]) popularized volumetric representations in the context of 3D reconstruction by introducing an Multi-Layer Perceptron (MLP) that stores volume density and color. NeRF's success resulted in follow-up work that extended it to support anti-aliasing and unbounded scenes [BMT*21; BMV*22; ZRSK20], improve its training and rendering speed [RPLG21; SSC22; MESK22; CXG*22; FYT*22; BMV*23; WSN*23; SRYT24], increase its robustness when using fewer views [YYTK21; NBM*22], and better reconstruct surfaces and handle reflections [VHM*22; LME*23; YHR*23]. Most closely related to our work is Nuvo [SGV*24] that recovers a 2D parameterization of a volume. For a more complete survey we refer readers to [TTM*22].

Gaussian Splatting [KKLD23] has emerged as an alternative, point-based approach to NeRFs by replacing the volumetric field with a collection of semi-transparent ellipsoidal primitives. This technique achieves excellent quality and real-time rendering even at high resolutions. Several methods built on top of 3DGS to provide anti-aliasing [YCH*24], by changing the appearance model but are constrained by the 1-to-1 relationship between color samples and primitives [YGS*24; MST*25].

2D Gaussian Splatting [HYC*24] flattens one of the dimensions of the ellipsoids to create planar discs or *surfels* to align better with surfaces. Surfels are parametrized by the set of parameters $A = \{\mu, \sigma, \mathbf{q}, o, \mathbf{SH}\}$, where $\mu \in \mathbb{R}^3$ is the primitive's center, $\sigma \in \mathbb{R}^{+2}$ are the scales of the primal axes of the surfel, $\mathbf{q} \in \mathbb{R}^4$ is a quaternion that represents the rotation $\mathbf{R}$, o is the opacity and $\mathbf{SH}$ are the spherical harmonics coefficients used to get the view-dependent colour $\mathbf{c}$. The normal vector $\mathbf{n}$ of the surfel is the third column of the rotation matrix $\mathbf{R}$.

With this formulation, the intersection point between a ray $\mathbf{r} = \mathbf{r_0} + t\mathbf{d}$ and a primitive can be computed as:

$$\mathbf{p} = \mathbf{r}_0 + t\mathbf{d}, t = \frac{\mathbf{n} \cdot (\mu - \mathbf{r}_0)}{\mathbf{n} \cdot \mathbf{d}} \tag{1}$$

We also build on 2D Gaussian Splatting for our method.

The initial primitive-based representations were not as compact as NeRFs, but a number of recent results allow competitive compression strategies [BKL*24]. Several recent methods build on traditional CG solutions, demonstrating that disentangling appearance from geometry can achieve significant improvements in terms of storage and memory costs. Texture-GS [XHL*24] combines deferred shading and a per-pixel UV coordinate to fetch appearance from a texture. This assumes that the objects can be represented by a sphere imposing topological constraints. Our results demonstrate that non-parametric texture mapping is a convenient and flexible way to recover texture during the optimization.

Recent and concurrent work [SMAB25; CTP*25; XCW*24; RCB*25] have used textures to represent fine visual details. [RCB*25] and [CTP*25] use converged 2DGS and 3DGS point clouds, respectively, as a starting point of their method, and then proceed to add and optimize texture parameters. Several methods [SMAB25; CTP*25; XCW*24] use a fixed texture resolution for each primitive that is a hyperparameter of the method. GS-Tex [RCB*25] distributes a texel budget over the primitives proportional to their size, resulting in different resolutions for different-sized primitives. All these methods define their textures in the Gaussian canonical space, with the textures undergoing the same transformations as the primitive, which leads to texture stretching and shrinking. SuperGaussians [XCW*24] also experiment with representing the texture with a small Neural Network. In our approach, we explore a different and more robust design that allows for texture maps to dynamically adjust to the captured content. Along with two strategies that control the size of the texels and the number of primitives that are designed specifically for our representation, we are able to reconstruct scenes without the need for a pretrained model.

3. Method

We first define a new representation for appearance of the Gaussian primitives that allows for content-aware texturing. We then show how to adapt the texturing process to scene content, by taking into account the different screen-space frequencies that need to be represented using textures. This is achieved in a progressive manner that is compatible with the optimization process. Finally, we propose a resolution-based primitive management method that splits primitives allowing the geometry and appearance of the scene to be approximated as required.

In contrast to the original 2D Gaussian Splatting approach [HYC*24] (see Sec. 2.2 and Eq. 1), we express the intersection between a camera ray and a primitive in different coordinate systems, denoted by a superscript. This is done, as some coordinate systems make some operations easier. Specifically:

- $\mathbf{p}^w = \mathbf{p}$ is the intersection point in world space,
- $\mathbf{p}^{w_0} = \mathbf{p}^w - \mu$ is in world space, centered on the primitive,
- $\mathbf{p}^l = \mathbf{R}^{-1}\mathbf{p}^{w_0}$ is in the local, axis-aligned coordinate system of the primitive. Since the last component of this point is 0, we consider that $\mathbf{p}^l \in \mathbb{R}^2$
- $\mathbf{p}^c = \mathbf{S}^{-1}\mathbf{p}^l$, where $\mathbf{S} = \text{diag}(\sigma)$ is in the local, normalised coordinate system of the primitive.

This last coordinate system can be considered as the "canonical" space of the Gaussian primitive. The use of these spaces is purely to facilitate some operations. For instance, Eq. 1 is evaluated in the camera view space, where $\mathbf{r}_0$ is on top of the origin, while anything that involves querying the texture map is better done in a coordinate frame local to the respective primitive.

The color of a ray $\mathbf{r}$ is computed by alpha blending:

$$\mathbf{C}(\mathbf{r}) = \sum_i w_i(\mathbf{p}_i)\mathbf{c}_i(\mathbf{d}) = \sum_i T_i o_i G_i(\mathbf{p}_i)\mathbf{c}_i(\mathbf{d}) \tag{2}$$

$$T_i = \prod_j^{i-1}(1 - o_j G_j(\mathbf{p}_j)) \tag{3}$$

where $w_i(\mathbf{p}_i)$ is the contribution of the Gaussian, $G(\mathbf{x})$ is the evaluation of the Gaussian function that defines the falloff $G(\mathbf{x}) = e^{-\frac{1}{2}(\mathbf{x}^T\mathbf{x})}$, and T_i is transmittance.

3.1. Content-Aware Textures for Gaussian Splats

We augment each 2D Gaussian primitive with a texture map $\mathcal{T}$. The texture map adds a spatially varying *offset* $\mathbf{c}^{\mathcal{T}}$ to the original spherical harmonic color representation. The texture colors vary only spatially and have no directional dependency. This representation is compact and models only the diffuse properties of the surface, akin to albedo multiplied by irradiance. As a result, the color of each primitive is now also dependent on the intersection point between the ray emanating from the pixel and the 2D primitive:

$$\mathbf{c}_i = \mathbf{SH}(\mathbf{d}) + \text{bilerp}(\mathcal{T}_i, \mathbf{u}) \tag{4}$$

where $\mathbf{u}$ are the coordinates in UV space at which the ray intersects primitive i with a direction vector $\mathbf{d}$, and $\mathbf{SH}$ are the spherical harmonics. We need to carefully design the texture coordinate calculation that transforms the intersection point into local, normalized primitive space $\mathbf{p}^c$ to UV coordinates. As we discuss below, it is especially important to consider how this mapping changes when the primitive parameters are updated during optimization.

A simple mapping from the canonical space to UV texture space fixes the relative texture coordinates on the primitive such that the texture will stretch and deform with scaling and rotation of the primitive:

$$\mathbf{u} = \left(\frac{\mathbf{p}^c}{2s_i} + 0.5\right) \cdot \mathcal{T}_{\text{res}} \tag{5}$$

where s_i is the extent of the texture in units of standard deviations of the Gaussian primitive, and T_{res} is the texture resolution. When the extent of the texture is smaller than the primitive, some type of padding needs to be applied.

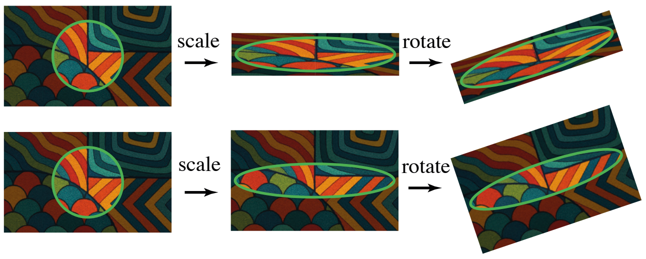

Figure 2: *Difference between two mappings for a primitive having learned a particular texture. Top row: a naive approach distorts the appearance after the primitive undergoes scaling. Bottom row: in our approach, as texel size is fixed in world space, scaling the primitive only reveals more part of the underlying texture, preserving existing content.*

Recall that primitives change size and shape during the optimization process. As a result, with such a texture mapping, the appearance parameters of the texture are coupled with the parameters that control the shape of the primitive. During optimization, small changes in the scale of the Gaussians result in changes of all the values of the texture. This can induce local minima in the optimization process that are visible as texture stretching, see Fig. 2 (top).

Instead, we define a *content-aware* representation in which textures are adapted to scene content. Our goal is to have textures that can faithfully represent the frequency of image details at the highest resolution present in the input images. To do this, we fix the texel size with respect to the optimization process such that no optimizable parameters change it. We achieve this with a small modification to Eq. 5:

$$\mathbf{u} = \frac{\mathbf{p}^l}{k_i} + T_{\text{offset}} \tag{6}$$

where k_i is the texel size and T_{offset} is an offset to center the texture map. In constrast to Eq. 5, the intersection point $\mathbf{p}^l$ and hence the texel size k_i are defined in world space, with the same units as the primitive's scales σ.

This mapping implies that the textures do not have a fixed texture resolution. On the contrary, as the primitives grow or shrink, more or less texture resolution is needed to cover their surface. This is demonstrated in Fig. 2 (bottom). We modified our optimization routines to allocate and de-allocate texture resolution dynamically.

Since the number of texels can grow quadratically, we risk running out of resources if every primitive has a texture. Our goal is to maintain an expressive representation while carefully managing resources. To achieve this, we enforce two properties on our representation: First, the projected sampling frequency of the texture needs to be bounded by the image sampling frequency of the closest camera. This means that the projected texel size should never be smaller than the smallest input pixel that can see it. Second, texel sizes should adapt to *scene content*, i.e., smaller texel sizes should be allocated for high frequency image content.

The first goal can be achieved using a conservative choice to determine minimum texel size as the pixel size back-projected in world space from the input training view closest to the primitive's center k^p_{min}, similarly to [PKK*24].

Regarding the second goal, optimal texel sizes cannot be determined at initialization since pritimives change size and rotate dynamically during optimization. We need a way to adapt to the freqency content of the scene progressively. To do this, we next introduce an adaptive strategy to determine texel size during optimization, where we downscale and upscale texture by adapting to the scene *content*. We complement this approach with a resolution-aware primitive management method that add primitives to represent geometric – rather than appearance – error. We describe these two components in the following sections.

3.2. Progressive Adaptive Texel Size Determination

Our goal is to adapt textures to the frequency content of the input images. We also need to achieve this in a progressive manner that fits well with the optimization process. We define a *texel size-to-pixel* size ratio t_2p_r, which we manipulate during optimization using downscaling and upscaling operations.

We define the texel size k based on the minimum pixel size k^p_{min}, with t_2p_r as follows:

$$k = k^p_{\text{min}} \cdot t_2p_r \tag{7}$$

We use this parameter to assign a low t_2p_r to primitives in regions with fine details. As a result, such primitives will have more texels for a given primitive size. Similarly, primitives that lie in areas with low-frequency appearance will have a higher ratio, giving them relatively fewer texels. To provide a lower bound of 1 and facilitate texture rescaling operations, t_2p_r can only take values that are powers of two.

To this end, we propose two strategies to increase and decrease t_2p_r leading to a downscale and an upscale of the texture map respectively. Note that texel size and texture resolution are linked, but not the same. In the operations below, when texel size changes, texture resolution will change but only because the *primitive size is unchanged*.

Increase of texel size-to-pixel size ratio - Downscaling: Our goal is to find texture maps that represent low frequency details and decrease their texel size accordingly. We do this by applying a low-pass filter on textures and then comparing them to the originals. The error $\mathcal{E}_d$ is weighted by the Gaussian falloff of the primitive, leading to the following equation:

$$\mathcal{E}_d = \frac{1}{\sum_{\mathbf{p}} G(\mathbf{p})} \sum_{\mathbf{p}} G(\mathbf{p})(T_{\text{orig}}(\mathbf{p}) - T_{\text{lowpass}}(\mathbf{p})) \tag{8}$$

If this error is less than a threshold τ_{ds}, we assume that the texture map can be reconstructed with good-enough fidelity by the downscaled version. Hence, we double t_2p_r, which reduces the total number of texture parameters by 4. This reduces the resolution

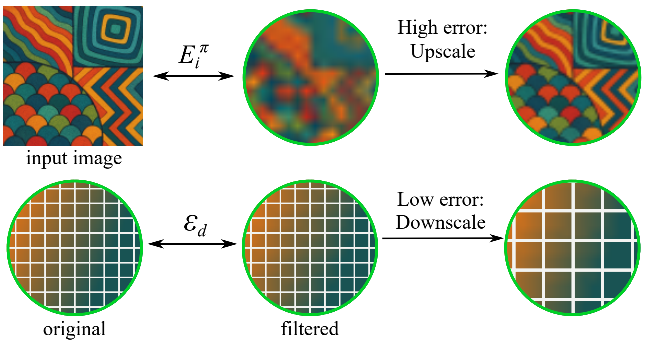

Figure 3: *We illustrate the downscale and upscale process used.*

of texture, since the world-space size of the texels has grown. The size of the primitive is however unchanged.

Decrease of texel size-to-pixel size ratio - Upscaling: In regions where the texel size of the primitives involved is too large to capture the fine, underlying details, we expect our images to be blurry, and thus induce large error. To identify these regions, we estimate primitive error using the approach presented in [RPK24]. Specifically, for a primitive i, a ray $\mathbf{r}$, a camera view π and corresponding RGB error image $\mathcal{E}_\pi$, we compute the per primitive error for a single view:

$$E_i^\pi = \sum_{\mathbf{r} \in \mathcal{P}_i} \mathcal{E}_\pi(\mathbf{r}) w_i^\pi(\mathbf{r}) \tag{9}$$

where $\mathcal{P}_i$ are all the pixels covered by the primitive. Here we assume that we have one ray per pixel, emanating from its center. Differently from [RPK24], instead of taking the maximum error over views, we perform a weighted sum, with the weight being the total contribution of a primitive in that image.

$$E_i = \frac{\sum_{\pi \in \Pi} E_i^\pi \overline{w_i^\pi}}{\sum_{\pi \in \Pi} \overline{w_i^\pi}}, \quad \overline{w_i^\pi} = \sum_{\mathbf{r} \in \mathcal{P}_i} w_i^\pi(\mathbf{r}) \tag{10}$$

where Π is the set of all input views. This choice assigns an error value to each primitive that is proportional to their contribution to the rendered image.

We choose the top 10% of primitives with the highest error, and upscale their textures by a factor of 4, by halving $t_2 p_r$. With smaller sized texels, these primitives can better fit to the scene content, reducing their error.

Progressive texel size adaptation. The calculation of the per primitive error E_i and the application of upscale/downscale process happens regularly during optimization. Texel size adapts to scene content during this process, getting bigger for primitives that lie in low-frequency regions, and smaller for primitives that exhibit large error, as shown in Fig. 1(right).

There are two main sources of error: appearance and geometry. For the first case, our upscaling approach reduces error as the optimization progresses. However, for the case of geometry that is not well represented, an additional step is required. To address this and complete our content-aware method, we next introduce resolution-aware splitting.

3.3. Resolution-aware Primitive Management

In Gaussian splatting methods where each primitive contains one color sample, *densification* – i.e., adding new primitives through cloning and splitting – plays an essential role in improving the reconstruction of the scene. This results in a local increase in both the geometric (position, scale, rotation) and appearance parameters (SH, opacity) that are treated together since they are tied to a single Gaussian primitive. However, this is not always ideal, since there are cases where the geometry is low frequency and has been approximated well, but we are missing texture detail, or conversely, cases where the color information is low frequency, but the underlying surface is not correctly represented. The latter case can appears as holes, "softened" edges or elongated primitives in places where they are not required.

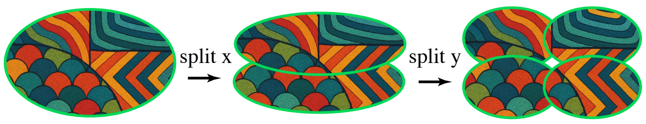

Figure 4: *Left to right:The primitive has been upscaled to a resolution greater than τ_{tr} in both axes. Our splitting approach creates four new primitives, each with half the scale and texture resolution in both axes.*

Our separation of geometric and appearance parameters provides an additional degree of freedom, allowing us to independently decide whether to increase the number of parameters linked to *geometry*, i.e., increasing the number of primitives or to *appearance*, i.e., increasing texture resolution.

Given that we do not have a supervision signal explicitly for geometry, we attempt to first match the appearance, using the upscaling approach described above. If the error is still high despite upscaling, we interpret the remaining error as geometric. In these cases, we can add more primitives, locally increasing the geometric degrees of freedom. We found that densification strategies that were based on cloning either partially [KKLD23] or entirely in the case of 3DGS-MCMC [KRS*24], are incompatible with our representation. This is because the superimposition of multiple textured primitives makes convergence difficult and requires an excessive number of parameters. As a result our primitive management is based on *splitting*, which fits well with our method. We describe this next.

Similar to the approach for upscaling, we take the top 10% of primitives with the highest error, and check which of these exceed a threshold τ_{tr} for texture resolution. The primitive is replaced by two new primitives, each displaced by ± 1 standard deviation of the Gaussian. This process is performed separately for each axis; the new primitives have half the scale (size) in each corresponding axis, as a result, half the texture resolution, $G(1)$ times the opacity, where G is the Gaussian. The texture map of the newly added primitives is created by sampling the original point at the respective locations. Fig. 4 illustrates the splitting process for a primitive that happens to have a texture resolution greater than τ_{tr} in both axes. A flowchart of how upscaling and splitting interact is shown in Fig. 5. Alg. 1 displays in high level how the two methods are integrated in code.

	DeepBlending							Mip-Nerf-360							Tanks&Temples						
	SSIM↑	PSNR↑	LPIPS↓	Points	Texels	Params	FPS	SSIM↑	PSNR↑	LPIPS↓	Points	Texels	Params	FPS	SSIM↑	PSNR↑	LPIPS↓	Points	Texels	Params	FPS
3DGS-MCMC	0.903	29.81	0.311	1323K	0.0M	78.1M	265	0.831	27.82	0.232	2587K	0.0M	152.6M	103	0.855	24.25	0.211	695K	0.0M	41.1M	230
2DGS*	0.899	29.52	0.324	1444K	0.0M	83.8M	96	0.801	27.18	0.282	2079K	0.0M	120.6M	91	0.834	23.37	0.239	872K	0.0M	50.6M	165
BBSplat	0.898	29.25	0.318	160K	41.0M	173.0M	27	0.781	26.67	0.273	237K	60.9M	257.0M	23	0.848	23.62	0.178	300K	76.8M	324.3M	38
GSTex	0.906	29.63	0.323	1503K	10.0M	117.2M	21	0.802	27.06	0.285	2025K	10.0M	147.5M	20	0.842	23.48	0.240	877K	10.0M	80.9M	20
Ours	0.907	30.03	0.303	222K	21.6M	78.1M	70	0.795	27.00	0.263	218K	46.6M	152.6M	67	0.835	23.43	0.225	164K	10.4M	41.1M	121

Table 1: *We compare our method against 2DGS* trained with no geometric regularisations, BBSplat and GSTex, with default settings. We show standard quality metrics (PSNR, SSIM, L-PIPS), total number of primitives, texels and parameters. Our method achieves competitive results while using significantly fewer, highly expressive primitives. We also include 3DGS-MCMC for completeness; please note that 3DGS-based methods achieve better NVS but worse geometry quality compared to all approaches based on 2DGS (see discussion Sec. 5.1).*

	DeepBlending							Mip-Nerf-360							Tanks&Temples						
	SSIM↑	PSNR↑	LPIPS↓	Points	Texels	Params	FPS	SSIM↑	PSNR↑	LPIPS↓	Points	Texels	Params	FPS	SSIM↑	PSNR↑	LPIPS↓	Points	Texels	Params	FPS
BBSplat	0.895	28.93	0.332	72K	18.5M	78.1M	57	0.768	26.02	0.291	141K	36.1M	152.6M	33	0.801	22.77	0.259	47K	12.3M	51.8M	87
GSTex	0.896	28.29	0.354	222K	21.6M	78.1M	60	0.748	25.19	0.352	217K	46.6M	152.6M	46	0.745	20.65	0.363	164K	10.4M	41.1M	67
Ours	0.907	30.03	0.303	222K	21.6M	78.1M	70	0.795	27.00	0.263	218K	46.6M	152.6M	67	0.835	23.43	0.225	164K	10.4M	41.1M	121

Table 2: *We compare against BBSplat and GSTex in a same parameter setting, by adjusting their primitive count and texels accordingly. The slight discrepancy in BBSplat's parameters in Tanks & Temples is due to a sphere sampling method they use to model the background and distant objects.*

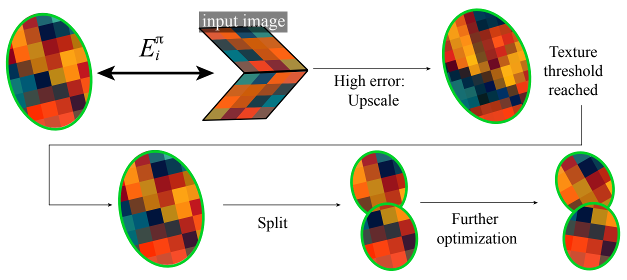

Figure 5: *The primitive has high error, and successive upscalings result in a texture resolution above threshold. In this case, the error is geometric rather than due to appearance. Our algorithm performs a split, and further optimization will match the geometry.*

3.4. Training and Regularisation

As in [KKLD23], we train our model using the weighted sum of the per-pixel $\mathcal{L}_1$ and the structural similarity losses, between the ground truth and the rendered images:

$$\mathcal{L}_{\text{RGB}} = (1 - \lambda_{\text{SSIM}})\mathcal{L}_1 + \lambda_{\text{SSIM}}\mathcal{L}_{\text{SSIM}} \tag{11}$$

with $\lambda_{\text{SSIM}} = 0.2$.

We observed that textures could finish converging to high-frequency settings, even though alpha-blending would create a smooth result. This prevented them from being downscaled, leading to high parameter usage. To resolve this, we apply a sparsity regularization on the texel values, pushing them towards zero,

$$\mathcal{L}_{\text{texture}} = \lambda_{\text{texture}} \sum_i |\mathbf{c}_i^{\mathcal{T}_i}| \tag{12}$$

We constrain the texel values in the $[-1, 1]$ range, using a sigmoid activation, scaled and shifted accordingly:

$$\mathbf{c}^{\mathcal{T}_i} = 2\sigma(\mathbf{c}'^{\mathcal{T}_i}) - 1 \tag{13}$$

where $\sigma(x)$ is the sigmoid function and $\mathbf{c}'^{\mathcal{T}_i}$ are the unactivated texture features. Intuitively, this parametrization coupled with the sparsity loss forces the view-dependent color $\mathbf{SH}(\mathbf{d})$ to learn the base color of the surface, while the texels operate as offsets to model high-frequency details. An example of this is illustrated in Fig. 1(left).

Finally, as in [PKK*24; KRS*24], we incentivize low-contribution primitives to effectively disappear by applying an opacity regularization term:

$$\mathcal{L}_{\text{opacity}} = \lambda_{\text{opacity}} \frac{1}{N} \sum_i^N \mathbf{o}_i. \tag{14}$$

Taking both training objectives and regularizations into consideration, the total loss is formed as:

$$\mathcal{L} = \mathcal{L}_{\text{RGB}} + \mathcal{L}_{\text{texture}} + \mathcal{L}_{\text{opacity}}. \tag{15}$$

4. Implementation

We have implemented our method using the 3DGS codebase, introducing 2D primitives as in 2DGS. Unlike 2DGS, we do not use the $\mathbb{R}^2 \to \mathbb{R}^2$ transformation to find the intersection point using three non-parallel planes. Instead, we use Eq. 1 directly in camera space. We did not observe any instability issues. Since our method requires per-ray querying of texture maps, both the forward and backward passes incur additional overhead, that is not compensated by the smaller number of primitives rendered. In general, compared to 2DGS, training time is 1.5-2 times longer, and rendering is 25% slower.

As the texture grids can have different resolutions, we created custom "jagged" tensor data structure to be used for their storage. The texture resolution for each primitive gets (de-)allocated dynamically, so that it covers the extent of the primitive, that is ± 3

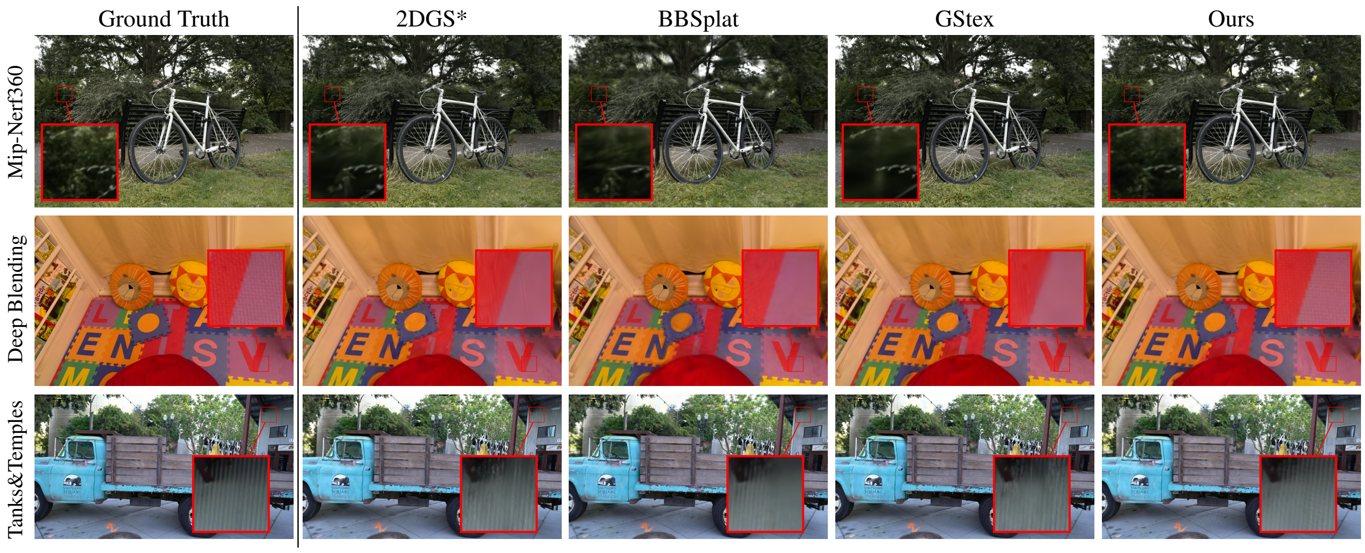

Figure 6: *We provide renderings from one scene per dataset for every method, trained with default settings. Our method is able to reconstruct high frequency details on images, even while using fewer parameters compared to the other methods.*

standard deviations. This dynamic memory management happens every 100 iterations. A hard limit of 256 texture resolution is enforced to avoid using too much memory. When a primitive grows outside of its allocated texture grid, either because memory has not been allocated yet or it has surpassed the hard limit, zero-padding is used. Since we are storing an offset, zero-padding reduces to the DC color.

We observed that limiting the $t_2 p_r$ to be greater or equal to 2 did not have a detrimental effect on the visual quality, as subtexel scaled details can be retrieved thanks to alpha blending and the overlap of our primitives. We thus allow upscaling to happen only for primitives with a $t_2 p_r$ greater or equal to 2.

Storage and Memory Considerations. The size of our representation is tied to the number of parameters used, with each parameter saved as a 4-byte float, as in previous Gaussian Splatting implementations. However, most Gaussian Splatting compression techniques (see [BKL*24]) are applicable to our approach either directly or with minor modifications. For our newly introduced texture maps, our choice to use a sigmoid activation to limit the effective range of the texel values allows for a very efficient application of K-means clustering compression. Please see Tab. 5 for complete statistics.

5. Results and Evaluation

We test our method on 13 indoor and outdoor scenes from three different standard datasets. We evaluate on all scenes of Mip-NeRF 360 [BMV*22], two scenes from Tanks & Temples [KPZK17], as well as two scenes from Deep Blending [HPP*18]. Fig. 6 shows that our method faithfully reconstructs the input images.

5.1. Evaluation

We compare our method against 2DGS [HYC*24] as a baseline, and two 2DGS texturing methods, namely (unpublished) BBSplat [SMAB25] and GStex [RCB*25], for which the code was available at the time of submission. Directly comparing to 3DGS-based models is not straightforward and might lead to misleading conclusions, since these models have different properties and consistently perform better for NVS but worse in geometry reconstruction, compared to 2DGS-based solutions [HYC*24]. For the same reason, and because the code was not available at the time of submission, we do not compare against [CTP*25], which is a 3DGS-based method with textured primitives. This model uses a pretrained 3DGS-MCMC model as initialization, similar to GStex, and includes RGBA textures, similar to BBSplat. We thus expect it to suffer from the disadvantages of both these two models (see discussion of the second experiment). We ran these methods on all scenes using our machines to ensure fair comparison. For 2DGS, the normal consistency and depth distortion regularization terms were disabled, as they are designed to enhance the geometric reconstruction but often result in lower novel view synthesis (NVS) quality.

Qualitative Evaluation. In Figure 6, we show visual results for the different models. We show test views (i.e., not used for training) from one scene of each dataset. Our method succeeds in reconstructing high-frequency details with high fidelity, even with fewer parameters. Similarly, Figure 7 displays visual results from the second experiment, which highlight that other texturing methods struggle when constrained to the same parameter budget as ours. GSTex uses a point cloud that is not trained with textured primitives in mind and a static heuristic distribution of texels, which causes it to not reconstruct well large parts of the scene. BBSplat succeeds in geometrically reconstructing the scene, but exhibits texture stretch-

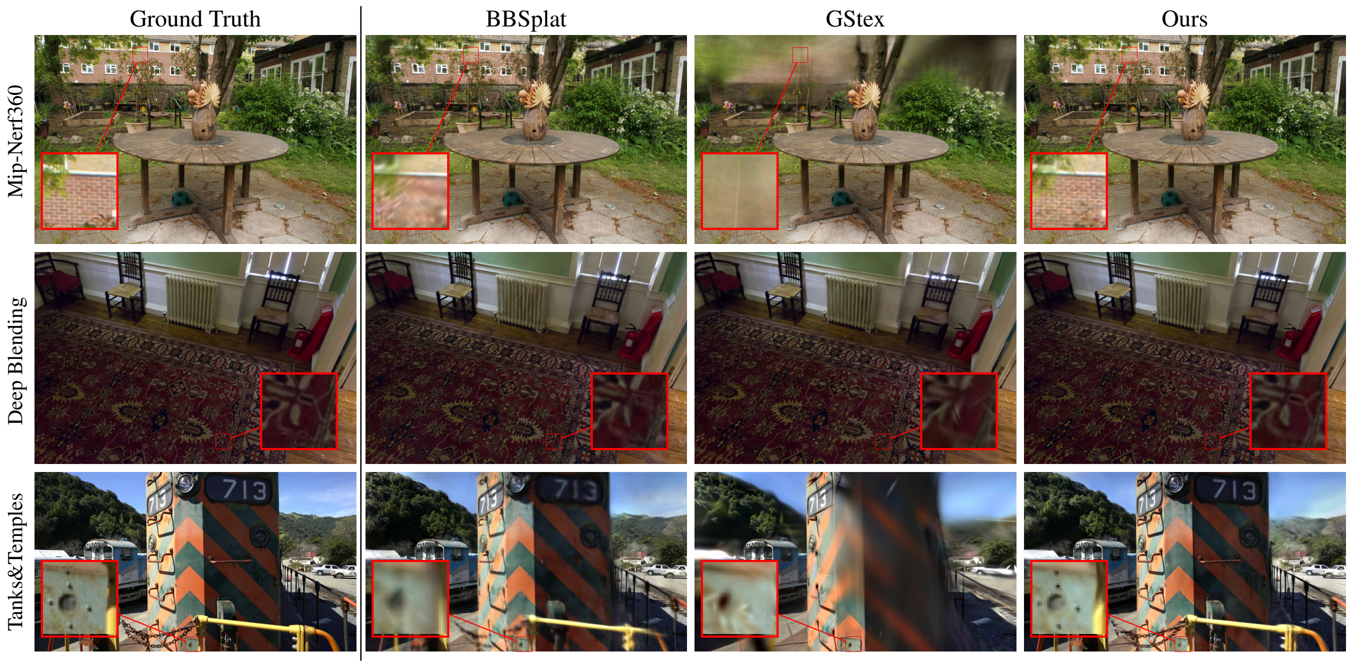

Figure 7: *Adjusting BBSplat and GSTex to use the same budget as our method, results in significant visual degradation.*

ing, which appears as blurriness. This is due to the choice of mapping between intersection points and UV coordinates.

Quantitative Evaluation. For our quantitative comparisons, we present results for three error metrics, SSIM, PSNR, and LPIPS [ZIE*18], that are typically used to evaluate NVS. Additionally, we report the number of primitives and texels used, which provides a precise measure of the resources required for each method and demonstrates whether and how much textures contribute to the scene reconstruction. Finally, since the models' capacity is directly tied to the number of parameters used, we include the relevant column so that we can make fair comparisons and draw meaningful conclusions.

The formula for the number of parameters differs slightly among the models. For 2DGS, GSTex each primitive has 58 parameters (3 for position, 2 for scale, 4 for rotation, 1 for opacity, 48 for color), while for BBSplat this number is 57, as it lacks opacity. In our method, in addition to the parameters of 2DGS, we include one additional per primitive for the texel size, pushing the number to 59 parameters. Regarding the parameters coming from texels, GSTex and our method have 3 per texel, while BBSplat has one additional, corresponding to the alpha channel.

In Table 1, we compare methods trained with default settings. In most cases, our approach achieves competitive or superior quality across all three metrics, while requiring a lower number of trainable parameters. Focusing on the composition of these parameters, our converged models have significantly fewer, but more expressive primitives than the baseline as illustrated in Fig. 1 (left). This highlights the ability of our model to account for the difference in complexity between geometry and appearance. In contrast, GSTex uses a pretrained 2DGS point cloud and has a fixed budget of texels, distributed at initialization time; this results in limited usage of texture. This is in part because it starts with a converged set of primitives from 2DGS that is not well adapted to a solution with texture. On the contrary, we optimize both primitives and textures from scratch.

Moreover, while our primitive count is close to BBSplat's, the total number of parameters used is significantly lower than theirs. This can be attributed to our content-aware texel size determination, which dynamically allocates model capacity to regions according to their needs, as opposed to the fixed texel to primitive ratio that BBSplat imposes.

To emphasize the importance of our content-aware texturing approach, we next compare the other texturing methods by fixing both the primitive and texel budgets to ours. In GSTex this is feasible, because it allows the distribution of a given texel budget over a pretrained 2DGS point cloud. We first trained 2DGS with the specified primitive budget and then gave the texel budget as input to GSTex. However, in BBSplat, controlling both the primitive and texel budgets is not possible, as the method imposes a fixed ratio between the two, determined by the texture resolution (16x16). Therefore, we calculated the number of primitives that would result in the same number of trainable parameters. Table 2, reports these results under these fixed parameters. Note that BBSplat uses a skybox to model background and distant objects in outdoor scenes, which was not taken into consideration when calculating the number of primitives. This accounts for the slight discrepancy in the number of parameters. Both the other methods observe a noticeable to significant drop in performance. This is possibly due to the fact that model capacity is not distributed efficiently both across primitives and across parts

	DeepBlending						Mip-Nerf-360						Tanks&Temples					
	SSIM↑	PSNR↑	LPIPS↓	Texels	Params	FPS	SSIM↑	PSNR↑	LPIPS↓	Texels	Params	FPS	SSIM↑	PSNR↑	LPIPS↓	Texels	Params	FPS
Points40k	0.890	28.78	0.346	13.2M	41.9M	121	0.758	25.80	0.304	42.2M	128.9M	97	0.809	22.74	0.246	15.4M	48.4M	162
Points80k	0.898	29.56	0.328	15.5M	51.2M	88	0.777	26.39	0.285	43.9M	136.5M	79	0.823	23.25	0.235	12.3M	41.7M	108
Points120k	0.903	29.88	0.315	17.9M	60.7M	75	0.785	26.68	0.275	45.0M	142.0M	68	0.830	23.34	0.232	10.9M	39.6M	96
Points160k	0.905	29.98	0.308	19.9M	69.1M	56	0.791	26.88	0.269	45.7M	146.6M	46	0.833	23.44	0.227	10.5M	40.8M	88

Table 3: *We run our technique with a limited primitive budget. An increased primitive count increases quality, without proportionally increasing the number of texels and parameters.*

of the scene. In this setup, our method performs consistently better on average than previous solutions.

As a third experiment, we run our method with a low, fixed primitive budget, by skipping the splitting procedure whenever the model exceeds it. The adaptive texel size strategies were left unchanged. Table 3 shows the results. As expected, the performance gets better with an increasing primitive count, as the use of primitives with a fixed gaussian falloff prevents faithfully reconstructing sharp edges with sparser point clouds. However, we note that the number of texels does not grow proportionally to the number of primitives or even diminishes, in the case of Tanks & Temples. This is a direct result of our choice of texture coordinate mapping function and the adaptive texel size determination strategy. Our separation of geometric and appearance parameters allows our models to automatically achieve a balance between the two that is appropriate for the scene.

6. Limitations and Discussion

Our method presents a good balance of resources used for textured Gaussian Splatting, allowing a smooth variation between number of primitives and number of texels used to represent a scene. However, like all other methods that build on 2DGS, we do not achieve the quality of 3DGS. Using 3D primitives with texture raises interesting questions about how to represent texture: should one use a 2D texture on a plane, or rather a voxel or hash-grid in 3D ? While Textured-GS [CTP*25] proposes a first solution using the former approach, the analysis presented in the paper is insufficient to determine how well the choices made perform compared to our solution.

We do not have any special treatment for anti-aliasing. For NVS, the user can only navigate freely within – or at least close – to the convex hull of the input cameras. Since we choose a $t_2 p_r$ value that is at least 2, aliasing will rarely by occuring in this range of viewpoints. Nonetheless, a complete solution for anti-aliasing would be an interesting avenue of future work.

We did not investigate the use of dedicated GPU hardware capabilities for texture. Given that our implementation of textured Gaussian Splatting uses a custom CUDA renderer, it is unclear how beneficial this would actually be. However, for the case, e.g., of WebGL renderers [Kwo], this may be a much more interesting direction that could allow accelerated rendering, especially in the case of low-end hardware.

7. Conclusion

We have presented a new representation for textured 2D Gaussian Splatting, that is driven by scene content. By adaptively choosing texel size to fit the content of the scene, and carefully balancing resources used with our resolution-aware spltting approach, we provide a versatile method that allows users to choose between more primitives or more texture while preserving image quality. Our method provides an additional point in the design space of primitive-based NVS algorithms, building on the long tradition of texture mapping in CG rendering.

8. Acknowledgements

This work was funded by the European Research Council (ERC) Advanced Grant NERPHYS, number 101141721 `https://project.inria.fr/nerphys`. The authors are grateful to the OPAL infrastructure of the Université Côte d'Azur for providing resources and support, as well as Adobe and NVIDIA for software and hardware donations. F. Durand acknowledges funding from Google, Amazon, and MIT-GIST. The authors thank the anonymous reviewers for their valuable feedback.

References

[BKL*24] Bagdasarian, Milena T., Knoll, Paul, Li, Yi-Hsin, et al. "3DGS.zip: A survey on 3D Gaussian Splatting Compression Methods". *arXiv preprint arXiv:2407.09510* (2024) 3, 7.

[BMT*21] Barron, Jonathan T., Mildenhall, Ben, Tancik, Matthew, et al. "Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields". *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2021, 5855–5864 2.

[BMV*22] Barron, Jonathan T., Mildenhall, Ben, Verbin, Dor, et al. "Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields". *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022, 5470–5479 2, 7.

[BMV*23] Barron, Jonathan T., Mildenhall, Ben, Verbin, Dor, et al. "Zip-NeRF: Anti-Aliased Grid-Based Neural Radiance Fields". *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2023, 19697–19705 2.

[Cat74] Catmull, Edwin Earl. *A subdivision algorithm for computer display of curved surfaces*. The University of Utah, 1974 2.

[CTP*25] Chao, Brian, Tseng, Hung-Yu, Porzi, Lorenzo, et al. "Textured Gaussians for Enhanced 3D Scene Appearance Modeling". *CVPR*. 2025 2, 3, 7, 9.

[CXG*22] Chen, Anpei, Xu, Zexiang, Geiger, Andreas, et al. "TensoRF: Tensorial Radiance Fields". *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXII*. Tel Aviv, Israel: Springer-Verlag, 2022, 333–350. ISBN: 978-3-031-19823-6. DOI: 10.1007/978-3-031-19824-3_20. URL: https://doi.org/10.1007/978-3-031-19824-3_20 2.

[FYT*22] Fridovich-Keil, Sara, Yu, Alex, Tancik, Matthew, et al. "Plenoxels: Radiance Fields without Neural Networks". *CVPR*. 2022 2.

[HLS07] Hormann, Kai, Lévy, Bruno, and Sheffer, Alla. "Mesh parameterization: Theory and practice". (2007) 2.

[HPP*18] Hedman, Peter, Philip, Julien, Price, True, et al. "Deep blending for free-viewpoint image-based rendering". *ACM Trans. Graph.* 37.6 (Dec. 2018). ISSN: 0730-0301. DOI: 10.1145/3272127.3275084. URL: https://doi.org/10.1145/3272127.3275084 7.

[HYC*24] Huang, Binbin, Yu, Zehao, Chen, Anpei, et al. "2D Gaussian Splatting for Geometrically Accurate Radiance Fields". *ACM SIGGRAPH 2024 Conference Papers*. SIGGRAPH '24. Denver, CO, USA: Association for Computing Machinery, 2024. ISBN: 9798400705250. DOI: 10.1145/3641519.3657428. URL: https://doi.org/10.1145/3641519.3657428 1–3, 7.

[KKLD23] Kerbl, Bernhard, Kopanas, Georgios, Leimkühler, Thomas, and Drettakis, George. "3D Gaussian Splatting for Real-Time Radiance Field Rendering". *ACM Transactions on Graphics* (2023). DOI: 10.1145/3592433 1, 2, 5, 6.

[KPZK17] Knapitsch, Arno, Park, Jaesik, Zhou, Qian-Yi, and Koltun, Vladlen. "Tanks and temples: benchmarking large-scale scene reconstruction". *ACM Trans. Graph.* 36.4 (July 2017). ISSN: 0730-0301. DOI: 10.1145/3072959.3073599. URL: https://doi.org/10.1145/3072959.3073599 7.

[KRS*24] Kheradmand, Shakiba, Rebain, Daniel, Sharma, Gopal, et al. "3D Gaussian Splatting as Markov Chain Monte Carlo". *Advances in Neural Information Processing Systems*. Ed. by Globerson, A., Mackey, L., Belgrave, D., et al. Vol. 37. Curran Associates, Inc., 2024, 80965–80986. URL: https://proceedings.neurips.cc/paper_files/paper/2024/file/93be245fce00a9bb2333c17ceae4b732-Paper-Conference.pdf 5, 6.

[Kwo] Kwok, Kevin. *Splat Viewer*. URL: https://github.com/antimatter15/splat?tab=readme-ov-file 9.

[LKK*18] Li, Minchen, Kaufman, Danny M, Kim, Vladimir G, et al. "Optcuts: Joint optimization of surface cuts and parameterization". *ACM transactions on graphics (TOG)* 37.6 (2018), 1–13 2.

[LME*23] Li, Zhaoshuo, Müller, Thomas, Evans, Alex, et al. "Neuralangelo: High-Fidelity Neural Surface Reconstruction". *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2023, 8456–8465. DOI: 10.1109/CVPR52729.2023.00817 2.

[MESK22] Müller, Thomas, Evans, Alex, Schied, Christoph, and Keller, Alexander. "Instant neural graphics primitives with a multiresolution hash encoding". *ACM Trans. Graph.* 41.4 (July 2022). ISSN: 0730-0301. DOI: 10.1145/3528223.3530127. URL: https://doi.org/10.1145/3528223.3530127 2.

[MSO*19] Mildenhall, Ben, Srinivasan, Pratul P, Ortiz-Cayon, Rodrigo, et al. "Local light field fusion: Practical view synthesis with prescriptive sampling guidelines". *ACM Transactions on Graphics (ToG)* 38.4 (2019), 1–14 2.

[MST*21] Mildenhall, Ben, Srinivasan, Pratul P., Tancik, Matthew, et al. "NeRF: representing scenes as neural radiance fields for view synthesis". *Commun. ACM* 65.1 (Dec. 2021), 99–106. ISSN: 0001-0782. DOI: 10.1145/3503250. URL: https://doi.org/10.1145/3503250 2.

[MST*25] Malarz, Dawid, Smolak-Dyżewska, Weronika, Tabor, Jacek, et al. "Gaussian splatting with NeRF-based color and opacity". *Computer Vision and Image Understanding* 251 (2025), 104273 2.

[MSY20] Mallett, Ian, Seiler, Larry, and Yuksel, Cem. "Patch textures: Hardware support for mesh colors". *IEEE Transactions on Visualization and Computer Graphics* 28.7 (2020), 2710–2721 2.

[NBM*22] Niemeyer, Michael, Barron, Jonathan T., Mildenhall, Ben, et al. "RegNeRF: Regularizing Neural Radiance Fields for View Synthesis From Sparse Inputs". *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022, 5480–5490 2.

[PKK*24] Papantonakis, Panagiotis, Kopanas, Georgios, Kerbl, Bernhard, et al. "Reducing the Memory Footprint of 3D Gaussian Splatting". *Proc. ACM Comput. Graph. Interact. Tech.* 7.1 (May 2024). DOI: 10.1145/3651282. URL: https://doi.org/10.1145/3651282 4, 6.

[RCB*25] Rong, Victor, Chen, Jingxiang, Bahmani, Sherwin, et al. "GStex: Per-Primitive Texturing of 2D Gaussian Splatting for Decoupled Appearance and Geometry Modeling". *Proceedings of the Winter Conference on Applications of Computer Vision (WACV)*. Feb. 2025, 3508–3518 2, 3, 7.

[RPK24] Rota Bulò, Samuel, Porzi, Lorenzo, and Kontschieder, Peter. "Revising Densification in Gaussian Splatting". *Computer Vision – ECCV 2024: 18th European Conference, Milan, Italy, September 29–October 4, 2024, Proceedings, Part LXIII*. Milan, Italy: Springer-Verlag, 2024, 347–362. ISBN: 978-3-031-73035-1. DOI: 10.1007/978-3-031-73036-8_20. URL: https://doi.org/10.1007/978-3-031-73036-8_20 5.

[RPLG21] Reiser, Christian, Peng, Songyou, Liao, Yiyi, and Geiger, Andreas. "KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs". *International Conference on Computer Vision (ICCV)*. 2021 2.

[SGV*24] Srinivasan, Pratul P, Garbin, Stephan J, Verbin, Dor, et al. "Nuvo: Neural uv mapping for unruly 3d representations". *European Conference on Computer Vision*. Springer. 2024, 18–34 2.

[SMAB25] Svitov, David, Morerio, Pietro, Agapito, Lourdes, and Bue, Alessio Del. *BillBoard Splatting (BBSplat): Learnable Textured Primitives for Novel View Synthesis*. 2025. arXiv: 2411.08508 [cs.CV]. URL: https://arxiv.org/abs/2411.08508 2, 3, 7.

[SRYT24] Sharma, Gopal, Rebain, Daniel, Yi, Kwang Moo, and Tagliasacchi, Andrea. "Volumetric rendering with baked quadrature fields". *European Conference on Computer Vision*. Springer. 2024, 275–292 2.

[SSC22] Sun, Cheng, Sun, Min, and Chen, Hwann-Tzong. "Direct Voxel Grid Optimization: Super-fast Convergence for Radiance Fields Reconstruction". *CVPR*. 2022 2.

[TTM*22] Tewari, Ayush, Thies, Justus, Mildenhall, Ben, et al. "Advances in neural rendering". *Computer Graphics Forum*. Vol. 41. 2. Wiley Online Library. 2022, 703–735 2.

[VHM*22] Verbin, Dor, Hedman, Peter, Mildenhall, Ben, et al. "Ref-NeRF: Structured View-Dependent Appearance for Neural Radiance Fields". *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022, 5491–5500 2.

[WSN*23] Wang, Zian, Shen, Tianchang, Nimier-David, Merlin, et al. "Adaptive Shells for Efficient Neural Radiance Field Rendering". *ACM Trans. Graph.* 42.6 (Dec. 2023). ISSN: 0730-0301. DOI: 10.1145/3618390. URL: https://doi.org/10.1145/3618390 2.

[XCW*24] Xu, Rui, Chen, Wenyue, Wang, Jiepeng, et al. *SuperGaussians: Enhancing Gaussian Splatting Using Primitives with Spatially Varying Colors*. 2024 2, 3.

[XHL*24] Xu, Tian-Xing, Hu, Wenbo, Lai, Yu-Kun, et al. "Texture-GS: Disentangling the Geometry and Texture for 3D Gaussian Splatting Editing". *Computer Vision – ECCV 2024: 18th European Conference, Milan, Italy, September 29–October 4, 2024, Proceedings, Part XXV*. Milan, Italy: Springer-Verlag, 2024, 37–53. ISBN: 978-3-031-72697-2. DOI: 10.1007/978-3-031-72698-9_3. URL: https://doi.org/10.1007/978-3-031-72698-9_3 3.

[YCH*24] Yu, Zehao, Chen, Anpei, Huang, Binbin, et al. "Mip-Splatting: Alias-free 3D Gaussian Splatting". *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2024, 19447–19456 2.

[YGS*24] Yang, Ziyi, Gao, Xinyu, Sun, Yang-Tian, et al. "Spec-Gaussian: Anisotropic View-Dependent Appearance for 3D Gaussian Splatting". *Advances in Neural Information Processing Systems*. Ed. by Globerson, A., Mackey, L., Belgrave, D., et al. Vol. 37. Curran Associates, Inc., 2024, 61192–61216. URL: https://proceedings.neurips.cc/paper_files/paper/2024/file/708e0d691a22212e1e373dc8779cbe53-Paper-Conference.pdf 2.

[YHR*23] Yariv, Lior, Hedman, Peter, Reiser, Christian, et al. "BakedSDF: Meshing Neural SDFs for Real-Time View Synthesis". *ACM SIGGRAPH 2023 Conference Proceedings*. SIGGRAPH '23. Los Angeles, CA, USA: Association for Computing Machinery, 2023. ISBN: 9798400701597. DOI: 10.1145/3588432.3591536. URL: https://doi.org/10.1145/3588432.3591536 2.

[YKH10] Yuksel, Cem, Keyser, John, and House, Donald H. "Mesh colors". *ACM Transactions on Graphics (TOG)* 29.2 (2010), 1–11 2.

[YLT19] Yuksel, Cem, Lefebvre, Sylvain, and Tarini, Marco. "Rethinking texture mapping". *Computer graphics forum*. Vol. 38. 2. Wiley Online Library. 2019, 535–551 2.

[YYTK21] Yu, Alex, Ye, Vickie, Tancik, Matthew, and Kanazawa, Angjoo. "pixelNeRF: Neural Radiance Fields From One or Few Images". *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021, 4578–4587 2.

[ZIE*18] Zhang, Richard, Isola, Phillip, Efros, Alexei A., et al. "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric". *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018 8.

[ZRSK20] Zhang, Kai, Riegler, Gernot, Snavely, Noah, and Koltun, Vladlen. "Nerf++: Analyzing and improving neural radiance fields". *arXiv preprint arXiv:2010.07492* (2020) 2.

[ZTF*18] Zhou, Tinghui, Tucker, Richard, Flynn, John, et al. "Stereo magnification: learning view synthesis using multiplane images". *ACM Trans. Graph.* 37.4 (July 2018). ISSN: 0730-0301. DOI: 10.1145/3197517.3201323. URL: https://doi.org/10.1145/3197517.3201323 2.

Appendix A: Implementation Details

In this section we provide some more details on the implementation.

The texture grids are initialized with a value of 0 on every channel and start optimizing after 500 iterations. To avoid running out of memory and to avoid early overfitting, at 500 iterations, we set $t_2 p_r$ for each primivite so that their smallest axis is 8 texels wide, in practice using the closest power of two value. The error calculation along with the adaptive texel size determination and resolution-aware primitive management strategies run every 250 iterations, until 25k iterations. The threshold τ_{tr} starts at 64 and is progressively reduced to 32 within 7000 iterations. We implement downscaling and upscaling using pytorch's interpolate function, with a scale factor of 2, which translates to reductions or increases of the number of texels by a factor of 4, respectively. Upscale is done using nearest neighbour interpolation, so that we avoid changing the appearance of the texture, that could disrupt that optimizer.

Appendix B: Hyperparameter Tuning

We provide here some intuition on their impact of hyperparameters to the final model.

The quantile of the error E used in both texel size adaptation and primitive management routines affects how aggressively they act on the model. A lower quantile means more aggressive changes as more primitives are potentially upscaled and/or split, while a higher one has the opposite effect. τ_{tr} controls how many primitives are split or not, contributing to the total number of primitives. Setting this hyperparameter to a low number leads to a model with more primitives, as it is easier for primitives to fall above the texture resolution threshold. Finally, λ_{texture} controls the variance of the texture maps, with higher values leading to smoother results, while downscale parameter $t_{\text{ds}}a$ affects how easily a texture map can get downscaled, and thus having a simpler appearance. Both of these hyperparameters have a direct effect on the number of texels. A low λ_{texture} and high t_{ds} result in models with few texels per primitive, while the opposite configuration leads to more texels being used overall, increasing the number of parameters used.

Algorithm 1: Texel Size Adaptation and Primitive Management Routine

```
if $iter$ mod $250 = 0$ then
    forall primitives do
        Compute $E_i$;
        if texture resolution > $\tau_{tr}$ and $E_i$ in top 90% then
            Split primitive along overflowing axes;
        end
        if $t_2 p_r > 1$ and $E_i$ in top 90% then
            Decrease texel size (Upscale);
        end
        else if $\mathcal{E}_d < \tau_{ds}$ then
            Increase texel size (Downscale);
        end
    end
end
```

Note that since the number of primitives and the number of texels are linked through our resolution-aware primitive management, a change in one hyperparameter can have secondary, indirect effects. For example, encouraging the spawning of more primitives by using a lower τ_{tr} can lead to fewer texels, since each primitive represents a more localized and therefore simpler part of the scene. This is demonstrated in the third experiment (Tab 3) with Tanks&Temples. Our method is robust enough to operate with different ratios of primitive and texel budgets, converging to good results and automatically adjusting the allocation of model capacity. In Tab. 4 we provide two additional models, one with more primitives, as a result of a lower τ_{tr} and one with less texels, as a result of a higher τ_{ds}.

	DeepBlending							Mip-Nerf-360							Tanks&Temples						
	SSIM↑	PSNR↑	LPIPS↓	Points	Texels	Params	FPS	SSIM↑	PSNR↑	LPIPS↓	Points	Texels	Params	FPS	SSIM↑	PSNR↑	LPIPS↓	Points	Texels	Params	FPS
Default	0.907	30.03	0.303	222K	21.6M	78.1M	70	0.795	27.00	0.263	218K	46.6M	152.6M	67	0.835	23.43	0.225	164K	10.4M	41.1M	121
Higher τ_{ds}	0.903	29.90	0.324	196K	7.9M	35.4M	67	0.791	26.97	0.277	233K	33.8M	115.1M	60	0.829	23.40	0.246	165K	6.6M	29.4M	118
Lower τ_{tr}	0.907	30.08	0.300	316K	22.0M	84.8M	66	0.812	27.30	0.244	542K	46.3M	170.8M	52	0.842	23.67	0.217	281K	10.9M	49.3M	104

Table 4: *We provide results for two extra models, trained with different τ_{ds} and τ_{tr} to demonstrate the effect of these hyperparameters in the final model.*

Scene	SSIM↑	PSNR↑	LPIPS↓	Points	Texels	Params	FPS	Mem	Texels/ Primitive
drjohnson	0.906	29.78	0.314	293K	20.8M	79.7M	77	318MB	70.9
playroom	0.907	30.28	0.293	152K	22.5M	76.4M	64	305MB	147.7
bicycle	0.726	24.22	0.254	186K	83.9M	262.7M	84	1050MB	450.5
bonsai	0.933	31.46	0.253	291K	9.0M	44.1M	35	176MB	30.8
counter	0.900	28.79	0.261	237K	12.4M	51.3M	41	205MB	52.2
flowers	0.531	20.22	0.392	177K	61.3M	194.5M	74	777MB	345.6
garden	0.832	26.56	0.156	164K	51.7M	164.7M	112	658MB	314.6
kitchen	0.915	30.90	0.173	288K	9.6M	45.9M	40	183MB	33.3
room	0.921	31.72	0.272	212K	17.7M	65.7M	62	262MB	83.3
stump	0.756	26.09	0.273	228K	101.4M	317.6M	77	1270MB	444.5
treehill	0.641	23.01	0.335	180K	72.2M	227.2M	77	908MB	398.8
train	0.795	21.38	0.279	172K	7.2M	31.8M	139	127MB	41.7
truck	0.875	25.47	0.172	156K	13.7M	50.4M	103	201MB	87.3

Table 5: *Per-scene metrics of our model, grouped by their dataset.*