Image-GS: Content-Adaptive Image Representation via 2D Gaussians

YUNXIANG ZHANG, New York University, USA
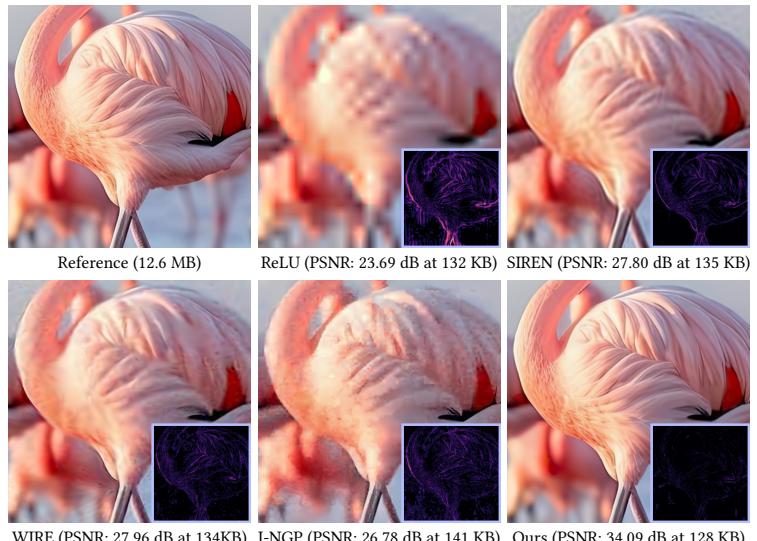ALEXANDR KUZNETSOV and AKSHAY JINDAL, Intel Corporation, USA
KENNETH CHEN, New York University, USA
ANTON SOCHENOV and ANTON KAPLANYAN, Intel Corporation, USA
QI SUN, New York University, USA

(a) Image-GS: Gaussian-based adaptive image representation

(b) visual comparison with previous neural image representations

Fig. 1. *Content-adaptive image representation.* Leveraging a tailored differentiable renderer, Image-GS adaptively distributes and progressively optimizes a set of anisotropic 2D Gaussians to fit a target image. Image-GS is a flexible image representation that has high memory & computation efficiency, supports fast random pixel access, and offers a natural level of detail. (a) shows the learned Gaussian position distribution (green dots); 20% of Gaussians are plotted for better visibility. (b) Image-GS's content-adaptive nature enables it to wisely allocate resources based on the local signal complexity and preserve fine image details with higher fidelity than alternative methods. The insets visualize the corresponding error images, with brighter colors indicating higher errors.

Neural image representations have recently emerged as a promising technique for storing, streaming, and rendering visual data. Coupled with learning-based workflows, these novel representations have demonstrated remarkable visual fidelity and memory efficiency. However, existing neural image representations often rely on explicit uniform data structures without content adaptivity or computation-intensive implicit models, limiting their adoption in real-time graphics applications.

Inspired by recent advances in radiance field rendering, we propose Image-GS, a content-adaptive image representation. Using anisotropic 2D Gaussians as the basis, Image-GS shows high memory efficiency, supports fast random access, and offers a natural level of detail stack. Leveraging a tailored differentiable renderer, Image-GS fits a target image by adaptively allocating and progressively optimizing a set of 2D Gaussians. The generalizable efficiency and fidelity of Image-GS are validated against several recent neural image representations and industry-standard texture compressors on a diverse set of images. Notably, its memory and computation requirements solely depend on and linearly scale with the number of 2D Gaussians, providing flexible controls over the trade-off between visual fidelity and run-time efficiency. We hope this research offers insights for developing new applications that require adaptive quality and resource control, such as machine perception, asset streaming, and content generation.

CCS Concepts: • **Computing methodologies** → **Computer graphics**.

Additional Key Words and Phrases: Image Representation

Authors' addresses: Yunxiang Zhang, yunxiang.zhang@nyu.edu, New York University, USA; Alexandr Kuznetsov, alexandr.kuznetsov@intel.com; Akshay Jindal, akshay.jindal@intel.com, Intel Corporation, USA; Kenneth Chen, kennychen@nyu.edu, New York University, USA; Anton Sochenov, anton.sochenov@intel.com; Anton Kaplanyan, anton.kaplanyan@intel.com, Intel Corporation, USA; Qi Sun, qisun@nyu.edu, New York University, USA.

1 INTRODUCTION

For a long time, images have been digitized as uniform pixel grids in both hardware and software. Such discrete representations do not align with the physical world, where visual content is continuous and non-uniform. As a result, these representations suffer from limited efficiency, especially for domain-specific tasks such as machine vision and content streaming [Chen et al. 2021; Li et al. 2020].

Neural representations have recently emerged to encode, process, and render images using neural networks [Chen et al. 2021; Dosovitskiy et al. 2020; Karnewar et al. 2022; Martel et al. 2021]. Incorporated into learning-based frameworks, these representations have demonstrated significant advantages over traditional image

formats in terms of visual fidelity, memory efficiency, and machine vision task performance. However, neural image representations commonly rely on explicit, uniform data structures that do not adapt to the image content or computationally heavy implicit networks for image decoding. Such characteristics become a major barrier in real-time graphics applications that require fast memory access or resource-dependent quality adaptation.

To this end, we propose Image-GS, a flexible, compact, and content-adaptive image representation based on anisotropic, colored 2D Gaussians. Specifically, each 2D Gaussian is characterized by its mean, covariance, and color. Given a target image, a group of 2D Gaussians is adaptively spawned based on the magnitude of local image gradients, with more Gaussians allocated to regions with high-frequency details. The Gaussian parameters are then optimized via a tailored differentiable renderer to reconstruct the target image, and additional Gaussians are progressively added to image regions exhibiting high reconstruction errors. To accelerate Image-GS's inference speed, we introduce a hierarchical grid structure based on binary space partitioning. The content-adaptive nature of Image-GS enables it to wisely allocate resources based on the local signal complexity and preserve fine image details with high fidelity.

Through a series of comparative experiments with several recent neural image representations and industry-standard texture compression algorithms, we validate Image-GS's generalizable performance in terms of visual quality and memory & computation efficiency. Additionally, Image-GS supports hardware-friendly fast random access, continuous level-of-detail adaptation, and real-time inference performance. We hope this research provides insights for developing novel image representations that have the advantages of both explicit (direct memory access & adaptive level-of-details) and implicit (efficient storage & learning-compatible) encoding, and therefore, supporting specific hardware and application needs.

In summary, this research contributes:

- a flexible, compact, and content-adaptive image representation based on anisotropic colored 2D Gaussians;
- a tailored differentiable renderer that efficiently aggregates anisotropic colored 2D Gaussians into images;
- a hierarchical spatial partitioning for accelerated inference.

2 RELATED WORK

2.1 Image and Texture Compression

Traditional image compression methods have prioritized efficient storage and transmission over real-time graphics. Lossless methods optimize pixel permutation and use entropy encoding [Welch 1985], while lossy methods transform image blocks into frequency domains using wavelet [Antonini et al. 1992] or cosine transforms [Wallace 1992] followed by quantization and entropy encoding. Advanced lossy methods also consider human color sensitivity, higher bit depths, wide color gamuts, user statistics [Alakuijala et al. 2019], and employ content-adaptive block sizes and looped filtering to reduce artifacts [Chen et al. 2018]. Despite high compression ratios, these methods are complex to decode, slow, and unsuitable for non-color data like normal maps in real-time graphics. In contrast, texture compression methods aim to reduce GPU bandwidth and support non-color data, random access, and fast decompression.

They operate on small 4x4 pixel blocks, maintaining local statistics (such as mean and variance) while reducing bits-per-pixel (*bpp*) within each block [Delp and Mitchell 1979]. Each block is compressed independently, storing per-pixel color values [Campbell et al. 1986], base color with adjustments [Ström and Akenine-Möller 2005; Ström and Pettersson 2007], or color endpoints with interpolation indices [BC 2024]. Advanced methods allow dynamic block sizes, HDR content, and content-adaptive compression strategy per block [Nystad et al. 2012]. These methods offer fast random access but are limited to an 8:1 compression ratio. Recently, Vaidyanathan et al. [2023] proposed compressing multiple material textures and mipmap chains with a small multilayer perceptron (MLP), achieving high compression and real-time random access. Our Image-GS representation also achieves a high compression ratio, good visual fidelity, and real-time inference, with additional content-adaptive optimization, at-will level-of-detail queries, and variable bit rate.

2.2 Neural Image Representation

Neural image representation is an emerging field that diverges from traditional pixel-based methods, using deep features or implicit neural functions to encode images. Ballé et al. [2018] use variational autoencoders (VAEs) with deep hyperprior to create compact latent space image representation. Chen et al. [2021] transform images into continuous signals via a 2D feature map and a shared decoder. These methods, trained on fixed image sets, may struggle to generalize to new images. In contrast, per-image encoder/decoder approaches employ MLPs to approximate 2D image signals, enhanced with activation functions like sinusoids [Sitzmann et al. 2020] and Gabor wavelets [Saragadam et al. 2023], or positional encoding [Tancik et al. 2020] to capture fine details. These methods can also be extended to discontinuous signals through hybrid neural-mesh representations [Belhe et al. 2023]. Karnewar et al. [2022] showed that even traditional grid-based representations can be improved by adding a fixed non-linearity to interpolated values. Martel et al. [2021] proposed a hybrid method that uses multiscale block-coordinate decomposition to adaptively allocate resources based on local signal complexity. These methods, though capable of representing complex images with high quality and low memory, often have long training and inference times and struggle with high-frequency variations due to single-scale representation. Müller et al. [2022] addressed these issues with fully fused MLPs and multi-resolution hash grids, enabling quick, memory-efficient learning and inference of gigapixel images. In Section 4, we demonstrate that our Image-GS representation achieves superior visual quality at low bit rate compared to these neural methods while supporting real-time inference.

Gaussian mixtures in graphics. In parallel to neural representation, Gaussian mixture representations are also seeing an emerging trend in computer graphics. Our work is inspired by the recent 3D Gaussian Splatting method [Kerbl et al. 2023] that uses explicit 3D Gaussian functions for real-time high-quality novel-view synthesis. Many follow-up works have extended this method to dynamic scenes [Luiten et al. 2023], on-the-fly training and streaming [Sun et al. 2024], 2D Gaussians for surface modeling [Huang et al. 2024], and a wide variety of other applications [Chen and Wang 2024].

Fig. 2. *Optimization pipeline of our Gaussian-based adaptive image representation.* At initialization, a group of 2D Gaussian primitives are adaptively spawned based on the magnitude of local image gradients, with more Gaussians allocated to regions with fine details (Section 3.3). During training, the parameters associated with the Gaussian primitives (Section 3.1) are optimized via a tailored differentiable renderer (Section 3.2) to reconstruct the target image, and additional Gaussians are progressively added to image regions exhibiting high reconstruction errors (Section 3.3). Note that we plot Gaussians as colored elliptical discs with red frames based on their mean and covariance, and overlay them with rendered images for better visibility of the training progress.

Although Gaussian mixture models have been used for image generation [Gepperth and Pfülb 2021], compression [Sun et al. 2019, 2021], and stylization [Cheng 2024], their application as 2D image representation remains largely unexplored in real-time graphics.

3 METHOD

We first introduce a flexible and compact image representation based on anisotropic, colored 2D Gaussians (Section 3.1), then present a tailored differentiable renderer to aggregate Gaussians into pixel values (Section 3.2). The Gaussian parameters are optimized to reconstruct the image content, while additional 2D Gaussians are progressively added to error-prone regions (Section 3.3). The resulting representation, named Image-GS, adaptively allocates 2D Gaussians based on the complexity of local image regions, achieving a favorable trade-off between visual fidelity and memory/computation efficiency. To enable low-cost random access and fast rendering speed, we build an image-specific hierarchical grid to spatially partition optimized Gaussians and reduce run-time computations (Section 3.4).

3.1 Images as Anisotropic 2D Gaussians

Due to the adoption of memory-consuming feature grids [Lombardi et al. 2019; Sitzmann et al. 2019a] and computation-intensive implicit models [Martel et al. 2021; Niemeyer et al. 2020; Sitzmann et al. 2019b], existing neural representations exhibit limited scalability for complex visual data in real-time graphics applications. To address this issue, we draw inspiration from the recent success of 3D Gaussian splatting [Kerbl et al. 2023], an explicit scene representation supporting high-quality and real-time rendering, and propose to use anisotropic 2D Gaussians as the representation basis for images.

Similar to the Gaussian primitives in 3D, the geometry and orientation of an anisotropic 2D Gaussian is characterized by a covariance matrix $\Sigma \in \mathbb{R}^{2\times2}$ centered at image coordinates $\boldsymbol{\mu} = (u, v)$. Its density value evaluated at an arbitrary pixel location $\mathbf{x} \in \mathbb{R}^2$ gives:

$$G(\mathbf{x}) = \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})\right), \tag{1}$$

To ensure that the covariance matrix Σ remains physically feasible, i.e., positive semi-definite, during the numerical optimization

process detailed in Section 3.3, we factorize it into a rotation matrix $\mathbf{R} \in \mathbb{R}^{2\times2}$ and a diagonal scaling matrix $\mathbf{S} \in \mathbb{R}^{2\times2}$:

$$\Sigma = \mathbf{R}\,\mathbf{S}\,\mathbf{S}^T\,\mathbf{R}^T, \tag{2}$$

Specifically, we create and maintain a rotation angle $\theta \in [0, \pi]$ and a scaling vector $\mathbf{s} \in \mathbb{R}_+^2$ for each 2D Gaussian. These parameters are optimized via stochastic gradient descent and clipped to their allowed value range during the training process. The rotation matrix $\mathbf{R}$ and the scaling matrix $\mathbf{S}$ are constructed on the fly using θ and $\mathbf{s}$ at both the training and inference stages.

Unlike Gaussian splatting in 3D that adopts spherical harmonics to model view-dependent color effects [Kerbl et al. 2023], we only utilize a 3-dimensional vector $\mathbf{c} \in \mathbb{R}_+^3$ for each 2D Gaussian to store its RGB values, as an image essentially captures a single view of a scene. In addition, 3D Gaussian splatting relies on a per-Gaussian trainable opacity parameter for depth-based occlusion computation and α-blending during the rendering process. By contrast, depth information is not necessary for accurate rendering in the 2D space, and 2D Gaussian primitives can be effectively aggregated regardless of their relative order, as we explain in Section 3.2. Therefore, our 2D Gaussian primitives do not have an opacity property.

Based on the discussion above, an arbitrary 2D Gaussian primitive in our image representation, G_i with $1 \leq i \leq N_g$, is fully characterized by a vector containing 8 trainable parameters $\mathbf{p}_i \in \mathbb{R}^8$:

$$\mathbf{p}_i := \mathbf{p}_i(\boldsymbol{\mu}_i, \theta_i, \mathbf{s}_i, \mathbf{c}_i), \tag{3}$$

3.2 Aggregating 2D Gaussians into Images

While Gaussian splatting in 3D necessitates per-view depth sorting and per-Gaussian opacity to handle occlusions and enforce multi-view consistency, especially when there exist objects that are visible in some views but not in others, we argue that depth sorting and occlusion modeling can be safely omitted in the 2D case without negatively impacting the rendering quality. Since an image only captures a single view of an underlying 3D scene, there is no need to account for any potential multi-view consistency issues in other viewing directions due to inconsistent depth information. As a result, an image represented by a set of 2D Gaussians can be rendered

by applying the Gaussians in arbitrary order, as long as the final rendering result is pixel-wise accurate for that particular image.

With this insight in mind, we simplify the standard point-based α-blending approach from the literature [Kopanas et al. 2022, 2021; Yifan et al. 2019] by treating our Gaussian primitives as an unordered set of semi-transparent anisotropic points, and aggregate their color contributions to form an image pixel $\mathbf{x} \in \mathbb{R}^2$:

$$\mathbf{c}_r(\mathbf{x}) = \sum_{i=1}^{N_g} G_i(\mathbf{x}) \cdot \mathbf{c}_i, \tag{4}$$

While this naive formulation effectively renders 2D Gaussians in an order-agnostic manner, it involves all Gaussians to compute the color of each pixel. Such global pixel-Gaussian correlation breaks the data locality that is necessary for efficient random pixel access on GPUs and largely limits the overall rendering speed [Vaidyanathan et al. 2023]. Moreover, the fact that each Gaussian receives gradients through pixel errors across the whole image domain makes the optimization less consistent across iterations and slower to converge, as training updates in irrelevant, faraway image regions can influence a 2D Gaussian in unpredictable and undesirable ways.

For this reason, we limit the number of Gaussians that contribute to a given pixel based on their density. Specifically, we first evaluate and rank the density values of all Gaussians at $\mathbf{x} \in \mathbb{R}^2$, then only keep the top-K and use their density values as weights for rendering. Besides, we also normalize these weights before the aggregation to avoid situations where certain pixels are not adjacent enough to their top-K Gaussians to receive sufficient color contributions. Overall, our rendering algorithm for 2D Gaussians is formulated as:

$$\mathbf{c}_r(\mathbf{x}) = \frac{1}{\sum_{j \in \mathcal{N}_K(\mathbf{x})} G_j(\mathbf{x})} \sum_{j \in \mathcal{N}_K(\mathbf{x})} G_j(\mathbf{x}) \cdot \mathbf{c}_j, \tag{5}$$

where $\mathcal{N}_K(\mathbf{x})$ represents the set of top-K Gaussians for $\mathbf{x}$, defined as $\mathcal{N}_K(\mathbf{x}) = \text{top-K}(\{G_i(\mathbf{x})\}_{i=1}^{N_g})$. Throughout our experiments, K is set to 10 if not explicitly specified otherwise.

3.3 Content-Adaptive Initialization and Optimization

Unlike scene geometries which are commonly sparse in the 3D space, images (except the ones with an alpha channel) typically contain dense color information everywhere in the 2D image domain. A good initialization of our 2D Gaussian primitives, therefore, should output a spatial distribution that covers the entire image domain while emphasizing regions with high-frequency, fine details.

To this end, we propose a content-adaptive sampling strategy that combines local image gradient guidance with uniform sampling. Specifically, we only sample pixel locations to initialize the position of Gaussians. During the position sampling for each Gaussian, the probability of a given pixel $\mathbf{x}$ being sampled is a weighted sum of the relative magnitude of its local image gradient and a constant shared across all pixel locations, as formulated below. In particular, the left term advocates image-content adaptivity, while the right term ensures appropriate image-domain coverage.

$$\mathbb{P}_{init}(\mathbf{x}) = \frac{(1 - \lambda_{init}) \cdot \|\nabla I(\mathbf{x})\|_2}{\sum_{h=1}^{H} \sum_{w=1}^{W} \|\nabla I(\mathbf{x}_{h,w})\|_2} + \frac{\lambda_{init}}{H \cdot W}, \tag{6}$$
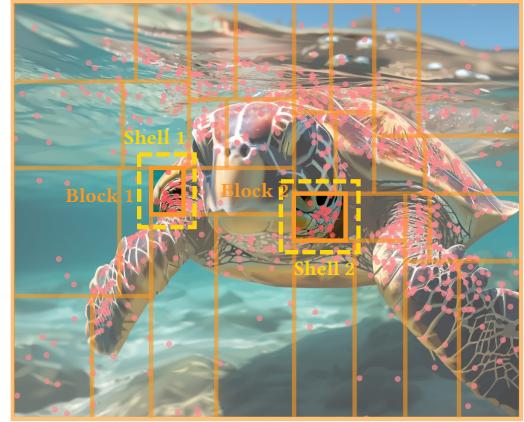

Fig. 3. *Accelerated inference via hierarchical spatial partitioning.* After training, the target image is adaptively subdivided into N_b disjoint blocks (solid orange) such that each block covers a subset of optimized Gaussians (red dots show their locations) with similar cardinality ($\approx N_g/N_b$). Instead of querying all N_g Gaussians, rendering a pixel in block B_k only involves the Gaussians within its corresponding outer shell S_k (dotted yellow). The size difference between each block-shell pair is designed to mitigate boundary artifacts. Only 10% of Gaussians are plotted for better visibility.

where H/W give the height/width of the image, and $\nabla I(\cdot)$ denotes the image gradient operator. $\lambda_{init} \in [0, 1]$ balances local content adaptivity and uniform coverage, and is set to 0.3 in our experiments. In addition to position initialization, all Gaussians are assigned the target pixel color at their initialized location $\mathbf{c}_t(\mathbf{x})$.

To optimize the Gaussian parameters toward reconstructing the target image, during each training iteration, we sample a large set of pixel locations and compute the L_1 loss between rendered (using the differentiable renderer in Section 3.2) and ground-truth pixel values at sampled locations. Similar to Equation (6), the sampling distribution again has an image-gradient term to emphasize complex local image content and a constant term to enforce broad coverage.

$$\mathbb{P}_{opt}(\mathbf{x}) = \frac{(1 - \lambda_{opt}) \cdot \|\nabla I(\mathbf{x})\|_2}{\sum_{h=1}^{H} \sum_{w=1}^{W} \|\nabla I(\mathbf{x}_{h,w})\|_2} + \frac{\lambda_{opt}}{H \cdot W}, \tag{7}$$

The trade-off parameter $\lambda_{opt} \in [0, 1]$ is set to 0.8 in our experiments.

Besides the initially created Gaussians, we also periodically add new Gaussian primitives to image regions having high reconstruction errors as the training progresses. This is achieved by sampling pixel locations based on their relative error magnitude.

$$\mathbb{P}_{add}(\mathbf{x}) = \frac{|\mathbf{c}_r(\mathbf{x}) - \mathbf{c}_t(\mathbf{x})|}{\sum_{h=1}^{H} \sum_{w=1}^{W} |\mathbf{c}_r(\mathbf{x}_{h,w}) - \mathbf{c}_t(\mathbf{x}_{h,w})|}, \tag{8}$$

Figure 2 illustrates the optimization pipeline of Image-GS.

3.4 Hierarchical Spatial Partitioning for Efficient Inference

For real-time graphics applications with demanding performance requirements, the fact that rendering a single pixel requires evaluating and ranking the density values of all Gaussian primitives is computationally infeasible. Fortunately, the constraint on the number of Gaussians making contributions to each pixel (Equation (5))

enforces pixel-Gaussian locality during the optimization and ensures that the top-K Gaussians of each pixel are located within a small local neighborhood after the training.

Following this intuition, we tailor a hierarchical grid structure to each Image-GS-represented image to spatially partition the set of all N_g optimized 2D Gaussians into N_b smaller subsets of similar cardinality ($\approx N_g/N_b$). This is achieved via a variant of binary space partitioning (BSP), where only horizontal and vertical splitting lines are employed. Given the maximum number of Gaussians allowed in each subset N_{max}, a BSP tree is constructed to iteratively and adaptively subdivide the image domain into smaller blocks until the termination criterion is met. Specifically, each node in the BSP tree corresponds to a block in the image space. The ensemble of leaf nodes forms a set of disjoint blocks that together cover the entire image domain. Each leaf block undergoes a sequence of alternating horizontal and vertical splitting. Each splitting line is computed such that it separates a leaf block into two smaller blocks with an equal number of Gaussians. This process continues until no leaf block contains more than N_{max} Gaussians. The resulting leaf blocks give the desired subsets of Gaussians.

However, rendering pixels in a leaf block B_k using only the Gaussians in B_k can result in severe boundary artifacts, where neighboring pixels in two adjacent leaf blocks exhibit unnatural color changes. To address this issue, we extend the boundaries of each block B_k by $1/4$ to obtain its corresponding encompassing shell S_k and make all Gaussians within S_k available for rendering pixels in B_k. Figure 3 illustrates the resulting hierarchical spatial partitioning. Instead of querying all N_g Gaussians, rendering a pixel only involves $\approx 9N_g/4N_b$ Gaussians now. Figure 4 shows the inference acceleration performance for a varying number of blocks.

4 EVALUATION

4.1 Experimental Setup

Evaluation dataset. To comprehensively understand and validate Image-GS's image representation performance, we prepared an evaluation dataset of 30 RGB images (2K×2K resolution) with diverse characteristics, including 10 photographs, 4 vector-style images, 8 texture maps, 4 anime posters, and 4 watercolor/oil paintings.

Evaluation metrics. We adopt 4 image quality metrics, PSNR, SSIM [Wang et al. 2004], LPIPS [Zhang et al. 2018], and FLIP [Andersson et al. 2020], to evaluate the visual fidelity of Image-GS and establish quantitative comparisons with baseline methods. We also report the parameter size of each representation and the corresponding bit rate in *bpp* (bits per pixel) to evaluate their memory efficiency.

Implementation. The only trainable parameters in Image-GS are the Gaussian parameters in Equation (3). By mapping image domains to $[0, 1]^2$ grids, Image-GS works for target images of any aspect ratio and resolution. At initialization, the Gaussian positions μ and colors c are populated by sampling pixel coordinates based on Equation (6). Their scaling vectors s and rotation angles θ are set to $2/\max(H, W)$ and 0, respectively. We adopt the Adam optimizer [Kingma and Ba 2015] to iteratively update these parameters for 50K iterations. The learning rates for (μ, c, s, θ) start at $(2e\text{-}4, 2e\text{-}3, 1e\text{-}3, 1e\text{-}3)$ and decay by 10 (only once) if no improvement (PSNR and SSIM are computed
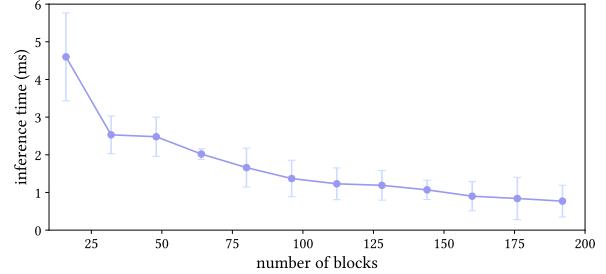

Fig. 4. *Performance analysis with hierarchical grid partitioning.* We measure the time of rendering 10K pixels with Image-GS (8K Gaussians) for a varying number of BSP blocks (Section 3.4). The measurement is performed on an NVIDIA GeForce RTX 3080 GPU. Each data point is averaged over 100 trials. Note that our proof-of-concept implementation uses pure PyTorch code only. Further acceleration can be achieved with customized CUDA kernels.

every 1K iterations) has been made for 3 consecutive measurements. During each iteration, 10K pixel locations $\mathcal{P}$ are sampled based on Equation (7) to evaluate the loss function:

$$L = \frac{1}{|\mathcal{P}|} \sum_{x \in \mathcal{P}} |c_r(x) - c_t(x)|, \tag{9}$$

During training, we progressively allocate additional Gaussians to image regions with high fitting errors based on Equation (8). For a budget of N_g Gaussians, we initialize training with $N_g/2$ Gaussians and optimize for 10K iterations. $N_g/8$ additional Gaussians are added every 5K iterations until the budget runs out. We implement Image-GS in PyTorch [Paszke et al. 2019] and use half-precision floating-point numbers (float16) for all parameters.

4.2 Acceleration via Hierarchical Spatial Partitioning

Using the acceleration technique introduced in Section 3.4, we analyze Image-GS's system performance under a varying number of blocks. With the BSP tree constructed, the image domain $[0, 1]^2$ is hierarchically partitioned into N_b disjoint blocks. Each block B_k is defined by the coordinates of its top-left and bottom-right corners (x_1, x_2). At run-time, the boundary locations of its corresponding shell S_k are derived from (x_1, x_2) on the fly and used to filter the Gaussians within S_k for rendering the pixels within B_k.

Results and discussion. Figure 4 illustrates the run-time efficiency of Image-GS (8K Gaussians) for a varying number of spatial partitioning blocks. The measurement is performed on an NVIDIA GeForce RTX 3080 GPU. The inference time required for rendering 10K pixels decreases from 4.59 ms for 16 blocks to 0.70 ms for 192 blocks, showing a 6.56× acceleration. Further engineering efforts, such as customized CUDA kernels, could enable substantial acceleration over our proof-of-concept implementation in pure PyTorch code but are beyond the scope of this work. Notably, storing these block coordinates for accelerated rendering only requires $4N_b$ float16 parameters, and incurs $8N_b$ bytes additional size to our representation, which is practically negligible compared to memory consumed by the Gaussian parameters. For instance, 150 blocks (1.20 KB) only account for 0.93% memory consumption of an Image-GS-represented

Table 1. *Trade-off between visual fidelity and memory efficiency.* We adjust the size of Image-GS by varying the number of 2D Gaussians therein ("nK"). ↑/↓ symbols indicate that higher/lower values are better.

Method	PSNR↑	SSIM↑	LPIPS↓	FLIP↓	Size↓
Ours (1K)	24.48	0.7946	0.2734	0.1748	16 KB
Ours (2K)	26.80	0.8316	0.2076	0.1392	32 KB
Ours (4K)	29.43	0.8706	0.1573	0.1074	64 KB
Ours (6K)	30.85	0.8828	0.1310	0.0967	96 KB
Ours (8K)	32.19	0.8912	0.1176	0.0866	128 KB

Table 2. *Quantitative comparison with previous neural image representations.* We measure the representation efficiency of Image-GS in terms of visual quality and memory consumption against four baseline methods. All metrics are computed on images rendered at 2K×2K resolution.

Method	PSNR↑	SSIM↑	LPIPS↓	FLIP↓	Size↓
ReLU	23.25	0.7148	0.4288	0.2012	132 KB
SIREN	27.48	0.7760	0.3662	0.1613	135 KB
WIRE	26.53	0.6996	0.4062	0.2117	134 KB
I-NGP	27.12	0.7815	0.2641	0.1786	141 KB
Ours	32.19	0.8912	0.1176	0.0866	128 KB

image with 8K Gaussians. These results demonstrate the scalability of our acceleration approach via hierarchical spatial partitioning.

4.3 Visual Fidelity vs Memory Efficiency

We optimize Image-GS to the 22 non-texture images in our evaluation dataset under varying memory consumption. By adjusting the number of 2D Gaussians, we obtain 5 different bit-rate levels (in bpp) for Image-GS: 0.244, 0.183, 0.122, 0.061, 0.031.

Results and discussion. For all 5 bit-rate levels, we render Image-GS-represented images at 2K×2K resolution and evaluate their visual fidelity against the reference images. As summarized in Table 1, Image-GS achieves an average performance of 32.19 (PSNR), 0.89 (SSIM), 0.12 (LPIPS), and 0.09 (FLIP) with a memory size of 128 KB. Even at an ultra-low bit rate of 0.031 bpp (16 KB), Image-GS is able to fit the target images at 24.48 (PSNR), 0.79 (SSIM), 0.27 (LPIPS), and 0.17 (FLIP) on average. Notably, our progressive optimization strategy (Section 3.3) automatically generates a sequence of Image-GS-represented images at varying bit rates during training. This forms a natural level of detail (LoD) stack for the target image. Figure 6 shows several samples rendered at 2K×2K resolution.

4.4 Comparison with Neural Image Representations

Baseline methods. We establish comparisons with 4 recent neural image representations: ReLU fields [Karnewar et al. 2022], SIREN [Sitzmann et al. 2020], WIRE [Saragadam et al. 2023], and Instant NGP [Müller et al. 2022]. For fair comparisons under similar bit rates, we modify these baseline models to match our bit rates by decreasing the resolution of feature grids (ReLU, Instant NGP) and/or reducing the number of hidden layers/features (Instant NGP, WIRE, SIREN). The 22 non-texture images in our evaluation dataset are employed.

Table 3. *Quantitative comparison with GPU texture compressors.* All metrics are computed on textures rendered at 2K×2K resolution. We use the mipmap level 2 for BC1/BC7 to match their bit rates to ours.

Method	PSNR↑	SSIM↑	LPIPS↓	FLIP↓	Bit Rate↓
BC1	28.86	0.8714	0.2369	0.1012	0.250 bpp
BC7	27.31	0.8425	0.3006	0.1054	0.253 bpp
Ours	33.03	0.8922	0.1591	0.0757	0.244 bpp

Results and discussion. As summarized in Table 2, our Gaussian-based representation Image-GS achieves an average performance of 32.19 (PSNR), 0.89 (SSIM), 0.12 (LPIPS), and 0.09 (FLIP), significantly outperforming all baseline methods despite a smaller memory footprint. Figure 7 shows several visual examples. At an ultra-low bit rate of 0.244 bpp, all baselines show different levels of image distortions and artifacts. For instance, decreasing the resolution of grid-based methods (ReLU, Instant NGP) leads to block artifacts due to feature vectors being interpolated at sparser locations. Implicit functions (SIREN, WIRE) exhibit artifacts such as ringing or blurring, which are more pronounced after reducing the base network parameters. By contrast, Image-GS exhibits much less visible artifacts.

4.5 Comparison with Texture Compression Methods

Baseline methods. We compare to two BCx variants designed for RGB texture compression, BC1 (highest settings) and BC7 (quality 0.25), using their AMD Compressonator [AMD 2024] implementations. Since these settings only offer a maximum compression of up to 4 bpp, we use the compressed images in the mipmap level 2 for both algorithms and upsample them back to the original resolution using bilinear interpolation to match our 0.244 bpp for fair comparisons. The 8 texture maps in our evaluation dataset are employed.

Results and discussion. As summarized in Table 3, our Gaussian-based representation Image-GS achieves an average performance of 33.03 (PSNR), 0.89 (SSIM), 0.16 (LPIPS), and 0.08 (FLIP), consistently outperforming the two baseline methods despite a smaller memory footprint. Figure 5 presents visual comparisons of Image-GS versus BC1 and BC7. The error maps indicate that Image-GS shows visibly less distortions from the uncompressed reference images. More samples can be found in Figures 8 and 9 in the appendix.

5 POTENTIAL APPLICATIONS

5.1 Adaptive Image Representation for Machine Vision

Deep-learning techniques have found tremendous success across a wide range of machine vision tasks, such as object tracking, facial recognition, and depth estimation. These achievements typically require high-resolution images as inputs for large neural network models to extract relevant information. While images represent visual information using a uniform grid structure, the task-related information often only resides in a few scattered local image regions exhibiting distinct features. Such spatially uniform representation for visual inputs results in an efficient usage of processing time and energy consumption. By contrast, Image-GS adaptively distributes

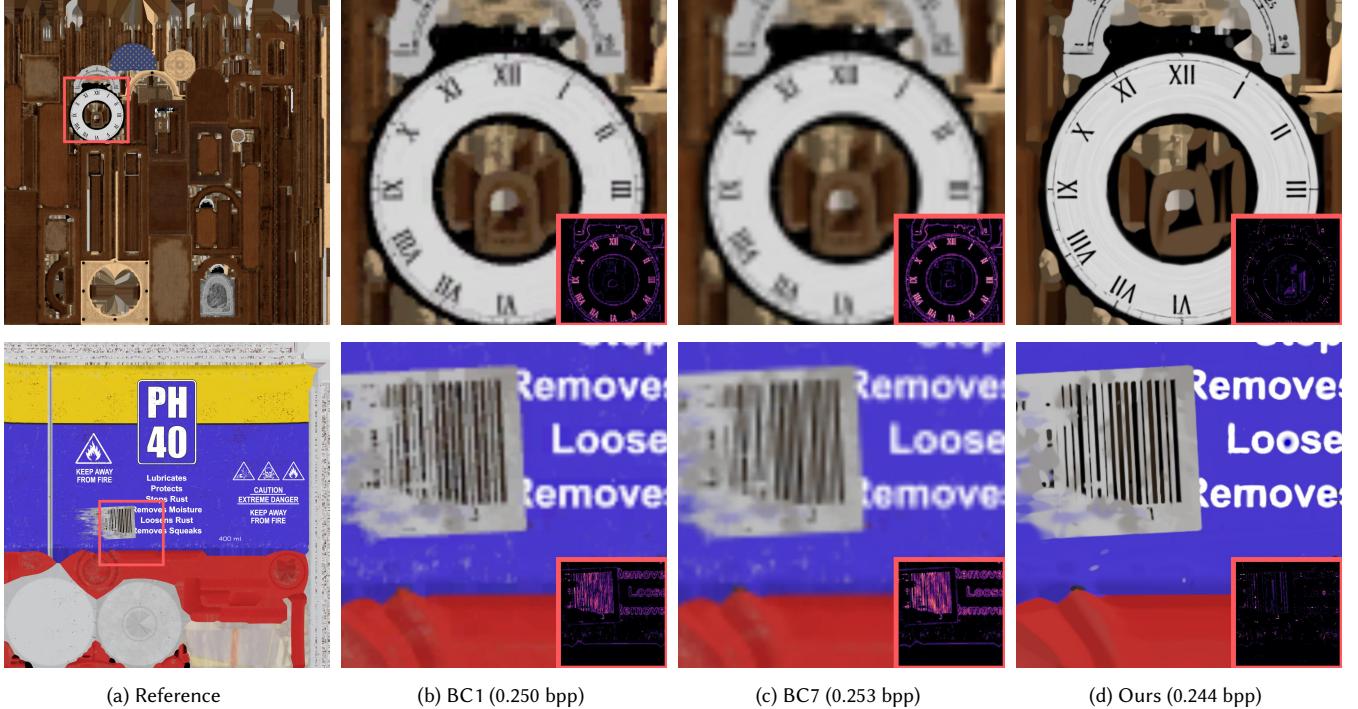| (a) Reference | (b) BC1 (0.250 bpp) | (c) BC7 (0.253 bpp) | (d) Ours (0.244 bpp) |

Fig. 5. *Qualitative comparison with GPU texture compressors.* For comparisons under similar compression rates, we use the mipmap level 2 for both BC1 and BC7 to match their bit rates to ours. The bottom-right insets visualize the corresponding error images, with brighter colors indicating higher errors.

representation resources across the image domain, with more bits allocated to image regions showing fine details. Compared to standard images, our Gaussian-based representation offers a more flexible and efficient encoding of visual inputs for machine vision models to operate on. Directly running machine vision models on Image-GS-represented images may improve their computation efficiency.

5.2 Resource-Adaptive Image Representation

The only trainable parameters in Image-GS are the ones associated with the 2D Gaussians. Consequently, its memory and computation requirements solely depend on and linearly scale with the number of 2D Gaussians. This provides a flexible way to trade off between visual fidelity and run-time efficiency. Moreover, our progressive optimization strategy with error-guided Gaussian addition produces a sequence of Image-GS-represented images at varying bit rates during training, which forms a natural level of detail stack for the target image. These properties of Image-GS are useful in situations where computational resources or network bandwidths are limited, such as mobile computing or web album streaming. While conventional compression algorithms may uniformly degrade the visual quality across the entire image domain, Image-GS is capable of maximizing the usage of available resources for optimized visual quality.

5.3 Image Restoration and Enhancement

The inherent low-frequency nature of Gaussian functions makes our Image-GS representation robust to various high-frequency image distortions and artifacts caused by JPEG compression (blockiness, ringing, and contouring), quantization errors (color banding and aliasing), transmission over noisy channels and low-light imagery (salt and pepper noise). We empirically observe in our experiments that, when using Image-GS to represent images containing such high-frequency artifacts, Image-GS effectively eliminates them during optimization and outputs artifact-free rendering. These observations suggest that Image-GS has the potential to accomplish certain image restoration and enhancement tasks.

6 LIMITATIONS AND FUTURE WORK

Hierarchical spatial guidance during optimization. While Image-GS is designed to be content-adaptive, numerical optimization algorithms, such as stochastic gradient descent, sometimes have trouble shifting the spatial distribution of Gaussians toward the global optimum. Following the acceleration technique in Section 3.4, we plan to introduce a dynamic BSP tree that guides the spatial allocation of Gaussians. During training, the tree structure and Gaussian parameters can be jointly optimized by formulating and solving an integer programming problem similar to [Martel et al. 2021].

Dynamic visual content. In this work, we demonstrate that images can be adaptively and efficiently represented with an explicit basis, anisotropic 2D Gaussian. Motivated by recent research that incorporates dynamics into Gaussian-based 3D scene representations [Luiten et al. 2023; Stavros et al. 2024], we plan to apply our method to represent videos by additionally modeling the movements of 2D

Gaussians within the image plane. We envision this extension to benefit graphics applications such as video streaming.

7 CONCLUSION

In this paper, we proposed Image-GS, a flexible, compact, and content-adaptive image representation based on anisotropic, colored 2D Gaussians. Image-GS has high memory & computation efficiency, supports fast random access, and offers a natural level of detail through progressive optimization. The content-adaptive nature of Image-GS enables it to wisely allocate resources based on the signal complexity of local image regions and preserve fine image details with higher fidelity than alternative methods. Through a series of quantitative comparisons with recent neural image representations and industry-standard texture compression algorithms, we validated the visual fidelity and memory efficiency of Image-GS. We hope this research will establish new grounds for developing new representations for visual data.

REFERENCES

2024. AMD Compressonator. https://gpuopen.com/compressonator/.

2024. Texture Block Compression in Direct3D 11. https://learn.microsoft.com/en-us/windows/win32/direct3d11/texture-block-compression-in-direct3d-11.

Jyrki Alakuijala, Ruud Van Asseldonk, Sami Boukortt, Martin Bruse, Iulia-Maria Comșa, Moritz Firsching, Thomas Fischbacher, Evgenii Kliuchnikov, Sebastian Gomez, Robert Obryk, et al. 2019. JPEG XL next-generation image compression architecture and coding tools. In *Applications of digital image processing XLII*, Vol. 11137. SPIE, 112–124.

Pontus Andersson, Jim Nilsson, Tomas Akenine-Möller, Magnus Oskarsson, Kalle Åström, and Mark D Fairchild. 2020. FLIP: A Difference Evaluator for Alternating Images. *Proc. ACM Comput. Graph. Interact. Tech.* 3, 2 (2020), 15–1.

Marc Antonini, Michel Barlaud, Pierre Mathieu, and Ingrid Daubechies. 1992. Image coding using wavelet transform. *IEEE Trans. Image Processing* 1 (1992), 20–5.

Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. 2018. Variational image compression with a scale hyperprior. *arXiv preprint arXiv:1802.01436* (2018).

Yash Belhe, Michaël Gharbi, Matthew Fisher, Iliyan Georgiev, Ravi Ramamoorthi, and Tzu-Mao Li. 2023. Discontinuity-Aware 2D Neural Fields. *ACM Transactions on Graphics (TOG)* 42, 6 (2023), 1–11.

Graham Campbell, Thomas A DeFanti, Jeff Frederiksen, Stephen A Joyce, Lawrence A Leske, John A Lindberg, and Daniel J Sandin. 1986. Two bit/pixel full color encoding. *ACM SIGGRAPH Computer Graphics* 20, 4 (1986), 215–223.

Guikun Chen and Wenguan Wang. 2024. A survey on 3d gaussian splatting. *arXiv preprint arXiv:2401.03890* (2024).

Yinbo Chen, Sifei Liu, and Xiaolong Wang. 2021. Learning continuous image representation with local implicit image function. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 8628–8638.

Yue Chen, Debargha Murherjee, Jingning Han, Adrian Grange, Yaowu Xu, Zoe Liu, Sarah Parker, Cheng Chen, Hui Su, Urvang Joshi, et al. 2018. An overview of core coding tools in the AV1 video codec. In *2018 picture coding symposium (PCS)*. IEEE, 41–45.

Chang-Chieh Cheng. 2024. Image representation and reconstruction by compositing Gaussian ellipses. *IET Image Processing* 18, 2 (2024), 493–506.

Edward Delp and O Mitchell. 1979. Image compression using block truncation coding. *IEEE transactions on Communications* 27, 9 (1979), 1335–1342.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*.

Alexander Gepperth and Benedikt Pfülb. 2021. Image modeling with deep convolutional gaussian mixture models. In *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–9.

Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2024. 2D Gaussian Splatting for Geometrically Accurate Radiance Fields. *arXiv preprint arXiv:2403.17888* (2024).

Animesh Karnewar, Tobias Ritschel, Oliver Wang, and Niloy Mitra. 2022. Relu fields: The little non-linearity that could. In *ACM SIGGRAPH 2022 Conference Proceedings*. 1–9.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–14.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Georgios Kopanas, Thomas Leimkühler, Gilles Rainer, Clément Jambon, and George Drettakis. 2022. Neural point catacaustics for novel-view synthesis of reflections. *ACM Transactions on Graphics (TOG)* 41, 6 (2022), 1–15.

Georgios Kopanas, Julien Philip, Thomas Leimkühler, and George Drettakis. 2021. Point-Based Neural Rendering with Per-View Optimization. In *Computer Graphics Forum*, Vol. 40. Wiley Online Library, 29–43.

Mengtian Li, Yu-Xiong Wang, and Deva Ramanan. 2020. Towards streaming perception. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*. Springer, 473–488.

Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. 2019. Neural volumes: learning dynamic renderable volumes from images. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–14.

Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. 2023. Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. *arXiv preprint arXiv:2308.09713* (2023).

Julien NP Martel, David B Lindell, Connor Z Lin, Eric R Chan, Marco Monteiro, and Gordon Wetzstein. 2021. Acorn: adaptive coordinate networks for neural scene representation. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–13.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant neural graphics primitives with a multiresolution hash encoding. *ACM transactions on graphics (TOG)* 41, 4 (2022), 1–15.

Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. 2020. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 3504–3515.

Jörn Nystad, Anders Lassen, Andy Pomianowski, Sean Ellis, and Tom Olson. 2012. Adaptive scalable texture compression. In *Proceedings of the Fourth ACM SIGGRAPH/Eurographics Conference on High-Performance Graphics*. 105–114.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019), 8026–8037.

Vishwanath Saragadam, Daniel LeJeune, Jasper Tan, Guha Balakrishnan, Ashok Veeraraghavan, and Richard G Baraniuk. 2023. Wire: Wavelet implicit neural representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 18507–18516.

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. 2020. Implicit neural representations with periodic activation functions. *Advances in neural information processing systems* 33 (2020), 7462–7473.

Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhofer. 2019a. Deepvoxels: Learning persistent 3d feature embeddings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2437–2446.

Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. 2019b. Scene representation networks: Continuous 3d-structure-aware neural scene representations. *Advances in Neural Information Processing Systems* 32 (2019).

Stavros Stavros, Tobias Zirr1, Alexandr Kuznetsov, Georgios Kopanas, and Anton Kaplanyan. 2024. N-Dimensional Gaussians for Fitting of High Dimensional Functions. In *ACM SIGGRAPH 2024 Conference Proceedings*. 1–9.

Jacob Ström and Tomas Akenine-Möller. 2005. i PACKMAN: High-quality, low-complexity texture compression for mobile phones. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. 63–70.

Jacob Ström and Martin Pettersson. 2007. ETC 2: texture compression using invalid combinations. In *Graphics Hardware*, Vol. 7. 49–54.

Jiakai Sun, Han Jiao, Guangyuan Li, Zhanjie Zhang, Lei Zhao, and Wei Xing. 2024. 3dgstream: On-the-fly training of 3d gaussians for efficient streaming of photo-realistic free-viewpoint videos. *arXiv preprint arXiv:2403.01444* (2024).

Jianjun Sun, Yan Zhao, and Shigang Wang. 2019. Image compression using GMM model optimization. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 1797–1801.

Jianjun Sun, Yan Zhao, Shigang Wang, and Jian Wei. 2021. Image compression based on Gaussian mixture model constrained using Markov random field. *Signal Processing* 183 (2021), 107990.

Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. 2020. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in neural information processing systems* 33 (2020), 7537–7547.

Karthik Vaidyanathan, Marco Salvi, Bartlomiej Wronski, Tomas Akenine-Moller, Pontus Ebelin, and Aaron Lefohn. 2023. Random-Access Neural Compression of Material Textures. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–25.

Gregory K Wallace. 1992. The JPEG still picture compression standard. *IEEE transactions on consumer electronics* 38, 1 (1992), xviii–xxxiv.

Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing* 13, 4 (2004), 600–612.

Terry A Welch. 1985. High speed data compression and decompression apparatus and method. US Patent 4,558,302.

Wang Yifan, Felice Serena, Shihao Wu, Cengiz Öztireli, and Olga Sorkine-Hornung. 2019. Differentiable surface splatting for point-based geometry processing. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–14.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 586–595.
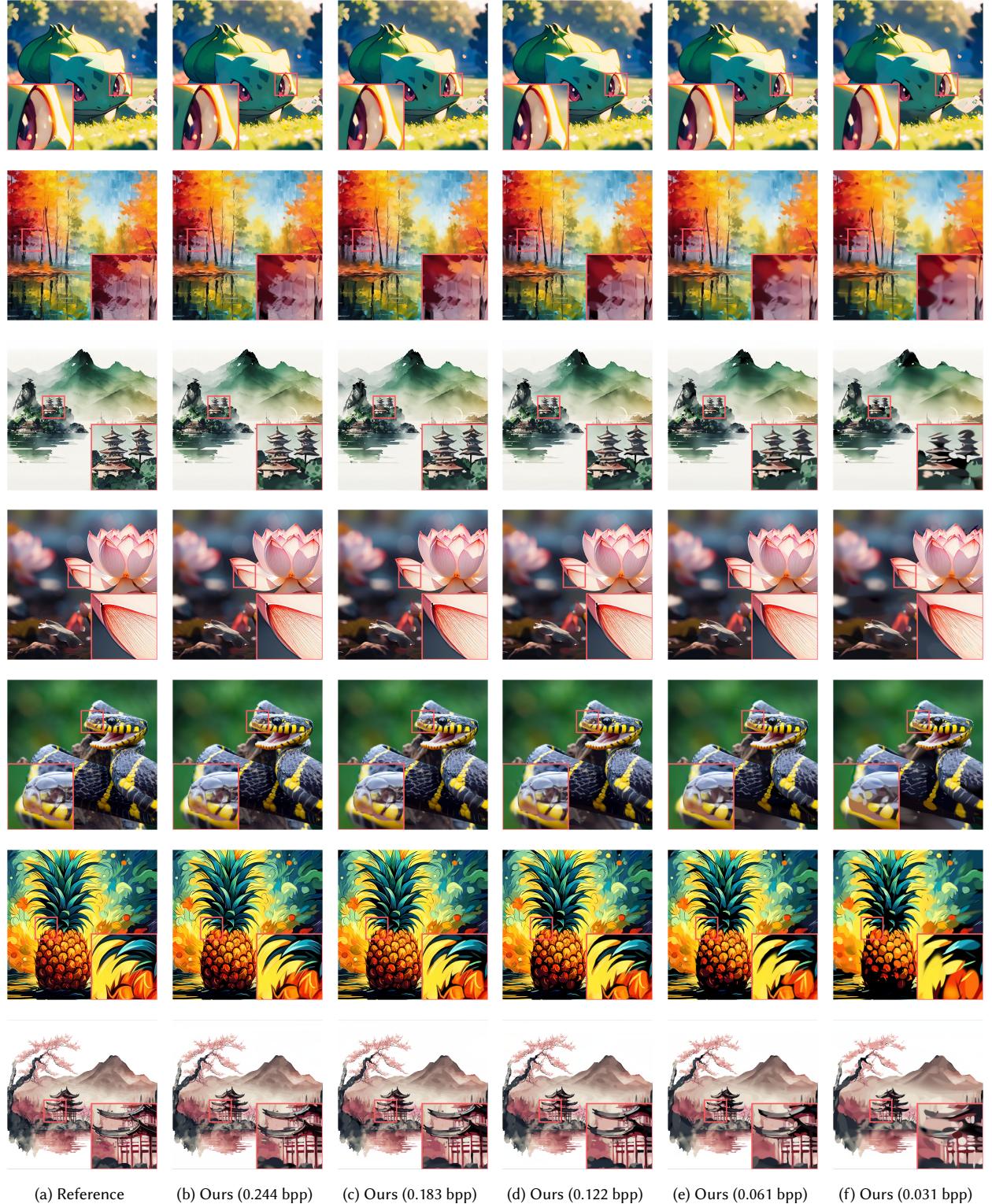
(a) Reference   (b) Ours (0.244 bpp)   (c) Ours (0.183 bpp)   (d) Ours (0.122 bpp)   (e) Ours (0.061 bpp)   (f) Ours (0.031 bpp)

Fig. 6. *Illustration of the trade-off between visual quality and memory/computation efficiency with Image-GS.* The bit rate of Image-GS is adjusted by controlling the maximum number of 2D Gaussians that are allowed. Notably, our progressive optimization with error-guided Gaussian addition generates a sequence of Image-GS-represented images at varying bit rates along the way, which forms a natural level of detail for the target image.

(a) Reference   (b) ReLU fields   (c) SIREN   (d) WIRE   (e) Instant NGP   (f) Ours

Fig. 7. *Qualitative comparison with previous neural image representations.* Our evaluation dataset covers various image types, including photographs, water-color/oil paintings, anime posters, and vector-style images. Notably, the content-adaptive nature of Image-GS enables it to wisely allocate resources based on the complexity of local image regions and better preserve fine image details than the baseline methods under similar memory consumption. The model size (in KB) for ReLU fields, SIREN, WIRE, Instant NGP, and our Image-GS are 132, 135, 134, 141, and 128, respectively.
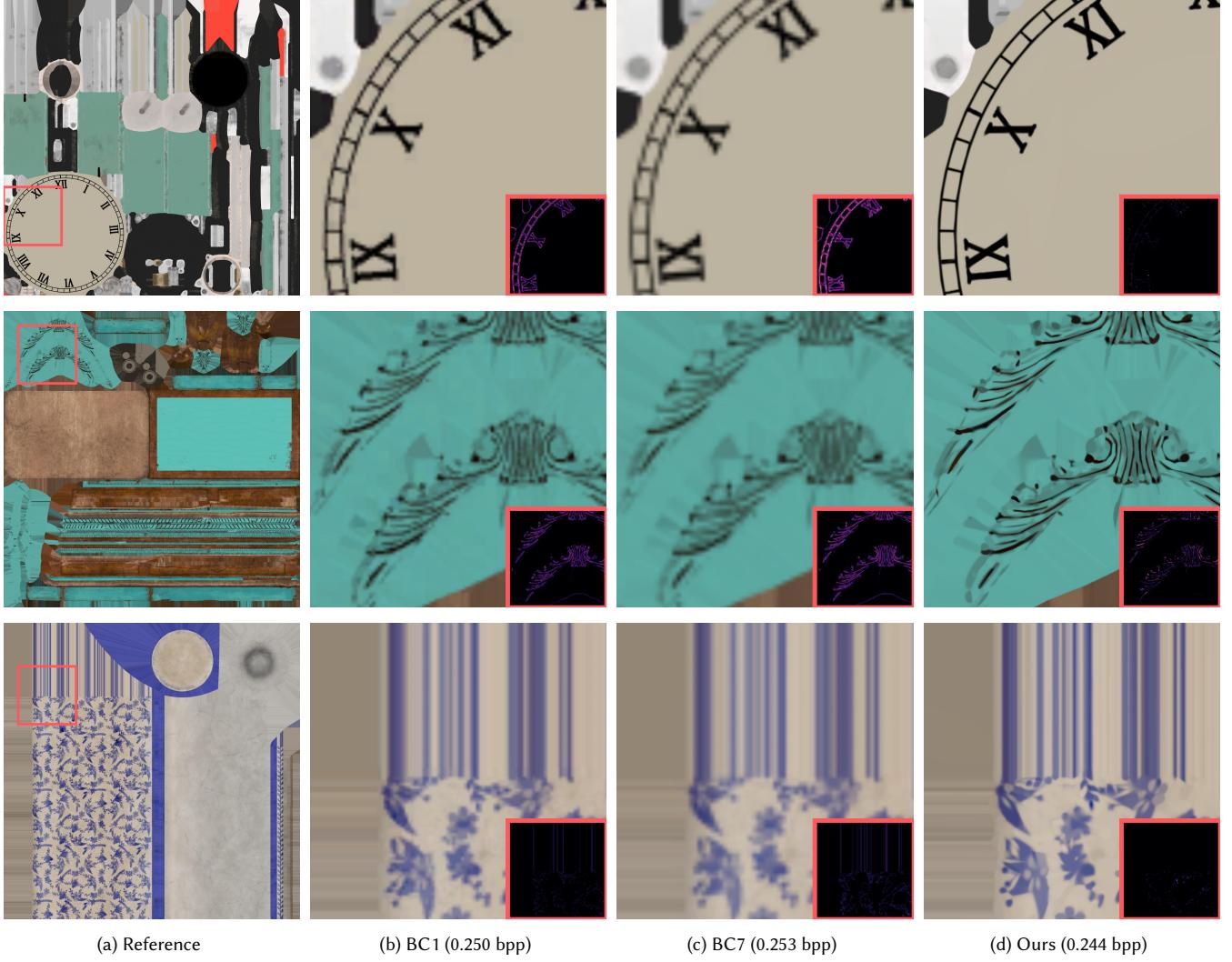
(a) Reference	(b) BC1 (0.250 bpp)	(c) BC7 (0.253 bpp)	(d) Ours (0.244 bpp)

Fig. 8. *Qualitative comparison with GPU texture compressors.* For comparisons under similar compression rates, we use the mipmap level 2 for both BC1 and BC7 to match their bit rates to ours. The bottom-right insets visualize the corresponding error images, with brighter colors indicating higher errors.
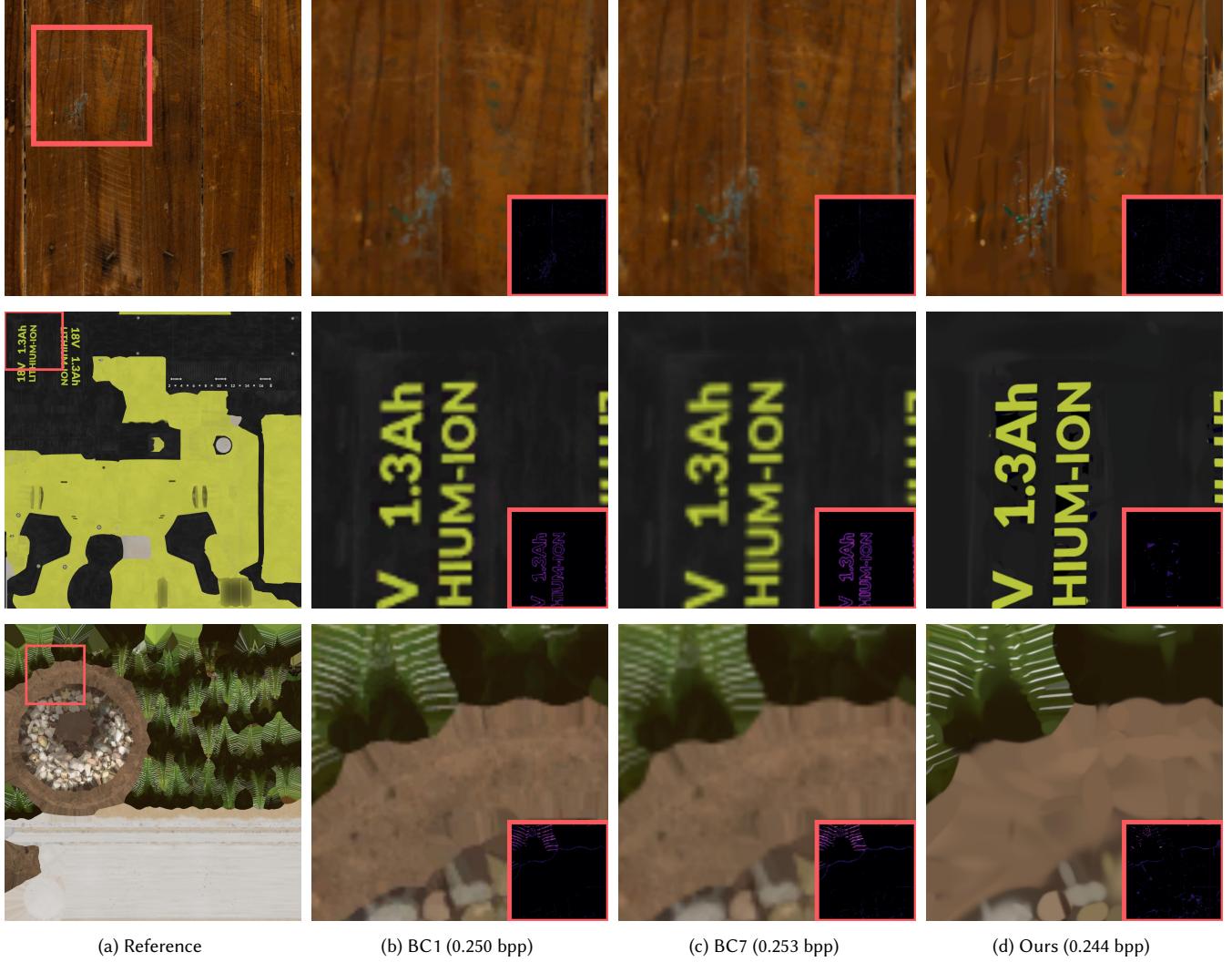
(a) Reference   (b) BC1 (0.250 bpp)   (c) BC7 (0.253 bpp)   (d) Ours (0.244 bpp)

Fig. 9. *Qualitative comparison with GPU texture compressors.* For comparisons under similar compression rates, we use the mipmap level 2 for both BC1 and BC7 to match their bit rates to ours. The bottom-right insets visualize the corresponding error images, with brighter colors indicating higher errors.