ESVR: 3D Ellipsoid-based Sparse Volume Rendering via Structure-aware Primitive Learning and Per-primitive Ray Sampling

Suemin Jeon*, Youjin Kim*, Jungwoo Park, Kyungryun Lee, and Won-Ki Jeong†

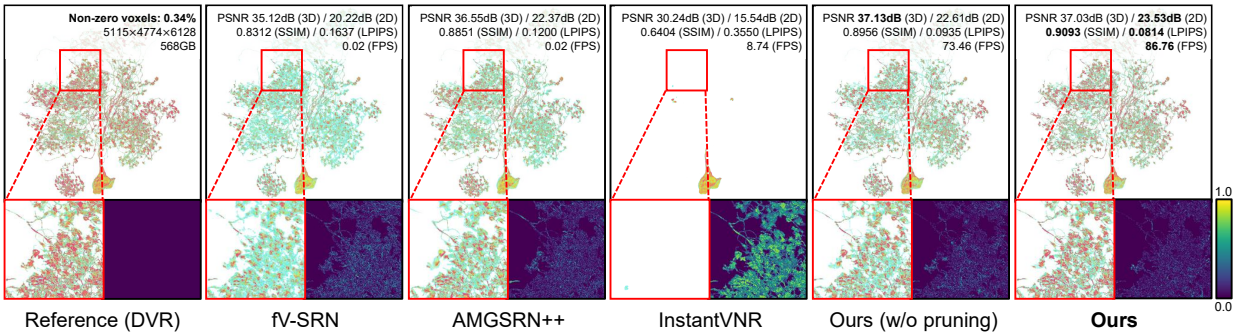


Fig. 1: Rendering results on the Hemibrain (ILN2P) dataset under a model-size-matched setting (5.8MB). Our method reconstructs detailed sparse structures more accurately while maintaining real-time framerates. Top-right text shows each method's 3D PSNR, 2D PSNR, SSIM, LPIPS, and FPS. The bottom row shows a zoomed-in view and the corresponding 2D absolute per-pixel error map.

Abstract—Efficient representation and rendering of large-scale sparse volumetric data remain challenging in scientific visualization, as meaningful structures often occupy only a small fraction of the spatial domain. While direct volume rendering (DVR) provides high-quality visualization, its computational and memory costs scale poorly with data size. Recent advances in 3D Gaussian Splatting (3DGS) address this challenge by representing volumetric scenes with compact geometric primitives, enabling efficient, high-fidelity rendering. However, existing 3DGS-based methods learn from DVR rendered images rather than raw volumes, leading to information loss and limiting flexible transfer function control for interactive exploration. To address these limitations, we propose ESVR, an ellipsoid-based sparse volume rendering framework that directly learns and renders volumetric data in 3D space. Our method combines differentiable ellipsoidal primitives with bounded support, structure-aware primitive learning with complementary pruning, and a per-primitive ray sampling strategy for fast and accurate transfer function mapping. To support large-scale datasets, we further introduce a chunk-based optimization scheme with ghost ellipsoids, providing boundary context during training. Across large sparse datasets, ESVR achieves up to four orders of magnitude compression and real-time rendering at 43–223 FPS while maintaining competitive reconstruction quality.

Index Terms—3D Gaussian Splatting, Scientific Visualization, Volume Visualization

1 Introduction

In scientific visualization, efficiently representing and rendering large-scale sparse volumetric data remains a long-standing challenge [10, 16, 19, 21, 39, 46, 54, 56]. In many real-world datasets, such as vascular structures in medical images and neuronal structures in microscopy images, meaningful signals occupy only a small fraction of the spatial domain, while most voxels are empty or homogeneous (see Fig. 1, only 0.34% of the volume contains non-zero values). As a result, conventional direct volume rendering (DVR) [22, 27] incurs high computational and memory overhead due to dense sampling across uninformative regions. Prior work addresses this inefficiency using specialized data structures and empty-space skipping [11, 31, 47, 57]. However, these approaches remain tied to grid representations, making them resolution-dependent and limiting adaptability to underlying data structures.

Recently, learning-based scene representation methods have been widely adopted in volume visualization (VolVis) as an alternative to conventional grid-based representations. One of them is Scene Representation Networks (SRNs) [12, 28, 30, 48–50, 52], which compress the original volume into compact neural representations. While these approaches reduce memory and eliminate direct access to raw volumes during rendering, their dense per-point evaluation introduces substantial computational cost, hindering real-time performance. Another learning-based method, 3D Gaussian Splatting (3DGS) [17], has shown strong performance in scene representation by using spatially adaptive primitives and efficient hardware rasterization. These properties, along with resolution independence and elimination of repeated network inference, make 3DGS promising for VolVis [2, 20, 34, 42, 44].

The motivation of this work arises from the observation that Gaussian primitives are well-suited for representing sparse volumetric data, as they can be placed adaptively only in regions containing meaningful signal, enabling efficient storage and rendering. However, current 3DGS-based volume visualization methods face several limitations: First, existing methods rely on image-level supervision from DVR-rendered images generated with fixed transfer functions (TFs), which inevitably causes information loss and limits interactive exploration of the raw input volume. Several works address this limitation by combining multiple Gaussian fields constructed with different TFs [42, 44]. However, this strategy incurs additional memory overhead and depends on predefined scene context. Second, the original 3DGS relies on alpha blending of 2D rasterized Gaussians under a pre-classification scheme, in which each Gaussian is assigned a color prior to compositing. This formulation is not aligned with TF mapping in alpha-blended ray casting, leading to inaccurate TF evaluation and rendering artifacts, particularly in regions with overlapping primitives (Fig. 2). Third, conventional 3DGS employs heuristic adaptive density control (ADC) based on 2D projections, which fails to capture local structural characteristics of the input volume. As a result, 3DGS often generates excessive overlapping Gaussians, leading to artifacts and increased

• *Suemin Jeon, Youjin Kim, Jungwoo Park, Kyungryun Lee and Won-Ki Jeong are with Korea University. E-mail: orangeblush, zinzinyou, oscar6090, krlee0000, wkjeong@korea.ac.kr*

• * *These authors contributed equally to this work.* † *Corresponding author*

Manuscript received xx xxx. 202x; accepted xx xxx. 202x. Date of Publication xx xxx. 202x; date of current version xx xxx. 202x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.202x.xxxxxxx

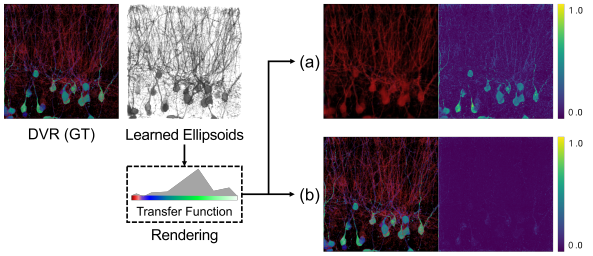


Fig. 2: Rendering results on the neuron dataset after TF application, comparing conventional (a) 2D rasterization in 3DGS with (b) our per-primitive ray sampling. The errormap shows absolute per-pixel errors.

memory consumption. Moreover, the long-tailed spatial support of Gaussians further exacerbates inter-primitive overlap and redundancy.

To address these limitations, we propose ESVR, an **E**llipsoid-based **S**parse **V**olume **R**endering framework that directly learns and renders volumetric data in 3D space. ESVR combines the memory efficiency of primitive-based scene representations with rendering quality comparable to DVR, making it particularly well suited for sparse volume rendering. Instead of Gaussian primitives, ESVR employs differentiable ellipsoidal primitives with compact spatial support, reducing inter-primitive overlap and the resulting artifacts. Building on this representation, ESVR incorporates a structure-aware primitive learning framework: ADC ensures sufficient coverage of the input volume, while complementary structure-aware pruning removes redundant primitives without sacrificing structural anisotropy or fine details. In addition, ESVR performs GPU-accelerated per-primitive ray sampling directly in 3D space, enabling fast and accurate TF evaluation without screen-space approximations as in 3DGS. To support large-scale datasets, ESVR further incorporates a chunk-based optimization scheme with ghost ellipsoids, preserving boundary context during training and enabling efficient optimization of volumes exceeding single-GPU memory. The following summarizes our main contributions:

- We propose a novel ellipsoid-based 3D volume rendering framework that represents and renders volumetric data using adaptive differentiable ellipsoidal primitives. By directly fitting compact-support ellipsoids to the input volume, the framework yields an efficient yet expressive representation for sparse volumetric data.
- We present a structure-aware primitive learning framework that combines ADC with complementary pruning for compact and efficient scene representations. We further extend this with a scalable chunk-based training scheme using ghost ellipsoids, enabling large-volume optimization without boundary artifacts.
- We develop a GPU-accelerated per-primitive ray sampling strategy that evaluates 3D ellipsoids without 2D hardware rasterization, enabling accurate TF mapping for alpha blending along rays while achieving efficient rendering.

2 Related Work

2.1 Primitive-based Volume Rendering

Primitive-based volume rendering approximates volumetric fields by splatting kernel primitives onto the image plane [6, 33, 46, 61]. Recent advances have extended this paradigm with 3DGS [17], enabling efficient rendering via spatially adaptive primitives and hardware-accelerated rasterization. Several works have applied 3DGS to volume visualization [2, 20, 34, 42, 44, 53], with improved rendering speed and compactness. However, most approaches optimize primitives from DVR-rendered images under predefined TFs, tying representations to specific TF configurations and limiting generalization to arbitrary TF manipulation. Furthermore, 3DGS relies on screen space rasterization, which is not well-aligned with volumetric signal reconstruction. This is particularly evident when primitives overlap, as compositing 2D rasterized Gaussians in screen space cannot correctly recover the underlying 3D intensity distribution. Recent efforts have partially addressed these limitations. iVR-GS [44] trains separate models across multiple TF configurations and composites them, but supports only palette-based adjustments. VEG [8] learns a scalar field from training images to enable unseen TF application, yet remains sensitive to training TF coverage and quality. Talegaonkar et al. [41] reformulate Gaussian rasterization to align with the volumetric ray integration model, and R^2-Gaussian [5] incorporates a pseudo-volume for volumetric consistency; however, both remain image-supervised without direct optimization against ground truth volumes. In contrast, our method learns from the volumetric intensity field and performs rendering in 3D space, avoiding image-based supervision and screen-space aggregation.

2.2 Neural Representation for Large Sparse Volume

Neural representations model volumetric data as continuous functions, offering an alternative to raw volumetric grids. Early approaches, commonly referred to as SRNs [12, 28, 30, 48–50, 52], enable compact representations and support downstream tasks such as temporal and spatial super-resolution [12, 43]. However, these methods are computationally expensive due to dense sampling and per-query inference [12, 28, 50]. Subsequent improvements such as latent grids [48] and hash encodings [30] significantly improved efficiency. To further address scalability, Wu et al. [49] introduce an out-of-core sampling strategy combined with macro-cell acceleration to enable interactive exploration of large volumes. Wurster et al. [51, 52] propose adaptive grid-based representations that improve efficiency for large sparse volumes. These approaches are well suited for continuous or smoothly varying volumetric signals. Despite these advances, such methods remain expensive for large-scale sparse datasets [12, 28, 50, 51], and less efficient when meaningful structures occupy only a small fraction of the spatial domain. In contrast, our method explicitly places primitives only in signal-bearing regions and retrieves information through per-primitive sampling, avoiding redundant memory access and improving efficiency for sparse volumetric data.

2.3 Improving Memory and Rendering Efficiency in 3DGS

Several works improve the efficiency of 3DGS by reducing primitive count and accelerating rendering. Methods such as Speedy-Splat [13] optimize rasterization, while compression techniques reduce the memory footprint via quantization and compact attribute encoding [35, 36]. EVER [29] introduces compact constant-density ellipsoidal primitives that better capture surface and edge structures, motivating our use of ellipsoids. However, its hard-boundary formulation is not suited to optimization against volumetric intensity fields and requires explicit ray-based accumulation. A growing body of work has sought to improve 3DGS efficiency by controlling primitive density and overlap. Prior analysis shows that densification often produces redundant primitives through cloning rather than informative splitting [15], motivating pruning-based methods [9, 14, 60]. In particular, LightGaussian [9], PUP 3D-GS [14], and LP-3DGS [60] estimate primitive importance from image-space rendering statistics to remove low-impact primitives. However, these approaches are inherently view-dependent and are therefore not directly suited to volumetric data. In contrast, our method formulates primitive importance directly in volumetric space by incorporating both signal contribution and structural characteristics. Our structure-aware pruning strategy prioritizes anisotropic, high-signal primitives, yielding compact, structurally aligned representations for sparse volumetric data, while preserving structural fidelity.

3 Preliminaries: 3DGS

In 3DGS, each primitive is a 3D Gaussian defined by its center, covariance, opacity, and color. The Gaussian function is expressed as:

$$G(x) = o \cdot e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}, \quad \Sigma = RSS^T R^T, \tag{1}$$

where $x \in \mathbb{R}^3$ denotes an arbitrary 3D position, $\mu \in \mathbb{R}^3$ is the Gaussian center, o is the opacity, and $\Sigma \in \mathbb{R}^{3\times3}$ is the covariance matrix parameterized by rotation R and scale S. During rendering, Gaussians are projected into screen space, and the color C at each pixel is computed via front-to-back alpha compositing:

$$C = \sum_{i \in N} c_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j), \tag{2}$$

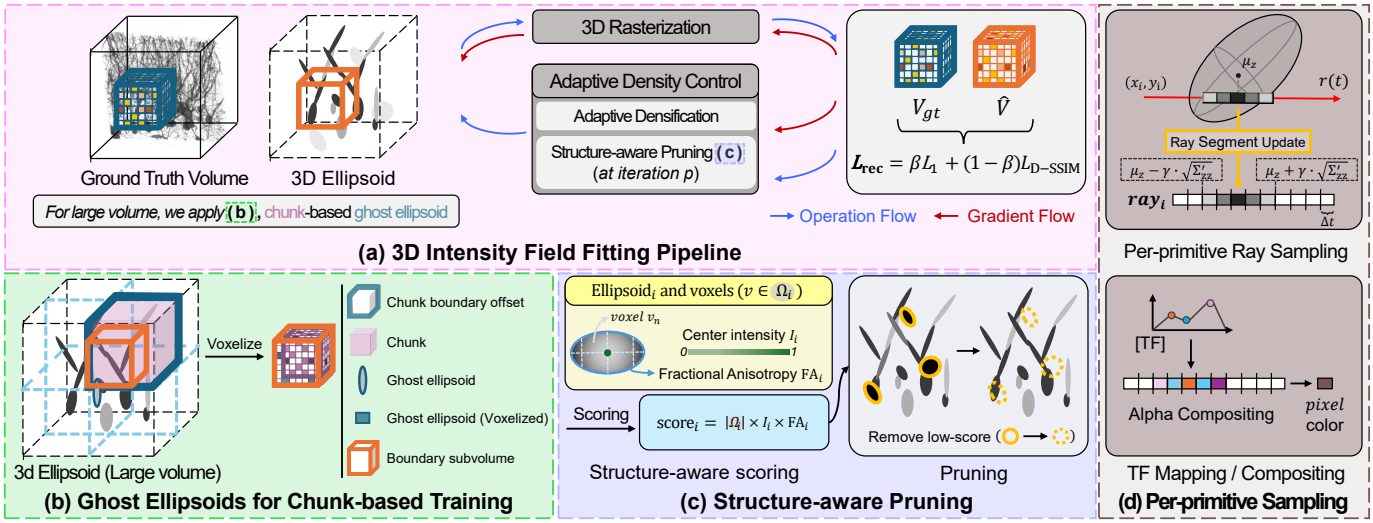


Fig. 3: Overview of our proposed framework: (a) 3D intensity field fitting pipeline, (b) chunk-based large volume training using ghost ellipsoids and (c) structure-aware pruning strategy, (d) per-primitive sampling and transfer function mapping of the intensity-fitted ellipsoids.

where N is the set of Gaussians contributing to the pixel, and c_i and α_i are the color and opacity of the i-th Gaussian after projection. 3DGS further accelerates rendering via a tile-based 2D rasterizer that depth-sorts Gaussians per tile and alpha-composites them at each pixel using shared memory parallelism. While efficient for image-based rendering, this strategy does not model volumetric ray integration [41]. 3DGS also employs ADC, increasing primitive density by cloning or splitting Gaussians based on positional gradients. While this rasterization strategy is highly efficient for image-based rendering, it does not explicitly model volumetric signal accumulation along rays.

A key limitation of 3DGS arises from the spatial support of each Gaussian. In practice, 3DGS adopts the rule of considering influence up to three standard deviations from the mean, defined as:

$$d^\top \Sigma^{-1} d \le 9, \quad d = x - \mu, \tag{3}$$

where d is the displacement vector from center μ. This 3σ support often overestimates the primitive footprint, leading to excessive overlap across primitives and increased memory and rendering cost. As a result, 3DGS produces redundant primitives misaligned with the underlying volumetric structure, motivating alternative primitive representations and learning strategies better suited for sparse volumetric signals.

4 Method

Motivated by the limitations of 3DGS discussed in Section 3, particularly excessive primitive overlap and limited ability to represent volumetric structures, we propose ESVR, an ellipsoid-based sparse volume rendering framework that directly learns and renders volumetric data in 3D space. Figure 3 illustrates the overall pipeline of ESVR. Our framework consists of two main components: (1) a structure-aware primitive learning stage that fits differentiable ellipsoids to the volumetric intensity field, and (2) a rendering stage based on per-primitive ray sampling for accurate TF evaluation. To support large-scale volumes, we further incorporate a chunk-based training scheme with ghost ellipsoids, providing boundary context across neighboring chunks during optimization and enabling scalable learning on volumes that exceed GPU memory.

4.1 Differentiable Ellipsoid Primitive

As mentioned in Section 3, long-tailed Gaussian primitives introduce excessive spatial overlap, increasing memory usage and computational cost. To address these limitations, we adopt constant-density ellipsoidal primitives inspired by EVER [29]. EVER represents primitives as indicator functions restricted to ellipsoidal boundaries:

$$P_{\text{EVER}}(x) = o \cdot \mathbf{1}_{d^\top \Sigma^{-1} d \le 1}, \tag{4}$$

where $d = x - \mu$ is the displacement from the center μ, and $\mathbf{1}_{\cdot}$ is the indicator function. Compared to the conventional Gaussian support of 3σ (Eq. 3), this formulation reduces the spatial footprint by up to $9\times$. However, the hard boundary is non-differentiable, producing vanishing gradients during optimization. EVER addresses this via ray-based accumulation, where the hard boundary is implicitly softened via transmittance integration along the ray. However, this approach is designed for image-based scene reconstruction and is not directly applicable to optimization against a 3D intensity field. To enable stable training in volumetric space, we replace the hard boundary with a differentiable sigmoid formulation ensuring that the primitive itself is fully differentiable:

$$P(x) = I \cdot \sigma\big(k \cdot (1 - d^\top \Sigma^{-1} d)\big), \tag{5}$$

where $I \in \mathbb{R}$ denotes center intensity, $\sigma(\cdot)$ is the sigmoid function and k controls boundary sharpness. However, $P(x)$ exhibits reduced peak intensity when k becomes small, deviating from the desired unit response at the center (Appendix Fig. 1 (a)). To decouple peak intensity from k, we normalize the primitive as $\tilde{P}(x) = P(x)/P(\mu)$, preserving consistent peak opacity while allowing k to control the support width. The spatial cutoff is adjusted accordingly (Appendix Fig. 1 (b)) to avoid truncation artifacts during rendering and voxelization. In practice, we use k as a fixed hyperparameter for efficiency. Although making k learnable per primitive slightly improves reconstruction quality, it introduces additional computational overhead and reduces rendering speed. Further analysis of k and its variants is provided in Appendix E.3 and E.4.

4.2 Structure-Aware Primitive Learning

Structure-aware primitive learning consists of two components: 3D intensity field fitting (Sec. 4.2.1) and structure-aware pruning (Sec. 4.2.2). During 3D intensity field fitting, we optimize ellipsoidal primitive parameters using a reconstruction loss at every iteration, while applying densification to dynamically place and refine primitives through gradient-based splitting and cloning. We refer to the combination of densification and structure-aware pruning as ADC. ADC is applied during the early optimization to increase primitive density and improve signal coverage, while pruning is performed at selected iterations to remove redundant primitives. After densification, pruning is progressively applied during subsequent optimization, enabling a compact representation while preserving structural fidelity.

4.2.1 3D Intensity Field Fitting

Unlike 3DGS, which learns view-dependent color by storing spherical harmonics (SH) coefficients and an opacity for each primitive, our framework directly models the volumetric intensity field. Here, intensity denotes the per-voxel scalar value of the original volume. Each primitive stores its center intensity I, together with geometric parameters (μ, Σ) as defined in Section 4.1. The predicted voxel intensity at x

is the sum over all primitives:

$$\hat{V}(x) = \sum_{i \in N} \tilde{P}_i(x) = \sum_{i \in N} I_i \frac{\sigma\left(k\left(1 - d_i^\top \Sigma_i^{-1} d_i\right)\right)}{\sigma(k)}, \quad (6)$$

where N is the set of primitives contributing at x, and $P_i(x) = \sigma\left(k\left(1 - d_i^\top \Sigma_i^{-1} d_i\right)\right)$ denotes the unnormalized profile of primitive i. We normalize it by its value at the primitive center, $P_i(\mu_i) = \sigma(k)$, where $d_i = 0$. Since this value is identical for every primitive, the normalization factor is simply written as the constant $\sigma(k)$. As sampling the whole volume at once is infeasible, a subvolume $V_{\text{gt}} \in \mathbb{R}^{s \times s \times s}$ is sequentially subsampled from the ground truth (GT) volume at each iteration, as illustrated in Figure 3 (a). Then, we align primitive coordinates to the GT volume coordinates and voxelize only the overlapping region to obtain the prediction subvolume $\hat{V}$. We use an L1+SSIM hybrid loss:

$$L_{\text{rec}} = \beta L_1 + (1 - \beta) L_{\text{D-SSIM}}, \quad \beta \in [0, 1]. \quad (7)$$

4.2.2 Structure-aware Pruning

While ADC improves coverage and reconstruction accuracy, it rapidly increases the number of primitives through excessive overlap, with many contributing little information and structures that could be captured by a single elongated primitive are instead fragmented into multiple isotropic ones [15]. Furthermore, elongated primitives have been shown to better preserve high-frequency structural details [17]. As illustrated in Figure 3 (b), we address this by computing a structure-aware importance score for each ellipsoid at selected pruning iterations. Unlike LightGaussian [9], which scores primitives from accumulated 2D ray contributions and is inherently view-dependent, our setting operates in volumetric space and requires an importance measure reflecting both signal contribution and structural relevance. To this end, we define a structure-aware importance score that integrates both intensity and geometric properties of each primitive, given by

$$\text{score}_i = |\Omega_i| \cdot I_i \cdot \text{FA}_i, \; \Omega_i = \{v \mid v \text{ is covered by Ellipsoid } i\}, \quad (8)$$

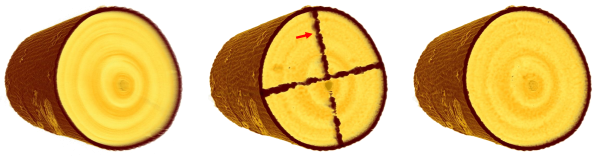
where i is the primitive index, I_i is the learned center intensity of the i-th ellipsoid, and $|\Omega_i|$ denotes its voxel coverage. A voxel v is considered *covered* if the ellipsoid yields a non-negligible contribution at its center, i.e., $\tilde{P}_i(v) \geq \varepsilon$ with $\varepsilon = 1/255$. $\text{FA}_i \in [0, 1]$ is the fractional anisotropy computed from the eigenvalues $(\lambda_{1,i}, \lambda_{2,i}, \lambda_{3,i})$ of covariance matrix Σ_i. Consequently, larger ellipsoids yield higher scores due to their broader support. The FA value ranges from 0 to 1, with higher values indicating stronger anisotropy, and is calculated as

$$\text{FA}_i = \sqrt{\frac{1}{2}} \frac{\sqrt{(\lambda_{1,i} - \lambda_{2,i})^2 + (\lambda_{2,i} - \lambda_{3,i})^2 + (\lambda_{3,i} - \lambda_{1,i})^2}}{\sqrt{\lambda_{1,i}^2 + \lambda_{2,i}^2 + \lambda_{3,i}^2}}. \quad (9)$$

The coverage term $|\Omega_i|$ scales with the spatial footprint of a primitive, while FA emphasizes anisotropic primitives that align with structural boundaries and high-frequency features [17]. Moreover, since I_i is optimized to reconstruct the ground truth field through the sum of overlapping primitive contributions, redundant fragments converge to similar local intensities and individually receive low importance scores, while the single well-fitted primitive retains a high score. At each pruning iteration, primitives are ranked by score_i and the lowest-scoring ones are removed according to a preset pruning ratio. This effect is further analyzed in Section 5.6. This formulation prioritizes high-intensity, elongated primitives while suppressing isotropic or redundant fragmented ones, improving efficiency while preserving structural fidelity.

4.3 Ghost Ellipsoids for Scalable Chunk-based Training

Direct access to the raw volume during training introduces two challenges for extremely large datasets: (1) volumes exceeding GPU memory cannot fit in a single-GPU setup; and (2) training time grows with volume resolution, as the voxelized subvolume used for loss computation becomes increasingly large. We address both through a divide-and-conquer strategy, partitioning the volume into spatially non-overlapping chunks and training each chunk on a dedicated GPU. However, training chunks separately can introduce boundary artifacts, as ellipsoids near chunk borders lack support from neighboring regions (Fig. 4). Unlike chunk-based 3DGS methods [7, 18, 24, 26], which resolve boundary inconsistencies by merging chunks into a global coarse model or through additional post-training stages, our volumetric setting requires boundary support during training itself without access to a global volume or merged model.



(a) Ground Truth (b) Train chunks separately (c) With ghost ellipsoid

Fig. 4: Visualization of Woodbranch dataset: (a) ground truth, (b) training chunks separately and (c) with ghost ellipsoids in large volume learning.

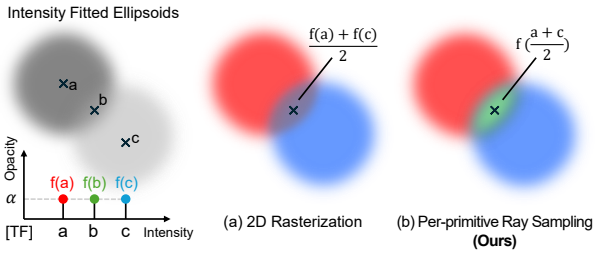


Fig. 5: Artifacts caused by inaccurate TF mapping in 2D rasterization. Vanilla 3DGS introduces blending artifacts due to incorrect TF evaluation, while per-primitive ray sampling maintains accurate TF mapping.

To address this, we introduce *ghost ellipsoids*, inspired by ghost cell techniques [45], but adapted for unstructured primitives. Unlike classical ghost cells, which copy boundary values from adjacent regular grid cells, we cannot rely on grid adjacency since our primitives are spatially unstructured. Instead, for each chunk, we identify boundary ellipsoids from all neighboring chunks (face-, edge-, and vertex-adjacent) by selecting those whose centers fall within a ghost region of width defined by the chunk boundary offset. These ghost ellipsoids are voxelized together with the current chunk's ellipsoids during loss computation but receive no gradient updates, providing boundary context without modifying neighboring chunks. We support two training modes: *sequential mode* for single-GPU training, where chunks are trained one at a time using ghost ellipsoids from completed neighbors, and *parallel mode* for multi-GPU training, where all chunks are trained simultaneously with ghost ellipsoids periodically synchronized across GPUs. Each chunk is trained in its own normalized coordinate space scaled to its spatial extent (e.g., $[-0.5, 0.5]^3$ for a volume split into $2 \times 2 \times 2$ chunks), with primitive positions stored after applying a per-chunk spatial offset that maps them directly into the shared world coordinate system. Training all chunks in a shared $[-1, 1]^3$ space and applying post-hoc rescaling would require computing $\Sigma' = K\Sigma K^T = KRSS^T R^T K^T$ for each primitive, where K is a chunk-dependent anisotropic scale matrix. Since K and R do not generally commute, this cannot be simplified by scaling S alone, and correctly decomposing Σ' back into rotation and scale parameters requires a QR decomposition per primitive, which is computationally impractical. By contrast, our approach avoids this entirely, and the resulting per-chunk primitive sets can be merged through simple concatenation without additional alignment.

4.4 Per-primitive Ray Sampling

The standard 3DGS rendering pipeline achieves efficiency via screen-space rasterization, but this design is fundamentally incompatible with accurate TF mapping. As illustrated in Figure 5 (a), the 2D pipeline blends primitive contributions in screen space after assigning colors to individual primitives (pre-classification), producing results that do not faithfully reflect the 3D intensity distribution. This limitation be-

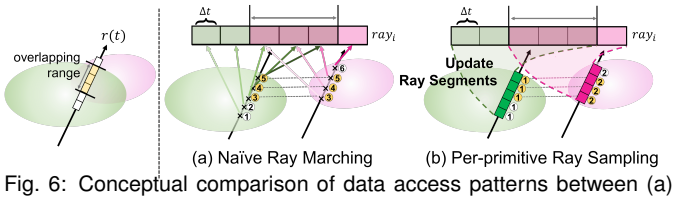


Fig. 6: Conceptual comparison of data access patterns between (a) naïve ray marching and (b) per-primitive ray sampling. In ray marching, primitive contributions are queried at each ray sample, causing repeated parameter loads. In contrast, our method processes each primitive once, determines its valid depth range, and updates only the affected ray segments, reducing redundant global memory traffic.

Algorithm 1 Ray Segment Update with Primitive Contribution

μ, Σ, r: *Primitive center, covariance, and support radius*
Δt: *sampling step*
ray_i: *ray segment*

for each primitive **do**
  $index_start \leftarrow \lfloor \frac{\mu_z - r\cdot\sqrt{\Sigma_{zz}}}{\Delta t} \rfloor$
  $index_end \leftarrow \lfloor \frac{\mu_z + r\cdot\sqrt{\Sigma_{zz}}}{\Delta t} \rfloor$
  for $j = index_start$ **to** $index_end$ **do**
    $ray_i[j] \leftarrow ray_i[j] + P(\mathbf{x}_i, \mathbf{y}_i, j\cdot\Delta t)$
  end for
end for

comes particularly problematic in regions with overlapping primitives. Our framework instead requires precise 3D sampling of the learned primitives for accurate TF evaluation (Fig. 5 (b)). We implement this via ray marching, stepping each ray through the volume and evaluating the primitive field at regular intervals. However, naively adopting this approach incurs significant overhead, as each ray step requires checking intersections with primitives and repeatedly fetching ellipsoid parameters from memory (Fig. 6 (a)). To address this, we propose a per-primitive ray sampling strategy where each ellipsoid independently updates only the ray segments it intersects (Fig. 6 (b)). By organizing computation around primitives rather than rays, we avoid redundant parameter loads and accumulate contributions in 3D space, enabling both accurate TF evaluation and real-time rendering.

Ray parameterization. Primitives are first transformed by the projection matrix, and only those visible in screen space are processed. For a given pixel position $(\mathbf{x}_i, \mathbf{y}_i)$, the associated ray is defined in ray space as $\mathbf{r}_i(t) = (\mathbf{x}_i, \mathbf{y}_i, 0) + t\cdot(0,0,1)$, where t varies along the z-axis. The ray is discretized with step size Δt into a 1D array $ray_i[\cdot]$, where the j-th entry stores the accumulated intensity of the ray segment at depth $t = j\Delta t$. Accordingly, the sample position corresponding to index j is given by $ray_i[j] = \hat{V}(\mathbf{r}_i(j\Delta t)) = \hat{V}(\mathbf{x}_i, \mathbf{y}_i, j\Delta t)$.

Primitive contribution. For each primitive with center $\mu = (\mu_x, \mu_y, \mu_z)$ and covariance Σ, only the ray segments lying within its depth range are updated. Specifically, segments within $|t - \mu_z| \le r_{supp}\cdot\sqrt{\Sigma'_{zz}}$ are considered, where $\sqrt{\Sigma'_{zz}}$ denotes the standard deviation of the primitive along the z-axis after projection, and r_{supp} is the support radius factor. For each valid segment, the primitive contributes as

$$\tilde{P}(\mathbf{x}_i, \mathbf{y}_i, j\cdot\Delta t) = I\cdot\frac{\sigma\big(k\cdot(1-\mathbf{d}^\top\Sigma^{-1}\mathbf{d})\big)}{\sigma(k)}, \tag{10}$$

where $\mathbf{d} = [\mathbf{x}_i, \mathbf{y}_i, j\cdot\Delta t] - [\mu_\mathbf{x}, \mu_\mathbf{y}, \mu_\mathbf{z}]$ is the displacement from the primitive center. Algorithm 1 summarizes the update procedure for each primitive. This eliminates explicit per-tile sorting required in 2D rasterization, since each ray segment corresponds to a discrete depth interval and contributions accumulate in depth order. Ellipsoid parameters are cached in tile-level shared memory, minimizing redundant accesses across pixels. These optimizations accelerate rendering while maintaining accurate TF mapping.

Transfer function mapping and compositing. After all primitives are processed, the accumulated ray segments ray_i contain the correct

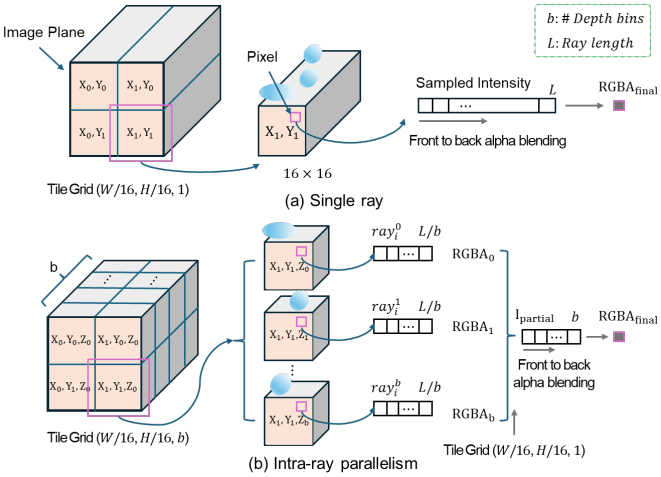


Fig. 7: Illustration of intra-ray parallelism.

3D intensity distribution along the ray. These segments are traversed front-to-back, where the TF maps non-zero intensities to color and opacity, and alpha blending (Eq. 2) produces the final pixel color

GPU-acceleration through intra-ray parallelism. Although per-primitive ray sampling improves accuracy and reduces memory access and sorting overhead, updating all ray segments still incurs significant computational cost. To further accelerate rendering, we adopt intra-ray parallelization that divides each ray into a fixed number of depth bins. Within each bin, primitive contributions are accumulated into intermediate color and transmittance values, then composited front-to-back yielding the same result as per-segment accumulation with significantly fewer redundant operations.

As illustrated in Figure 7, we extend the conventional CUDA tile grid beyond the image plane (x, y) to also span the z-axis, by partitioning the z-axis into b depth bins. This splits each pixel ray $\mathbf{r}_i$ into b partial rays. Each primitive is mapped to the 3D tiles it intersects, instead of 2D. For each primitive, we enumerate intersecting 3D tiles, generating tile–primitive keys that are used to assign primitives to depth-binned tiles. For each tile, partial color and opacity values are accumulated through per-primitive ray sampling, so each pixel holds b partial results. Finally, the tile grid is flattened back to $(x, y, 1)$ and the partial results are composited in front-to-back order across depth bins to yield the final pixel values. See Appendix Algorithm 2 for details.

This rendering framework extends the CUDA-based rendering pipeline of 3DGS and is integrated into our ESVR viewer for interactive TF editing and visualization (see Appendix Fig. 2).

5 Experiments

5.1 Setup

Dataset. We evaluate on the datasets in Table 1, covering raw single-precision floating-point volumes ranging from 65.5MB to 568GB. Datasets are categorized as small($< 1GB$), medium($\sim 1GB$), large($> 10GB$), and extremely large($> 100GB$). Sparsity is assessed by two metrics: non-zero voxel ratio and occupancy of voxels exceeding 10% of the maximum intensity. Aneurysm, Breast, and Hemibrain [38] are sparse by both metrics. Neuron, BigBrain [3], and Woodbranch have high non-zero voxel ratios but low high-intensity occupancy, indicating

Table 1: Dataset statistics.

Dataset	Resolution	Size	Non-Zero Voxel	10% Max Occupancy	Screen Occupancy
Aneurysm	$256\times256\times256$	65.5MB	0.64%	0.64%	23%
Breast	$448\times448\times160$	125.4MB	3.48%	3.48%	27%
Neuron	$2537\times1295\times133$	1.7GB	98.41%	1.84%	31%
BigBrain	$1749\times1643\times1428$	15.1GB	29.46%	29.43%	53%
Woodbranch	$2048\times2048\times2048$	32GB	84.89%	3.29%	54%
Hemibrain (vDeltaA)	$4072\times4372\times4171$	259GB	0.30%	0.29%	23%
Hemibrain (lLN2P)	$5115\times4774\times6128$	568GB	0.34%	0.34%	27%
Beechnut	$1024\times1024\times1546$	6.48GB	100.00%	100.00%	55%
Pigheart	$2048\times2048\times2612$	43.82GB	100.00%	84.60%	68%

Table 2: Quantitative comparison across seven volumetric datasets. Results are reported under two configurations: Size-M and PSNR-M—constraining storage size and reconstruction quality to match ours, respectively. Metrics include model size (MB), 3D PSNR, and rendering speed (FPS). Best results are highlighted in bold. Numbers in parentheses indicate primitive counts in thousands. OOM denotes out of memory.

		fV-SRN			NGP			AMGSRN++			InstantVNR			Ours			DVR
Dataset	**Config**	Size	PSNR	FPS	Size	PSNR	FPS	Size	PSNR	FPS	Size	PSNR	FPS	Size	PSNR	FPS	FPS
Aneurysm	Size-M	2.9	36.89	0.61	3.0	43.50	1.12	2.8	40.08	0.98	2.8	39.42	25.08	**2.6 (59K)**	**51.54**	**223.19**	137.84
	PSNR-M	64.0	42.35	0.66	62.0	46.17	1.08	64.0	43.67	0.53	66.3	43.29	16.30				
Breast	Size-M	2.9	36.87	0.90	3.0	41.19	1.48	3.3	40.22	1.35	3.1	39.79	21.78	**2.9 (67K)**	**46.00**	**213.49**	211.62
	PSNR-M	512	45.81	0.94	318	45.88	1.55	519	45.72	0.80	28.4	44.91	19.64				
Neuron	Size-M	3.3	38.33	1.77	3.0	38.97	2.87	3.3	**40.18**	2.25	2.9	38.29	24.63	**2.8 (65K)**	39.56	**131.08**	90.84
	PSNR-M	4.0	39.24	1.82	4.2	39.21	2.67	2.9	39.98	2.25	15.2	39.28	18.09				
BigBrain	Size-M	7.6	21.62	0.10	8.2	22.46	0.18	8.6	**22.75**	0.12	7.3	20.71	19.61	7.6 (176K)	21.53	**48.64**	14.44
	PSNR-M	8.8	21.74	0.10	**5.1**	21.86	0.18	6.5	22.51	0.14	14.3	21.58	15.05				
Woodbranch	Size-M	4.4	41.56	0.07	4.6	**41.72**	0.11	4.5	41.63	0.08	4.3	41.54	34.31	4.3 (99K)	40.60	**43.42**	OOM
	PSNR-M	3.6	41.55	0.07	4.1	41.70	0.12	3.9	41.59	0.09	**3.5**	41.41	32.00				
Hemibrain (vDeltaA)	Size-M	4.2	34.40	0.02	4.7	35.01	0.02	4.4	34.69	0.03	-	-	-	**4.2 (99K)**	**35.34**	**137.95**	OOM
	PSNR-M	17.9	34.97	0.02	5.5	35.29	0.02	4.93	35.30	0.02	-	-	-				
Hemibrain (lLN2P)	Size-M	6.1	35.12	0.02	6.2	36.56	0.02	6.1	36.55	0.02	7.7	30.24	8.74	**5.8 (136K)**	**37.03**	**86.76**	OOM
	PSNR-M	16.0	36.82	0.02	5.4	36.67	0.02	6.1	36.55	0.02	7.7	30.24	8.74				

Table 3: 2D rendering results under the SIZE constraint. Higher PSNR/SSIM and lower LPIPS indicate better quality. The best and second-best results are highlighted in bold and underline, respectively.

	fV-SRN			NGP			AMGSRN++			InstantVNR			Ours (w/o pruning)			Ours		
Dataset	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS
Aneurysm	23.67	0.8900	0.1098	27.86	0.9416	0.0676	25.90	0.9177	0.0946	22.47	0.8953	0.1088	**37.45**	**0.9905**	**0.0102**	31.58	0.9676	0.0402
Breast	24.38	0.8635	0.0974	26.26	0.8904	0.0803	25.15	0.8834	0.0864	27.62	0.8921	0.0831	**33.05**	**0.9590**	**0.0413**	32.22	0.9493	0.0504
Neuron	15.95	0.2805	0.6001	16.60	0.3485	0.5399	18.59	0.5113	0.4518	15.50	0.2982	0.5967	**19.32**	**0.5827**	**0.3538**	18.62	0.5239	0.4302
BigBrain	21.89	0.6560	0.2644	22.40	0.6682	**0.2374**	**22.69**	**0.6748**	0.2383	22.19	0.6589	0.2404	19.91	0.6328	0.2469	20.68	0.6281	0.2412
Woodbranch	31.02	0.8795	0.1913	32.00	**0.9148**	**0.1276**	31.38	0.8910	0.1969	31.97	0.8954	0.1490	29.04	0.8458	0.2194	**32.50**	0.8782	0.1812
Hemibrain (vDeltaA)	22.01	0.8505	0.1455	20.42	0.7873	0.2426	21.87	0.8520	**0.1334**	-	-	-	**23.93**	0.8329	0.2745	23.03	**0.8630**	0.1730
Hemibrain (lLN2P)	20.06	0.8157	0.1833	22.35	0.8755	0.1326	22.16	0.8737	0.1354	15.24	0.5819	0.4001	22.75	0.8904	0.1034	**23.54**	**0.9023**	**0.0953**

they are dense in occupancy but sparse in meaningful signal. We also report screen occupancy, the average fraction of rendered pixels covered by the volume over a fixed 360° camera trajectory. Beechnut and Pigheart are truly dense; while our primary focus is on sparse volumes, we include dense datasets for completeness. The Neuron dataset is a private dataset provided by our collaborators. See Appendix C.1 for dataset details.

Implementation. Our system is built upon the 3DGS framework [17] and the CUDA voxelizer from R^2−Gaussian [58]. All training is conducted on NVIDIA RTX A6000 GPUs, and evaluation on an NVIDIA RTX 3090. For volumes $>$ 10GB, we apply chunk-based training (8–125 chunks depending on dataset). We set $k = 5$, cutoff to 1.42, and $\beta = 0.2$. Each chunk (or full volume) is initialized with 100,000 points and partitioned into cubic subvolumes whose side length is approximately half its longest axis; subvolumes are traversed in a fixed sequential order and one epoch corresponds to a full traversal of subvolumes. Therefore, epoch counts vary with volume size. Details are provided in Appendix C.2.

Rendering and evaluation. After training, volumes are rendered using per-primitive ray sampling at 800×800 resolution with 1 voxel sampling step. FPS is measured over a full 360°rotation (0.5°increments, 720 views). We report PSNR and SSIM (higher is better), and LPIPS [59] (lower is better) for perceptual similarity. Rendering speed is measured using the full rendered view. Therefore, the multi-scale rendering framework used for acceleration is not applied to any of the comparison methods.

Baseline comparison. We compare our method against four SRN-based methods: fV-SRN, NGP, AMGSRN++, and InstantVNR. While NGP and InstantVNR share a similar backbone, InstantVNR incorporates out-of-core sampling for training large volumetric datasets, motivating its inclusion as a separate baseline. Each baseline is trained following its original protocol, with model size or PSNR matched for fair comparison. We use the neural volume renderer from AMGSRN++ to evaluate the rendering performance of fV-SRN, NGP, and AMGSRN++. For InstantVNR, we use its own framework with ray marching, sample streaming, and macro-cell acceleration. For volumes >10GB, fV-SRN and NGP exceed single-GPU memory limits; we therefore use the domain decomposition implementation from AMGSRN++ for both methods, and InstantVNR's out-of-core sampling, enabling comparison on the largest datasets. Feature grid compression is not used for AMGSRN++. We additionally compare rendering performance against standard DVR and RTX-accelerated DVR with empty space skipping (denoted as DVR (RTX)) [47], and further evaluate our method in a resource-constrained environment using a laptop GPU (NVIDIA RTX 4050 Laptop GPU). For DVR, we use the GPU-based volume rendering mode in ParaView [1]. Image-based 3DGS methods are discussed qualitatively in Section 6, as they do not support 3D PSNR evaluation or flexible TF mapping.

5.2 Quantitative Comparison

Compression and reconstruction quality. Table 2 presents a quantitative comparison across all datasets. We report results under two configurations: Size-M (model size matched to our method) and PSNR-M (reconstruction quality matched to our method). Compression performance varies with dataset sparsity. For high sparsity datasets (Aneurysm, Breast, Hemibrain), our method consistently outperforms all baselines under Size-M setting, achieving up to 8–12 dB higher 3D PSNR. Under PSNR-M setting, competing methods require substantially larger models, often 10 to 20× larger, to reach comparable quality. In some cases, they still fail to match our results. For datasets with lower sparsity (e.g., BigBrain and Woodbranch), our method achieves comparable reconstruction quality to SRN-based methods such as NGP and AMGSRN++, which are better suited for modeling smooth and dense signals. For large-scale Hemibrain datasets exceeding single-GPU memory, InstantVNR relies on out-of-core training. However, as noted by the authors, its random sampling strategy is ill-suited for sparse volumes and fails to capture thin structures. This is particularly evident in Hemibrain (vDeltaA), which contains finer structures than (lLN2P). InstantVNR fails to achieve meaningful reconstruction quality in the former and limited performance in the latter, while our method maintains consistent reconstruction quality across both. Overall, our primitive-based representation is particularly effective for sparse volumetric data, while remaining competitive on denser datasets.

Rendering performance. Our method consistently achieves real-time rendering across all datasets, while InstantVNR, despite being a dedicated acceleration of Instant-NGP for volume rendering, frequently falls below interactive frame rates on large volumes. On small to medium

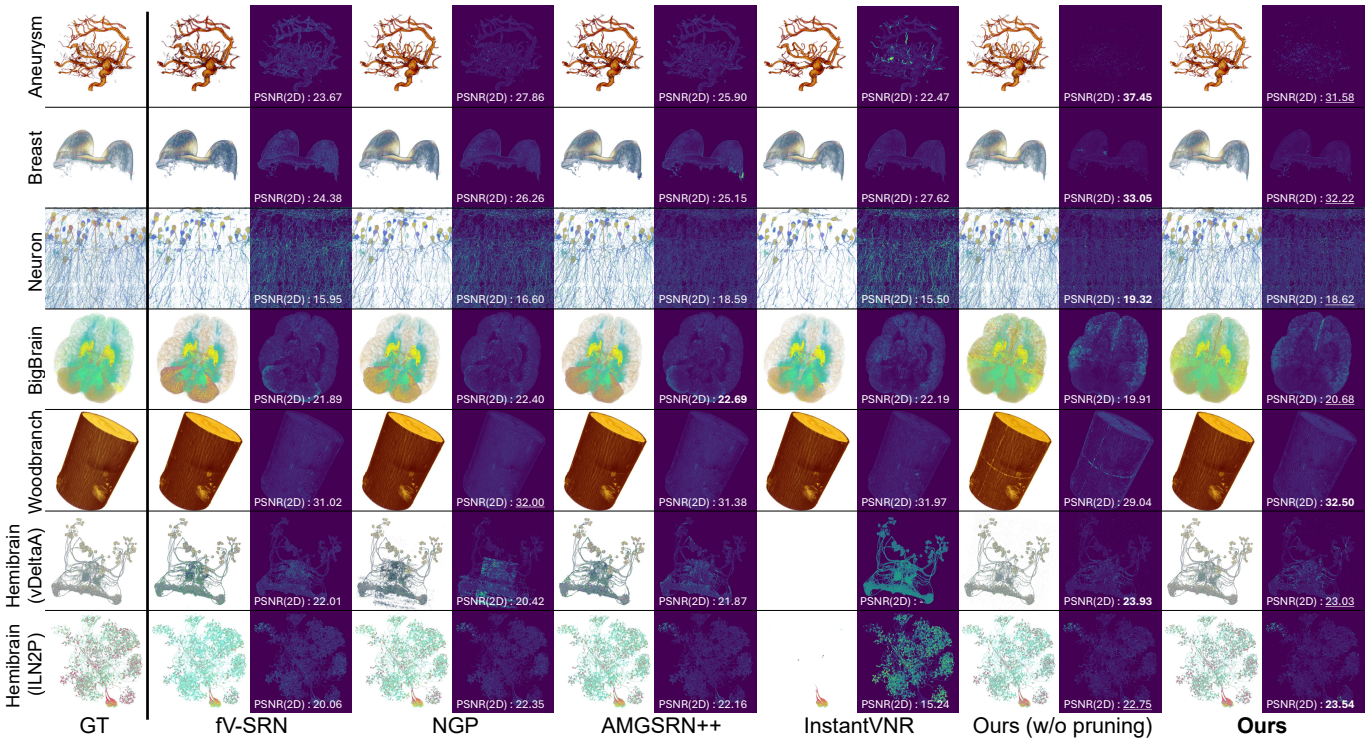


Fig. 8: Qualitative comparison of TF-mapped renderings and absolute per-pixel error maps with respect to DVR ground truth under a model-size–matched setting. Woodbranch and Hemibrain are downsampled ($\times 8$ and $\times 64$) to enable DVR (GT) rendering without OOM.

Table 4: Rendering performance and VRAM usage comparison across datasets. Measured on RTX 3090, except *Ours (Laptop)* (RTX 4050).

Dataset	Metric	Ours	Ours (Laptop)	DVR	DVR (RTX)
Aneurysm	FPS	223.19	125.08	137.84	**225.27**
	VRAM	1.64GB	1.43GB	1.49GB	1.77GB
Breast	FPS	213.49	118.25	211.62	**347.20**
	VRAM	1.64GB	1.43GB	1.51GB	1.79GB
Neuron	FPS	131.08	90.73	90.84	**164.69**
	VRAM	1.64GB	1.42GB	2.26GB	2.67GB
BigBrain	FPS	**48.64**	24.60	14.44	40.27
	VRAM	1.74GB	1.54GB	5.43GB	5.83GB
Woodbranch	FPS	**43.42**	19.51	OOM	33.91
	VRAM	1.71GB	1.50GB	OOM	18.2GB
Hemibrain (vDeltaA)	FPS	**137.95**	100.11	OOM	OOM
	VRAM	1.69GB	1.44GB	OOM	OOM
Hemibrain (lLN2P)	FPS	**86.76**	77.64	OOM	OOM
	VRAM	1.67GB	1.46GB	OOM	OOM

datasets, our renderer achieves 223, 213, and 131 FPS, well above interactive rates. On larger and relatively dense datasets (BigBrain, Woodbranch), our method outperforms InstantVNR while maintaining interactive rendering performance. As volume resolution increases to hundreds of gigabytes (e.g., Hemibrain), InstantVNR drops to low frame rates (e.g., 8 FPS), whereas our method sustains interactive performance (achieving 137 FPS and 86 FPS on Hemibrain (vDeltaA) and (lLN2P), respectively). As shown in Table 3, InstantVNR exhibits lower SSIM and higher LPIPS, especially on large-scale sparse datasets such as Hemibrain. In contrast, our method achieves competitive or superior 2D quality across all metrics without sacrificing speed.

Table 4 further compares rendering performance across a laptop GPU, standard DVR, and DVR (RTX) with empty space skipping. Our method maintains high frame rates on a mobile GPU with significantly less VRAM. Compared to conventional DVR, we consistently achieve higher frame rates across all datasets. Although DVR (RTX) outperforms our method on small sparse volumes, where hardware ray traversal is most effective, its memory usage scales poorly with resolution. DVR runs out of memory starting from Woodbranch, and DVR (RTX) from Hemibrain onward. In contrast, our method maintains a stable and low VRAM footprint across all tested datasets. Overall, these results demonstrate that our primitive-based framework compresses volumes of up to hundreds of gigabytes into compact representations while retaining competitive or superior PSNR on large sparse data, and simultaneously achieves rendering speeds well beyond real-time across all tested datasets. Together, the compression efficiency and rendering performance make our framework well-suited for interactive visualization of large-scale volumetric data.

Additional experiments are provided in the Appendix, including comparisons with image-based baselines (LightGaussian, R^2-Gaussian), traditional compression methods and VDB-based methods.

5.3 Qualitative Comparison

Figure 8 presents TF-mapped renderings and corresponding error maps under the Size-M setting. For sparse vascular structures (Aneurysm, Breast), our method preserves fine tubular structures that SRN-based methods tend to over-smooth. For datasets with high-frequency details and sparse occupancy (Neuron), our method achieves lower overall error, better preserving sharp intensity variations and thin structures. For BigBrain, which has the highest 10% max occupancy among our datasets, SRN-based methods achieve better overall reconstruction quality. However, our method shows lower error in thin and complex regions. For Woodbranch, despite its dense occupancy, our method remains competitive and often produces better renderings around surfaces and edges, where our primitive-based representation is advantageous. For the Hemibrain datasets, INR-based methods produce degraded results, failing to recover sparse structures. In particular, InstantVNR partially reconstructs Hemibrain (lLN2P) but fails on finer structures in Hemibrain (vDeltaA), while our method preserves fine structural details across both. These observations are mostly consistent with the quantitative results in Table 2 and highlight the complementary strengths of the two approaches: our primitive-based representation excels on sparse, high-frequency volumes, while continuous neural representations remain competitive on dense and smooth data.

Table 5: Training time comparison across datasets in Size-M setting. For large datasets, we report per-chunk average training time with the number of chunks in parentheses. Full training time is reported for InstantVNR, which uses out-of-core training.

Dataset	fV-SRN	NGP	AMGSRN++	InstantVNR	Ours
Aneurysm	58s	1m 52s	3m 49s	2m 18s	6m 28s
Breast	1m 24s	1m 22s	3m 58s	2m 33s	12m 36s
Neuron	1m 21s	1m 21s	4m 25s	4m 22s	22m 0s
BigBrain (8)	58s	1m 1s	4m 36s	1h 49m 54s	1h 7m 20s
Woodbranch (8)	1m 54s	1m 36s	4m 18s	1h 54m 39s	1h 34m 36s
Hemibrain (vDeltaA) (64)	1m 45s	1m 58s	1m 59s	-	49m 58s
Hemibrain (lLN2P) (125)	1m 24s	1m 36s	3m 36s	8h 29m 15s	58m 42s

Table 6: Quantitative comparison on two dense volumetric datasets under Size-M setting.

	NGP			AMGSRN++			InstantVNR			Ours		
Dataset	Size	PSNR	FPS	Size	PSNR	FPS	Size	PSNR	FPS	Size	PSNR	FPS
Beechnut	1.9	**45.46**	0.50	1.8	44.97	0.46	3.4	44.54	26.39	1.6	39.62	**43.08**
Pigheart	14.4	38.08	0.15	15.2	**38.11**	0.13	14.3	37.57	18.22	14.3	31.95	**25.10**

5.4 Training Time

Table 5 summarizes the training time across datasets under the Size-M setting. Per-chunk average training time is reported for large datasets, except for InstantVNR where full training time is reported due to its out-of-core training strategy. On small and medium datasets (Aneurysm, Breast, Neuron), our method shows longer training times than SRN-based approaches, reflecting the cost of direct volumetric optimization. This gap becomes more pronounced for larger datasets. While our chunk-based optimization enables multi-GPU parallelization, in a single-GPU setting, the total training time scales with the number of chunks, leading to an increase in overall training time as volume size grows (e.g., 22m for Neuron vs. 1h 7m for BigBrain). In contrast, InstantVNR does not adopt explicit chunking and instead processes the full volume using out-of-core sampling, resulting in substantially longer training times on large datasets (e.g., 1h49m on BigBrain and 8h29m on Hemibrain (lLN2P)) compared to other SRN-based methods. Overall, our method incurs higher training cost than SRN-based models but scales more effectively than out-of-core approaches for extremely large volumes under multi-GPU settings.

5.5 Dense Dataset

Although ESVR targets sparse volumetric data, we also evaluate our method on dense scientific datasets (e.g., Pigheart, Beechnut) to assess its scalability (Table 6). Unlike our main datasets, these volumes contain 100% non-zero regions. As expected, our method achieves lower reconstruction quality compared to network-based approaches, which are better suited for modeling dense, continuous structures and our advantage of selectively placing primitives is less pronounced. Nevertheless, our method maintains near real-time rendering performance, offering a practical trade-off between reconstruction quality and efficiency.

5.6 Ablation Study

For a fair comparison, we adjusted the densification threshold to roughly match the number of primitives across methods. As a result, some datasets show primitive counts that differ from those reported in Table 2.

Primitive design. Table 7 compares ellipsoid and Gaussian primitives at matched primitive counts. Ellipsoids consistently achieve higher PSNR and significantly faster rendering exceeding 10× those of Gaussians. This gain stems from their compact spatial support, which reduces primitive overlap and redundant computation. Synthetic dataset experiments across dense, surface-like, and tubular geometries further confirm that ellipsoids underperform Gaussians in dense regions and in structures with diffuse boundaries, while significantly outperforming them on surface-like and elongated structures in both PSNR and FPS (details in Appendix E.1). This also highlights the challenge that ellipsoids' sharp boundaries present for smooth blending compared to Gaussians' soft tails.

Table 7: Ablation study on primitive types (Gaussian vs. Ellipsoid) before and after pruning across three datasets. Numbers in parentheses indicate primitive counts. Avg PrimHits denotes the average number of primitives per voxel, computed over voxels with non-zero primitive hits.

		Ellipsoid		Gaussian	
Dataset	Metric	*Before*	*After*	*Before*	*After*
Aneurysm	Size (MB)	4.03 (93k)	1.37 (31k)	4.11 (95k)	1.40 (32k)
	3D PSNR (dB)	51.20	50.70	44.97	48.16
	3D SSIM	0.9986	0.9976	0.9979	0.9977
	FPS	66.45	248.92	6.02	34.06
	Avg PrimHits	20.10	9.05	34.73	19.96
Breast	Size (MB)	8.06 (187k)	2.74 (63k)	8.06 (187k)	2.74 (63k)
	3D PSNR (dB)	48.40	45.62	46.05	45.04
	3D SSIM	0.9983	0.9964	0.9975	0.9965
	FPS	83.82	214.14	8.99	23.08
	Avg PrimHits	13.69	6.10	24.00	11.24
BigBrain	Size (MB)	5.09 (118k)	1.73 (40k)	5.05 (117k)	1.72 (39k)
	3D PSNR (dB)	18.79	19.12	18.00	18.30
	3D SSIM	0.8129	0.8147	0.8000	0.8071
	FPS	145.69	172.68	6.63	13.69
	Avg PrimHits	6.48	5.37	22.49	12.63

Table 8: Ablation on different importance scores for structure-aware pruning. Metrics report 3D PSNR (dB) / primitive count. Results for BigBrain are reported on a single chunk.

Dataset	w/o pruning	w/o FA	only FA	Full
Aneurysm	44.91 / 32.4k	49.41 / 32.3k	50.23 / 31.8k	**50.76** / 32.3k
Breast	46.43 / 96.4k	45.46 / 95.3k	45.77 / 95.0k	**47.34** / 95.3k
BigBrain	16.66 / 32.9k	18.69 / 33.0k	18.87 / 32.8k	**18.92** / 33.0k

Table 9: Ablation on GPU intra-ray parallelism. Comparison of FPS before and after applying intra-ray binning to per-primitive ray sampling.

Method	Aneurysm (59k)	Neuron (65k)	BigBrain (176k)	Hemibrain (vDeltaA) (99k)
Single	133.95	**237.46**	41.27	144.862
Intra-ray (**Ours**)	**143.79**	231.803	**42.37**	**177.385**

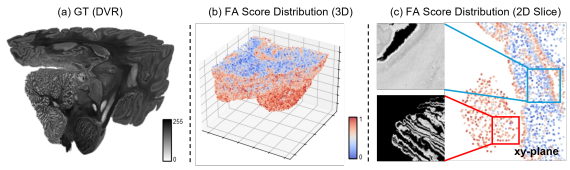


Fig. 9: Visualization of FA scores of ellipsoid primitives learned on a BigBrain chunk using our method. Red value indicates high FA score.

Pruning strategy. Table 8 presents an ablation study of structure-aware pruning across three datasets with distinct sparsity characteristics. As shown in the first column, primitives trained to represent the volume through densification alone yield worse reconstruction than our ADC pipeline, confirming that pruning redundant primitives after sufficient densification is more effective than sole densification. When removing the FA component from the importance score, reconstruction quality consistently degrades compared to using FA alone or combining FA with intensity and scale cues. The best performance is achieved when all cues are jointly considered. Qualitatively, as shown in Appendix Figure 6, FA-based scoring helps preserve fine vascular structures and small high-frequency details (highlighted in green), while intensity and scale cues contribute to maintaining dense regions (highlighted in red). Furthermore, we observe that ellipsoids with higher FA scores tend to cluster around high-frequency, fine-scale structures, suggesting that FA effectively identifies structurally informative regions (Fig. 9).

Rendering optimization. Table 9 compares intra-ray parallelism performance across four datasets. While it accelerates rendering in most cases, effectiveness varies with data characteristics: full-volume structures benefit from binning, but datasets with highly compressed z-resolution like Neuron incur merging overhead due to z-axis resolution disparity, concentrating computations in single tiles.

Additional ablations are provided in the Appendix, including primitive boundary sharpness (k) analysis, learnable k, primitive type comparison on synthetic data and number of depth bins.

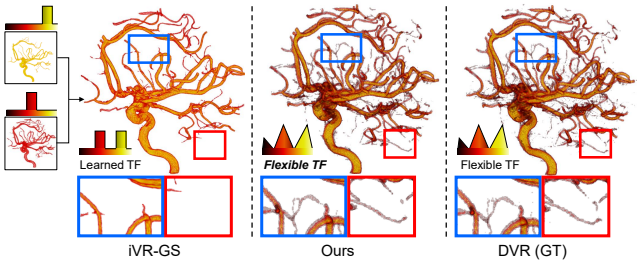


Fig. 10: Qualitative comparison with iVR-GS under unseen intensities.

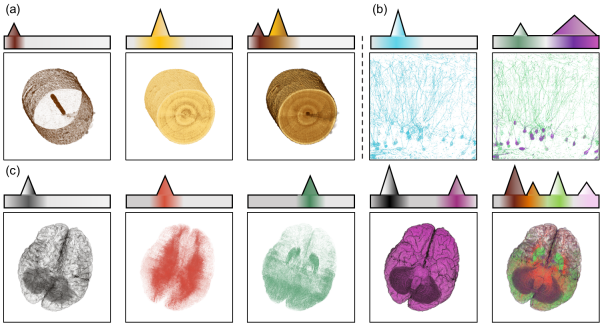


Fig. 11: Demonstration of the flexible TF capability of ESVR on three datasets : (a) Woodbranch, (b) Neuron, and (c) BigBrain.

6 Discussion

Our results show that ESVR is effective for large-scale sparse volumetric data, where meaningful structures occupy only a small fraction of the domain. By placing primitives only in signal-bearing regions, ESVR achieves strong compression while preserving fine details. Compact ellipsoidal primitives and structure-aware pruning further reduce overlap and retain structures often over-smoothed by continuous neural representations. Beyond sparse data, we also evaluate dense volumes, GPU-based empty-space skipping, and mobile GPU settings to assess the method across diverse scenarios. By directly optimizing the 3D intensity field and using per-primitive ray sampling, ESVR enables accurate TF evaluation without image-based supervision or screen-space aggregation, thereby avoiding artifacts introduced by 2D rasterization-based pipelines.

Comparison with closely related image-based methods. We compare our method with recent Gaussian-based approaches that share similar primitive representations but differ in optimization targets and rendering formulations. EVER, iVR-GS, and R^2-Gaussian all employ Gaussian or ellipsoidal primitives to represent volumetric structures. However, they are primarily designed for image-based objectives. *EVER* performs exact ray casting using ellipsoidal primitives via explicit entry/exit computation, enabling accurate rendering at the cost of significant overhead, and is not designed for VolVis. In particular, appearance is baked into the primitive parameters, preventing TF manipulation. *iVR-GS* is an image-based VolVis approach built on 3DGS, enabling interactive editing by decomposing appearance into palette-based parameters. However, TF control is limited to predefined palette combinations (Fig. 10), and flexibility depends on the number of trained models. *R^2-Gaussian* incorporates volumetric information but is designed for CT reconstruction from sparse X-ray projections. It relies on projection-based supervision and total variation regularization rather than direct volumetric reconstruction loss. As a result, it achieves high 2D projection quality but lower 3D reconstruction fidelity (Appendix Table 3). In contrast, our method directly optimizes primitives on the volumetric intensity field and supports accurate, flexible TF mapping (Fig. 11) in 3D space, enabling both compact representation and interactive visualization.

Comparison with grid-based compression methods. We further compare ESVR with grid-based compression approaches, including traditional compressors (TTHRESH [4], ZFP [25], and SZ3 [23]) and VDB-based sparse grid representations (OpenVDB [31] and NanoVDB [32]). Detailed configurations and results are provided in Appendix D.2 and D.3. Although these methods are strong baselines for compression, their representation capacity remains tied to the underlying voxel resolution. Across both groups, we observe a common trend. Grid-based methods are highly effective on small volumes, often achieving near-lossless reconstruction or high compression ratios with fast encoding. However, as the volume resolution increases, their advantage becomes less consistent. Traditional compressors can become memory- or decompression time-bound, while lossless VDB preserves the input with substantially larger storage cost. Under aggressive size-matched compression, both traditional compressors and lossy VDB remove or smooth out sparse foreground structures, even when voxel-space PSNR remains moderate. In contrast, ESVR allocates primitives directly to signal-bearing regions using adaptive ellipsoid primitives, making its storage less tied to the original grid resolution. This leads to a more favorable size-quality trade-off for large, sparse, and anisotropic volumes. In addition, traditional compressors also offer significantly faster compression (typically seconds to hours versus up to days for ESVR) (Appendix Table 6). However, unlike ESVR, they require full decompression before rendering, making them less suitable for memory-constrained interactive visualization.

Limitations. Our method has several limitations. First, while ESVR offers a rendering advantage over traditional compressors and INR-based methods, its training cost remains higher. Although chunk-based optimization enables parallelization across multiple GPUs, sequential single-GPU training of the largest volumes can take up to several days (e.g., Hemibrain (lLN2P)), which is a non-negligible bottleneck. Promising directions to reduce this cost include skipping empty subvolumes via volumetric preprocessing and exploiting a volume prior for initialization to accelerate convergence. Second, the ellipsoidal primitive is inherently better suited to sparse, anisotropic structures than to dense or smoothly varying fields. On dense and diffuse-boundary data, continuous neural and Gaussian representations achieve higher reconstruction quality, as the sharp ellipsoid support cannot blend smoothly varying signals as effectively. Since a fast rendering advantage is of limited value when reconstruction quality degrades substantially, the current primitive restricts ESVR to large, sparse volumes. Designing a primitive that remains compact while faithfully representing dense regions and diffuse boundaries is a promising step toward generalizing the framework. Third, the method relies on several hyperparameters (e.g., primitive sharpness k, pruning schedule, and subvolume size) whose sensitivity differs sharply by data type. Sparse, high-frequency structures such as Aneurysm are highly sensitive to k (a 2.2 dB PSNR drop from k=5 to k=3, Appendix Table 10), whereas dense, diffuse-boundary volumes such as BigBrain are largely k-insensitive but instead require heterogeneity-aware pruning (three pruning passes for BigBrain's cerebellar regions vs. one elsewhere, Appendix C.2). No fixed setting thus generalizes across data types, motivating data-adaptive ADC and parameter-selection strategies to improve robustness and reduce manual tuning across diverse volumetric data.

7 Conclusion and Future Work

We presented ESVR, an ellipsoid-based volume visualization framework that directly models volumetric data using compact primitives and per-primitive ray sampling. By fitting differentiable ellipsoids to the 3D intensity field and integrating structure-aware pruning, our method enables high compression, accurate TF evaluation, and real-time rendering. Extensive experiments demonstrate that ESVR is particularly effective for large-scale sparse volumes, consistently achieving significantly higher rendering performance than SRN-based approaches while maintaining competitive reconstruction quality. These properties make our framework well suited for interactive and exploratory volume visualization. Looking forward, we aim to further improve scalability and usability. Future work includes extending the framework to support out-of-core streaming, hierarchical multi-level representations, and adaptive parameter tuning. These directions will enable handling even larger datasets (e.g., terabyte-scale volumes) and broaden the applicability of primitive-based volume visualization in real-world scenarios.

Acknowledgments

This work was supported in part by the National Research Foundation of Korea under Grant RS-2024-00349697 and Grant RS-2021-NR060143; in part by the Institute for Information and Communications Technology Planning and Evaluation under Grant IITP-2026-RS-2020-II201819; in part by the National Research Council of Science & Technology(NST) grant by the Korea government (MSIT) (No. GTL24033-000); and in part by Korea University Grant.

References

[1] J. Ahrens, B. Geveci, and C. Law. ParaView: An end-user tool for large data visualization. *Visualization Handbook*, (8):717–731, 2005. doi: 10.1016/B978-012387582-2/50038-1 6

[2] K. Ai, K. Tang, and C. Wang. NLI4VolVis: Natural language interaction for volume visualization via LLM multi-agents and editable 3D Gaussian splatting. *IEEE Transactions on Visualization and Computer Graphics*, 32(1):46–56, 2026. doi: 10.1109/TVCG.2025.3633888 1, 2

[3] K. Amunts, C. Lepage, L. Borgeat, H. Mohlberg, T. Dickscheid, M.-É. Rousseau et al. BigBrain: An ultrahigh-resolution 3D human brain model. *Science*, 340(6139):1472–1475, 2013. doi: 10.1126/science.1235381 5, 13

[4] R. Ballester-Ripoll, P. Lindstrom, and R. Pajarola. TTHRESH: Tensor compression for multidimensional visual data. *IEEE Transactions on Visualization and Computer Graphics*, 26(9):2891–2903, 2020. doi: 10.1109/TVCG.2019.2904063 9, 14

[5] Y. Cai, Y. Liang, J. Wang, A. Wang, Y. Zhang, X. Yang et al. Radiative Gaussian splatting for efficient X-ray novel view synthesis. In *Proceedings of European Conference on Computer Vision*, pp. 283–299. Springer, 2024. doi: 10.1007/978-3-031-73232-4_16 2

[6] W. Chen, L. Ren, M. Zwicker, and H. Pfister. Hardware-accelerated adaptive EWA volume splatting. In *Proceedings of IEEE Visualization Conference*, pp. 67–74, 2004. doi: 10.1109/VISUAL.2004.38 2

[7] Y. Chen and G. H. Lee. DOGS: Distributed-oriented Gaussian splatting for large-scale 3D reconstruction via Gaussian consensus. In *Proceedings of Advances in Neural Information Processing Systems*, vol. 37, pp. 34487–34512, 2024. doi: 10.48550/arXiv.2405.13943 4

[8] L. Dyken, A. Sewell, W. Usher, N. Debardeleben, S. Petruzza, and S. Kumar. Volume Encoding Gaussians: Transfer function-agnostic 3D Gaussians for volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 32(6):4883–4893, 2026. doi: 10.1109/TVCG.2026.3694460 2

[9] Z. Fan, K. Wang, K. Wen, Z. Zhu, D. Xu, and Z. Wang. LightGaussian: Unbounded 3D Gaussian compression with 15x reduction and 200+ fps. In *Proceedings of Advances in Neural Information Processing Systems*, vol. 37, pp. 140138–140158, 2024. doi: 10.48550/arXiv.2311.17245 2, 4, 14

[10] D. Ganter and M. Manzke. An analysis of region clustered BVH volume rendering on GPU. *Computer Graphics Forum*, 38(8):13–21, 2019. doi: 10.1111/cgf.13756 1

[11] M. Hadwiger, A. K. Al-Awami, J. Beyer, M. Agus, and H. Pfister. SparseLeap: Efficient empty space skipping for large-scale volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):974–983, 2018. doi: 10.1109/TVCG.2017.2744238 1

[12] J. Han and C. Wang. CoordNet: Data generation and visualization generation for time-varying volumes via a coordinate-based neural network. *IEEE Transactions on Visualization and Computer Graphics*, 29(12):4951–4963, 2023. doi: 10.1109/TVCG.2022.3197203 1, 2

[13] A. Hanson, A. Tu, G. Lin, V. Singla, M. Zwicker, and T. Goldstein. Speedy-Splat: Fast 3D Gaussian splatting with sparse pixels and sparse primitives. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 21537–21546, 2025. doi: 10.48550/arXiv.2412.00578 2

[14] A. Hanson, A. Tu, V. Singla, M. Jayawardhana, M. Zwicker, and T. Goldstein. PUP 3D-GS: Principled uncertainty pruning for 3D Gaussian splatting. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5949–5958, 2025. doi: 10.48550/arXiv.2406.10219 2

[15] B. Huang, Z. Liu, and N. Wong. Decomposing densification in Gaussian splatting for faster 3D scene reconstruction. *arXiv preprint arXiv:2507.20239*, 2025. doi: 10.48550/arXiv.2507.20239 2, 4

[16] R. Kahler, M. Simon, and H.-C. Hege. Interactive volume rendering of large sparse data sets using adaptive mesh refinement hierarchies. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):341–351, 2003. doi: 10.1109/TVCG.2003.1207442 1

[17] B. Kerbl, G. Kopanas, T. Leimkuehler, and G. Drettakis. 3D Gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4):139:1–139:14, 2023. doi: 10.1145/3592433 1, 2, 4, 6

[18] B. Kerbl, A. Meuleman, G. Kopanas, M. Wimmer, A. Lanvin, and G. Drettakis. A hierarchical 3D Gaussian representation for real-time rendering of very large datasets. *ACM Transactions on Graphics*, 43(4):62:1–62:15, 2024. doi: 10.1145/3658160 4

[19] D. Kim, M. Lee, and K. Museth. NeuralVDB: High-resolution sparse volume representation using hierarchical neural networks. *ACM Transactions on Graphics*, 43(2):20:1–20:21, 2024. doi: 10.1145/3641817 1, 15

[20] C. Kleinbeck, H. Schieber, K. Engel, R. Gutjahr, and D. Roth. Multilayer Gaussian splatting for immersive anatomy visualization. *IEEE Transactions on Visualization and Computer Graphics*, 31(5):2353–2363, 2025. doi: 10.1109/TVCG.2025.3549882 1, 2

[21] M. Labschütz, S. Bruckner, M. E. Gröller, M. Hadwiger, and P. Rautek. Jittree: A just-in-time compiled sparse GPU volume data structure. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):1025–1034, 2016. doi: 10.1109/TVCG.2015.2467331 1

[22] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, 2002. doi: 10.1109/38.511 1

[23] X. Liang, S. Di, D. Tao, S. Li, S. Li, H. Guo et al. Error-controlled lossy compression optimized for high compression ratios of scientific datasets. In *Proceedings of IEEE International Conference on Big Data*, pp. 438–447, 2018. doi: 10.1109/BigData.2018.8622520 9, 14

[24] J. Lin, Z. Li, X. Tang, J. Liu, S. Liu, J. Liu et al. VastGaussian: Vast 3D Gaussians for large scene reconstruction. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5166–5175, 2024. doi: 10.48550/arXiv.2402.17427 4

[25] P. Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2674–2683, 2014. doi: 10.1109/TVCG.2014.2346458 9, 14

[26] Y. Liu, C. Luo, L. Fan, N. Wang, J. Peng, and Z. Zhang. CityGaussian: Real-time high-quality large-scale scene rendering with Gaussians. In *Proceedings of European Conference on Computer Vision*, pp. 265–282. Springer, 2024. doi: 10.1007/978-3-031-72640-8_1 4

[27] P. Ljung, J. Krüger, E. Groller, M. Hadwiger, C. D. Hansen, and A. Ynnerman. State of the art in transfer functions for direct volume rendering. *Computer Graphics Forum*, 35(3):669–691, 2016. doi: 10.1111/cgf.12934 1

[28] Y. Lu, K. Jiang, J. A. Levine, and M. Berger. Compressive neural representations of volumetric scalar fields. *Computer Graphics Forum*, 40(3):135–146, 2021. doi: 10.1111/cgf.14295 1, 2

[29] A. Mai, P. Hedman, G. Kopanas, D. Verbin, D. Futschik, Q. Xu et al. EVER: Exact volumetric ellipsoid rendering for real-time view synthesis. In *Proceedings of IEEE/CVF International Conference on Computer Vision*, pp. 4930–4939, 2025. doi: 10.1109/ICCV51701.2025.00469 2, 3

[30] T. Müller, A. Evans, C. Schied, and A. Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics*, 41(4):102:1–102:15, 2022. doi: 10.1145/3528223.3530127 1, 2, 13

[31] K. Museth. VDB: High-resolution sparse volumes with dynamic topology. *ACM Transactions on Graphics*, 32(3):27:1–27:22, 2013. doi: 10.1145/2487228.2487235 1, 9, 15

[32] K. Museth. NanoVDB: A GPU-friendly and portable VDB data structure for real-time rendering and simulation. In *Proceedings of ACM SIGGRAPH Talks*, pp. 1–2, 2021. doi: 10.1145/3450623.3464653 9, 15

[33] N. Neophytou, K. Mueller, K. T. McDonnell, W. Hong, X. Guan, H. Qin et al. GPU-accelerated volume splatting with elliptical RBFs. In *Proceedings of Joint Eurographics - IEEE VGTC Symposium on Visualization*, pp. 13–20, 2006. doi: 10.2312/VisSym/EuroVis06/013-020 2

[34] S. Niedermayr, C. Neuhauser, K. Petkov, K. Engel, and R. Westermann. Application of 3D Gaussian splatting for cinematic anatomy on consumer class devices. *arXiv preprint arXiv:2404.11285*, 2024. doi: 10.48550/arXiv.2404.11285 1, 2

[35] S. Niedermayr, J. Stumpfegger, and R. Westermann. Compressed 3D Gaussian splatting for accelerated novel view synthesis. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10349–10358, 2024. doi: 10.1109/CVPR52733.2024.00985 2

[36] P. Papantonakis, G. Kopanas, B. Kerbl, A. Lanvin, and G. Drettakis. Reducing the memory footprint of 3D Gaussian splatting. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 7(1):16:1–16:17, 2024. doi: 10.1145/3651282 2

[37] S. M. Plaza, J. Clements, T. Dolafi, L. Umayam, N. N. Neubarth, L. K. Scheffer et al. neuPrint: An open access tool for EM connectomics. *Frontiers in Neuroinformatics*, 16, 2022. doi: 10.3389/fninf.2022.896292 13

[38] L. K. Scheffer, C. S. Xu, M. Januszewski, Z. Lu, S.-y. Takemura, K. J. Hayworth et al. A connectome and analysis of the adult Drosophila central brain. *eLife*, 9:e57443, 2020. doi: 10.7554/eLife.57443 5, 13

[39] R. Sicat, J. Krüger, T. Möller, and M. Hadwiger. Sparse PDF volumes for consistent multi-resolution volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2417–2426, 2014. doi: 10.1109/TVCG.2014.2346324 1

[40] W. Silversmith, A. Zlateski, J. A. Bae, I. Tartavull, N. Kemnitz, J. Wu et al. Igneous: Distributed dense 3D segmentation meshing, neuron skeletonization, and hierarchical downsampling. *Frontiers in Neural Circuits*, 16, 2022. doi: 10.3389/fncir.2022.977700 13

[41] C. Talegaonkar, Y. Belhe, R. Ramamoorthi, and N. Antipa. Volumetrically consistent 3D Gaussian rasterization. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10953–10963, 2025. doi: 10.48550/arXiv.2412.03378 2, 3

[42] K. Tang, K. Ai, J. Han, and C. Wang. TexGS-VolVis: Expressive scene editing for volume visualization via textured Gaussian splatting. *IEEE Transactions on Visualization and Computer Graphics*, 32(1):933–943, 2026. doi: 10.1109/TVCG.2025.3634643 1, 2

[43] K. Tang and C. Wang. STSR-INR: Spatiotemporal super-resolution for multivariate time-varying volumetric data via implicit neural representation. *Computers & Graphics*, 119:103874, 2024. doi: 10.1016/j.cag.2024.01.001 2

[44] K. Tang, S. Yao, and C. Wang. iVR-GS: Inverse volume rendering for explorable visualization via editable 3D Gaussian splatting. *IEEE Transactions on Visualization and Computer Graphics*, 31(6):3783–3795, 2025. doi: 10.1109/TVCG.2025.3567121 1, 2, 13

[45] Y.-H. Tseng and J. H. Ferziger. A ghost-cell immersed boundary method for flow in complex geometry. *Journal of Computational Physics*, 192(2):593–623, 2003. doi: 10.1016/j.jcp.2003.07.024 4

[46] F. Vega-Higuera, P. Hastreiter, R. Fahlbusch, and G. Greiner. High performance volume splatting for visualization of neurovascular data. In *Proceedings of IEEE Visualization Conference*, pp. 271–278, 2005. doi: 10.1109/VISUAL.2005.1532805 1, 2

[47] I. Wald, S. Zellmann, N. Morrical, M. Larsen, and F. Sadlo. Faster rtx-accelerated empty space skipping using triangulated active region boundary geometry. In *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization*, pp. 37–44, 2021. doi: 10.2312/pgv.20211042 1, 6

[48] S. Weiss, P. Hermüller, and R. Westermann. Fast neural representations for direct volume rendering. *Computer Graphics Forum*, 41(6):196–211, 2022. doi: 10.1111/cgf.14578 1, 2, 13, 14

[49] Q. Wu, D. Bauer, M. J. Doyle, and K.-L. Ma. Interactive volume visualization via multi-resolution hash encoding based neural representation. *IEEE Transactions on Visualization and Computer Graphics*, 30(8):5404–5418, 2024. doi: 10.1109/TVCG.2023.3293121 1, 2, 13

[50] S. W. Wurster, H. Guo, H.-W. Shen, T. Peterka, and J. Xu. Deep hierarchical super resolution for scientific data. *IEEE Transactions on Visualization and Computer Graphics*, 29(12):5483–5495, 2023. doi: 10.1109/TVCG.2022.3214420 1, 2

[51] S. W. Wurster and H.-W. Shen. AMGSRN++: Improved adaptive SRN for scientific visualization. In *Proceedings of IEEE Pacific Visualization Symposium*, pp. 182–191, 2025. doi: 10.1109/PacificVis64226.2025.00024 2, 13

[52] S. W. Wurster, T. Xiong, H.-W. Shen, H. Guo, and T. Peterka. Adaptively placed multi-grid scene representation networks for large-scale data visualization. *IEEE Transactions on Visualization and Computer Graphics*, 30(1):965–974, 2024. doi: 10.1109/TVCG.2023.3327194 1, 2, 13

[53] S. Yao and C. Wang. VolSegGS: Segmentation and tracking in dynamic volumetric scenes via deformable 3D Gaussians. *IEEE Transactions on Visualization and Computer Graphics*, 32(1):407–417, 2026. doi: 10.1109/TVCG.2025.3642516 2

[54] S. Zellmann, M. Hellmann, and U. Lang. A linear time bvh construction algorithm for sparse volumes. In *Proceedings of IEEE Pacific Visualization Symposium*, pp. 222–226, 2019. doi: 10.1109/PacificVis.2019.00033 1

[55] S. Zellmann, M. Jaros, J. Amstutz, and I. Wald. GPU volume rendering with hierarchical compression using VDB. *arXiv preprint arXiv:2504.04564*, 2025. doi: 10.48550/arXiv.2504.04564 15

[56] S. Zellmann, D. Meurer, and U. Lang. Hybrid grids for sparse volume rendering. In *Proceedings of IEEE Visualization Conference*, pp. 1–5, 2019. doi: 10.1109/VISUAL.2019.8933631 1

[57] S. Zellmann, J. P. Schulze, and U. Lang. Binned k-d tree construction for sparse volume data on multi-core and GPU systems. *IEEE Transactions on Visualization and Computer Graphics*, 27(3):1904–1915, 2021. doi: 10.1109/TVCG.2019.2938957 1

[58] R. Zha, T. J. Lin, Y. Cai, J. Cao, Y. Zhang, and H. Li. R^2-Gaussian: rectifying radiative Gaussian splatting for tomographic reconstruction. In *Proceedings of Advances in Neural Information Processing Systems*, pp. 44907–44934, 2024. doi: 10.48550/arXiv.2405.20693 6, 14

[59] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 586–595, 2018. doi: 10.1109/CVPR.2018.00068 6

[60] Z. Zhang, T. Song, Y. Lee, L. Yang, C. Peng, R. Chellappa et al. LP-3DGS: Learning to prune 3D Gaussian splatting. *Advances in Neural Information Processing Systems*, 37:122434–122457, 2024. doi: 10.48550/arXiv.2405.18784 2

[61] M. Zwicker, H. Pfister, J. Van Baar, and M. Gross. EWA volume splatting. In *Proceedings of IEEE Visualization Conference*, pp. 29–538, 2001. doi: 10.1109/VISUAL.2001.964490 2

A METHOD SUPPLEMENTARY

Algorithm 1 Structure-aware primitive learning

V: GT volume
E: ellipsoid primitives with (μ, Σ, I)
d, p: densification and pruning iterations
β: reconstruction loss weight
One epoch: full traversal of subvolumes

$i \leftarrow 0$ ▷ Iteration count
while not converged **do**
  $V_{\text{gt}} \leftarrow \text{NextSubvolume}(V)$ ▷ Sequential GT subvolume
  $E' \leftarrow \text{GetOverlappingEllipsoids}(E, V_{\text{gt}})$ ▷ Subvolume ellipsoids
  $\hat{V} \leftarrow \text{Voxelize}(E')$ ▷ Predicted subvolume
  $L_{\text{rec}} \leftarrow \beta L_1(\hat{V}, V_{\text{gt}}) + (1-\beta) L_{\text{D-SSIM}}(\hat{V}, V_{\text{gt}})$ ▷ Subvolume loss
  $E \leftarrow \text{Adam}(\nabla L_{\text{rec}})$ ▷ Update (μ, Σ, I)
  if $i \in d$ **then**
    Densify(E) ▷ Clone / split (coverage)
  end if
  if $i \in p$ **then**
    for all ellipsoids $E_j \in E$ **do**
      $\Omega_j \leftarrow \text{GetCoveredVoxels}(E_j, V_{\text{gt}})$
      $\text{FA}_j \leftarrow \text{ComputeFA}(\Sigma_j)$
      $\text{score}_j \leftarrow |\Omega_j| \cdot I_j \cdot \text{FA}_j$ ▷ Importance score
    end for
    RemoveLowScore(E) ▷ Prune primitives
  end if
  $i \leftarrow i + 1$
  if EndOfTraversal() **then**
    ResetSubvolumeTraversal() ▷ One epoch completed
  end if
end while

Algorithm 2 GPU-acceleration through intra-ray parallelism

w, h: width and height of the image to rasterize
b: number of depth bins
E: Ellipsoid primitives with world-space parameters
(E^{μ}: mean, E^{Σ}: covariance)
TF: transfer function to map intensity to color and opacity
View: view configuration of current camera
I: rendered image

function RENDER($w, h, b, E, TF, View$)
  $T \leftarrow \text{CreateTiles}(w, h, b)$ ▷ Tiles include depth-wise subdivisions
  $E_{visible} \leftarrow \text{CullEllipsoid}(E, View)$
  $L, K \leftarrow \text{DuplicateWithKeys3D}(E_{visible}, T)$
  SortByTileKeys(K, L) ▷ No intra-tile depth sorting
  $R \leftarrow \text{IdentifyTileRanges}(K)$ ▷ range of primitives per 3D tile
  Initialize $I_{\text{bin}} \leftarrow 0$ ▷ Per-bin canvas
  for all Tiles t in T **do**
    $E_t \leftarrow L[R[t]_{\text{start}} : R[t]_{\text{end}}]$ ▷ Primitives assigned to tile t
    for all Pixels i in t **do**
      $\text{ray}[i, t_z] \leftarrow \text{IntensitySample}(i, E_t^{\mu}, E_t^{\Sigma}, TF)$
      $I_{\text{bin}}[i, t_z] \leftarrow \text{BlendPartialRay}(\text{ray}[i, t_z])$
    end for
  end for
  Flatten $T \leftarrow \text{CreateTiles}(w, h, 1)$ ▷ Reset $tile_grid.z = 1$
  for all Pixels i in I **do**
    $I[i] \leftarrow \text{MergeZBins}(I_{\text{bin}}[i, 0{:}b])$ ▷ Blend across b bins
  end for
  return I
end function

A.1 Structure-aware Primitive Learning

Appendix Algorithm 1 summarizes the full optimization loop of our structure-aware primitive learning, including 3D intensity field fitting, adaptive densification, and structure-aware pruning.

A.2 GPU Intra-Ray Parallelism Pseudo Code

Appendix Algorithm 2 describes our GPU-acceleration scheme based on intra-ray parallelism, which partitions each ray into depth bins and enables efficient parallel accumulation of primitive contributions during rendering.

B METHOD FIGURES

B.1 Differentiable Ellipsoid Primitive

Appendix Figure 1 compares the 1D spatial support of a Gaussian kernel and our sigmoid-based ellipsoidal primitive. As shown in Appendix Figure 1 (a), the unnormalized primitive $P(x)$ exhibits reduced peak intensity as the sharpness parameter k decreases, deviating from the desired unit response at the center. To address this, we normalize the primitive $\tilde{P}(x)$, as shown in Appendix Figure 1 (b), ensuring that the peak intensity remains constant regardless of k. In addition, the spatial support varies with k, and the cutoff region is adjusted accordingly to avoid truncation artifacts.

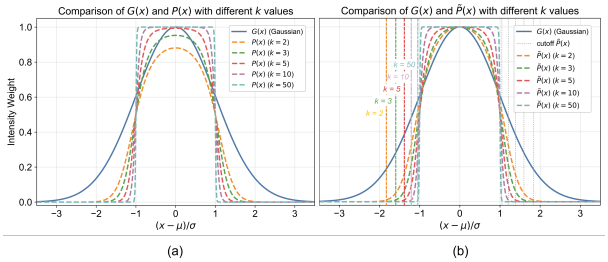


Fig. 1: 1D comparison between the Gaussian kernel $G(x)$ and our sigmoid-based ellipsoid primitives with unit center intensity. (a) Effect of the sharpness parameter k on the primitive $P(x)$. Peak intensity decreases as k becomes smaller than 5. (b) Normalized primitive $\tilde{P}(x)$ with corresponding cutoff regions for $k = 2, 3, 5, 10, 50$.

B.2 ESVR Interactive Viewer

Appendix Figure 2 presents the proposed viewer for flexible TF editing and interactive volume visualization. A demonstration of user interaction is provided in the supplemental video.

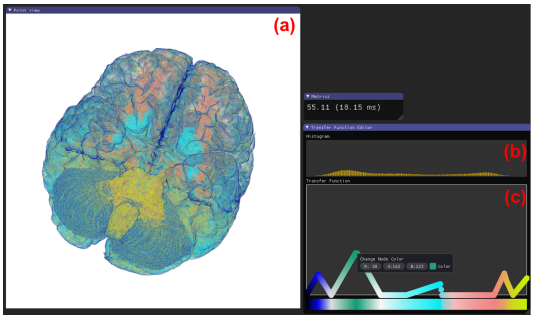


Fig. 2: TF editing interface of the ESVR viewer. (a) Rendering result of the extended CUDA-based rasterizer. (b) Histogram of data intensities. (c) TF editor for mapping intensities to color and opacity.

C EXPERIMENT CONFIGURATION

Table 1: Training hyperparameters of ESVR

Dataset	n_{chunk}	res_{chunk}	res_{subvol}	e_{total}	e_{densify}	e_{prune}	$thres_{\text{grad}}$	per_{dense}
Aneurysm	-	-	128^3	2800	16/400	1200	2.5e−9	0.008
Breast	-	-	128^3	2800	16/400	1200	2.5e−9	0.008
Neuron	-	-	133^3	2800	16/400	1200	6.5e−9	0.001
BigBrain	8	(875,822,714)	440^3	1600	16/400	200/300/800	2.5e−9	0.008
Woodbranch	8	(1024,1024,1024)	512^3	1600	16/400	800	2.5e−9	0.01
Hemibrain(vDeltaA)	64	(1093,1018,1043)	550^3	800	16/200	400	1.0e−10	0.01
Hemibrain(lLN2P)	125	(1226,955,1023)	614^3	800	16/200	400	1.0e−10	0.01

Table 2: Training hyperparameters for baseline methods. For large datasets, we report a per-chunk network configuration.

		fV-SRN		NGP				AMGSRN++			InstantVNR		
Dataset	**Config**	n_{feat}	res_{grid}	n_{lev}	$\log_2 h$	res_{base}	res_{max}	n_{feat}	n_{grid}	res_{grid}	n_{lev}	$\log_2 h$	res_{base}
Aneurysm	Size-M	2	(72,72,72)	14	15	16	512	2	10	(32,32,32)	12	15	16
	PSNR-M	8	(128,128,128)	17	20	16	256	4	16	(64,64,64)	19	19	16
Breast	Size-M	2	(72,72,72)	14	15	16	512	2	12	(32,32,32)	13	15	16
	PSNR-M	8	(256,256,256)	17	24	16	256	4	16	(128,128,128)	16	18	16
Neuron	Size-M	2	(75,75,75)	14	15	16	512	2	12	(32,32,32)	9	15	16
	PSNR-M	4	(64,64,64)	17	15	16	512	2	12	(101,52,6)	16	17	16
BigBrain	Size-M	2	(49,49,49)	16	13	16	512	2	14	(22,22,22)	12	16	16
	PSNR-M	2	(50,50,50)	10	13	16	512	2	12	(20,20,20)	14	17	16
Woodbranch	Size-M	4	(32,32,32)	17	12	16	512	2	16	(16,16,16)	10	15	16
	PSNR-M	4	(30,30,30)	15	12	16	512	2	12	(16,16,16)	14	17	16
Hemibrain (vDeltaA)	Size-M	2	(16,16,16)	9	10	8	512	1	10	(10,10,10)	-	-	-
	PSNR-M	4	(24,24,24)	10	10	8	512	1	16	(10,10,10)	-	-	-
Hemibrain (lLN2P)	Size-M	2	(20,20,20)	9	10	8	512	1	16	(10,10,10)	14	16	16
	PSNR-M	4	(24,24,24)	8	10	8	256	1	16	(10,10,10)	14	16	16

C.1 Dataset Details

The hemibrain dataset [38] is a synapse-resolution connectome of the adult Drosophila melanogaster brain, containing $\sim$ 25,000 neurons and millions of synaptic connections. Volumes corresponding to specific annotated neuron types (e.g., vDeltaA and lLN2P) are extracted using neuPrint [37] for querying neuron metadata and CloudVolume [40] for retrieving volumetric data. Due to the large size of full-resolution volumes, we use downsampled representations at MIP level 2 ($\times$8 downsampling) or 3 ($\times$64 downsampling) for analysis. The BigBrain dataset [3] is a 3D reconstruction of a complete human brain at 20μm isotropic resolution. Similar to the hemibrain dataset, we use downsampled volumes at MIP level 2. In addition, we utilize a Neuron dataset, a proprietary dataset provided by collaborators.

C.2 Training Implementation Details

Appendix Table 1 summarizes the hyperparameters used for training, including number of chunks (n_{chunk}), chunk resolution (res_{chunk}), subvolume resolution (res_{subvol}), total epochs (e_{total}), densification start/end epoch (e_{densify}), structure-aware pruning epoch (e_{prune}), gradient threshold ($thres_{\text{grad}}$) and percent-dense parameter (per_{dense}) for densification. Note that per-chunk parameters are denoted for large volumes.

Chunk-based training. Chunk-based training is applied to large-scale datasets with different numbers of chunks: using 8 chunks for BigBrain and Woodbranch, 64 for Hemibrain (vDeltaA), and 125 for Hemibrain (lLN2P). For all datasets, the full volume is optimized in a normalized $[-1,1]^3$ coordinate space. For small and medium volumes, ellipsoids are trained across the entire $[-1,1]^3$ space, while the training space for large volumes is scaled accordingly (e.g., $[-0.5,0.5]^3$ for 8 chunks, $[-0.25,0.25]^3$ for 64 chunks, and $[-0.2,0.2]^3$ for 125 chunks). Regarding hardware, all training is performed on NVIDIA RTX A6000 GPUs. BigBrain and Woodbranch are trained sequentially on a single A6000, while the two large Hemibrain datasets use the parallel chunk-based mode across multiple GPUs: 4 GPUs for Hemibrain (vDeltaA) and 5 GPUs for Hemibrain (lLN2P). All small and medium datasets are trained on a single A6000.

Subvolume configuration. During optimization, the training unit (either the full volume or an individual chunk) is partitioned into voxelized cubic subvolumes. The subvolume side length is set to approximately half the longest axis of the training unit. For flat volumes such as Neuron, the smallest axis is used instead.

Training schedule. Optimization begins with a 16 epoch warm-up phase without structural updates, followed by densification every 4 epochs. Along with 3D intensity field fitting, densification is performed until 14–25% of the training process, while structure-aware importance score pruning is applied at 42–50%. The total number of training epochs is scaled with volume size (2800 / 1600 / 800 for small–medium / large / extremely large volumes) to balance training time and quality. All datasets execute single pruning iteration except for BigBrain, which is treated as an exception with a multi-stage pruning strategy. The BigBrain dataset contains heterogeneous regions, including relatively smooth regions (cortical) and highly detailed regions (cerebellar), which require more aggressive densification. For the cerebellar chunks, pruning is performed three times: twice during the densification phase at epochs 200 and 300 (pruning ratio 0.8), and once after densification at epoch 800 (ratio 0.85).

Densification and gradient threshold. Densification follows the original 3DGS principle: primitives with large gradients indicate regions requiring more representational capacity and are selected for cloning or splitting. This gradient-based selection ensures efficient allocation of primitives to regions with high reconstruction error. The densification process is controlled by two key parameters. One is the gradient threshold ($thres_{\text{grad}}$), determining whether a primitive should be densified based on the magnitude of its gradient, while the other is the percent-dense parameter (per_{dense}), determining whether to split or clone the primitive based on the spatial extent. Each parameter is tuned depending on the dataset and volume resolution: the gradient threshold (1×10^{-10} to 6.5×10^{-9}) and the percent-dense parameter (0.001 to 0.01). These parameters jointly control the aggressiveness of densification.

C.3 Baseline Configuration

Appendix Table 2 summarizes the hyperparameters used for baseline SRN methods. All parameters for each method are adjusted to match the target model size or PSNR. Parameters not listed in the table follow the default settings of each method.

(1) fV-SRN [48] employs latent grids as its primary representation, with the number of features per grid (n_{feat}) and grid resolution (res_{grid}) being adjustable.

(2) NGP [30] leverages multi-resolution hash encoding for fast training. The key parameters that determine the hash table capacity are the number of levels (n_{lev}), hash table size ($\log_2 h$), and base/max grid resolution ($res_{\text{base}}, res_{\text{max}}$).

(3) AMGSRN++ [51] is based on the prior work [52] which introduces an adaptive multi-grid representation for scientific visualization. AMGSRN++ further enhances this framework by incorporating custom CUDA kernels, feature grid compression, and extensions to the temporal domain. Encoding is mainly configured using three parameters: the number of features per grid (n_{feat}), the number of grids (n_{grid}), and the grid resolution (res_{grid}).

(4) InstantVNR [49] is a volume rendering framework based on NGP [30], extended with sample streaming, macro-cell optimization, and out-of-core sampling techniques. Although it adopts the same encoding scheme as NGP, the network parameters are not exactly identical, as the macro-cell structure also contributes to the overall model size when matching the target model size or PSNR.

C.4 iVR-GS Training Details

iVR-GS [44] composes multiple base models where each model is trained under one basic TF. For each basic scene, Gaussians are opti-

mized under a view-dependent appearance of the rendered scene using a Blinn-Phong shading model. In this formulation, each Gaussian's color is decomposed into intrinsic color and lighting components, enabling post-training editing of both appearance and illumination. This design differs from our experimental setup, which assumes a lighting-free (emission-based) volumetric representation. To ensure identical rendering settings with our method, the training images for iVR-GS are generated from DVR images with lighting disabled. Otherwise, the inclusion of shading effects in the input images would lead to appearance differences. We train two basic models for Aneurysm dataset using 180 multi-view images and evaluate them on 180 novel views, following the icosphere sampling strategy of the original paper. For visualization, the iVR-GS result presented in Figure 10 in the main text is rendered using its default viewer. Although we disabled lighting in the training images, minor brightness variations may still persist due to the view-dependent nature of the iVR-GS training formulation. However, as our primary goal is to evaluate the information loss of image-based methods, we focus the comparison on structural differences rather than appearance variations.

D Additional Experiments

D.1 Comparison with Image-based Baseline Methods

R^2-Gaussian. Appendix Table 3 presents a quantitative comparison with the baseline, R^2-Gaussian [58]. R^2-Gaussian reconstructs volumes from X-ray projections. To ensure a fair comparison, we synthesize X-ray projections from the GT volume using TIGRE[1] and provide both the synthetic projections and the GT volume to R^2-Gaussian during reconstruction. As R^2-Gaussian is trained primarily from X-ray projections while using volumetric data only as a regularization signal, its reconstructed volumes exhibit significantly lower fidelity in 3D space, as reflected by substantially lower 3D PSNR values across all datasets (e.g., 22.92 vs. 51.54 on the Aneurysm dataset). In contrast, R^2-Gaussian achieves relatively higher 2D PSNR, since it is optimized to match X-ray projections. However, this metric reflects projection consistency rather than volumetric reconstruction quality, and thus cannot be directly applied to volume visualization tasks. Finally, due to its image-based training paradigm, R^2-Gaussian demonstrates notably faster training times compared to our method. Due to the lack of support for large raw volumetric data in R^2-Gaussian, comparisons are limited to datasets up to the Neuron scale.

Table 3: Quantitative comparison with R^2-Gaussian. Training time is measured from initialization to completion. Values in parentheses denote the number of primitives for each method.

Dataset	Ours	R^2-Gaussian		
	3D PSNR	3D PSNR	2D PSNR	Training Time (s)
Aneurysm (60.4k)	51.54 (59k)	22.92	53.27	553
Breast (69.2k)	46.00 (67k)	21.01	54.27	616
Neuron (65.3k)	39.56 (65k)	31.86	47.78	1063

LightGaussian. Appendix Table 4 summarizes the pruning performance comparison with LightGaussian [9]. Across three datasets of varying scale, our method achieves higher PSNR and SSIM while using a comparable or smaller number of primitives. LightGaussian scores primitives based on accumulated ray contributions from multiple synthesized views in a 2D projection-based framework, using view-dependent features such as ray hits and transmittance. Following this formulation, we generate 360 uniformly sampled views on a sphere for our experimental setting. When applied to our volume setting, this projection-based scoring yields lower accuracy (for example, 48.76 dB on Aneurysm and 38.34 dB on Neuron), compared to our volume-based scoring (50.76 dB and 39.44 dB). In contrast, our method computes importance directly in volumetric space by integrating voxel intensities covered by each primitive and weighting them with fractional anisotropy, leading to more reliable ranking of primitives.

[1] A. Biguri et al., Tigre: a matlab-gpu toolbox for cbct image reconstruction. *Biomedical Physics & Engineering Express*, 2(5):055010, 2016.

Table 4: Pruning performance comparison with LightGaussian. Metrics report 3D PSNR (dB) / SSIM / number of primitives. For Woodbranch we use one of the chunks for evaluation.

Method	Aneurysm	Neuron	Woodbranch
LightGaussian	48.76 / 0.9968 / 32k	38.34 / 0.9206 / 77k	41.19 / 0.9147 / 12k
Ours	**50.76** / 0.9976 / 32k	**39.44** / 0.9267 / 77k	**41.04** / 0.8808 / 12k

D.2 Comparison with Traditional Compression Methods

Though data reduction is not the primary objective of our method, we believe it is valuable to compare the compression capability of ESVR with traditional compression techniques. We emphasize that our method is not a compressor, but a volumetric representation and rendering framework. We compare against three traditional compression methods, TTHRESH [4], ZFP [25], and SZ3 [23]. We do not include rendering-based compressors or bricked variants of existing methods, as prior work [48] has shown that such approaches typically result in slower decoding, lower compression ratios, and higher memory usage. Our evaluation metrics include compressed size, 3D PSNR, compression time, and decompression time. Notably, for our method, compression time represents the full training time, and decompression time is the time required to query the entire volume via 3D rasterization (i.e., voxelization). All compression methods are evaluated on a system with dual Intel Xeon Gold 6226R CPUs (32 cores, 64 threads), 1.1 TB RAM, and an NVIDIA RTX A6000 GPU. We use CPU-based implementations for all traditional compressors to ensure a consistent evaluation environment. Although ZFP provides GPU-accelerated variants, they are strictly limited to fixed-rate mode and do not support the flexible error-bounded configurations required for our study.

Compression Results. Appendix Table 5 shows the quantitative results under PSNR-matched and Size-matched settings, where compression methods are configured to match either the PSNR levels or memory size reported in Table 2 of main text. The relative performance depends strongly on dataset scale. Under the 3D PSNR-matched setting, traditional compressors achieve substantially smaller storage sizes and nearly lossless reconstruction for Aneurysm. However, as the dataset size increases, ESVR increasingly outperforms ZFP and SZ3 in compression ratio. TTHRESH remains highly effective on Neuron and Woodbranch, achieving even smaller model sizes than ESVR, but fails earlier with out-of-memory errors on Hemibrain due to its higher memory demand during compression. The Size-matched setting shows a similar trend. On small datasets, traditional compressors can maintain high quality and sometimes undershoot the target size. On larger datasets, however, matching ESVR's compact size requires aggressive voxel-level compression for ZFP and SZ3, leading to lower PSNR. TTHRESH, in contrast, achieves higher PSNR than ESVR on Neuron and Woodbranch under the same size budget, while encountering the same OOM failure on Hemibrain.

Table 5: Comparison with traditional compression methods under 3D PSNR-matched and Size-matched settings.

Dataset	Metric	Ours	3D PSNR-matched			Size-matched		
			TTHRESH	ZFP	SZ3	TTHRESH	ZFP	SZ3
Aneurysm	Size (MB)	2.600	3.318	0.301	0.154	2.611	1.805	0.320
	3D PSNR (dB)	51.54	51.56	54.42	51.28	47.74	183.60	∞
	Comp. (s)	388	23.7459	0.1457	0.5319	26.27	0.15	0.07
	Decomp. (s)	0.0079	1.7490	0.1473	0.1343	1.41	0.08	0.06
Neuron	Size (MB)	2.800	1.175	11.567	2.060	2.866	3.417	2.795
	3D PSNR (dB)	39.42	39.49	40.15	39.57	40.44	34.17	40.41
	Comp. (s)	1320	159.9387	4.0231	17.1053	330.33	2.49	11.96
	Decomp. (s)	0.0482	25.6487	3.4315	3.2295	38.87	2.32	5.55
Woodbranch	Size (MB)	4.300	0.188	239.558	31.090	4.170	16.777	4.355
	3D PSNR (dB)	40.60	40.69	40.94	40.60	42.18	28.41	39.35
	Comp. (s)	5676	3406.1630	149.1826	355.3101	6779.67	37.68	232.69
	Decomp. (s)	1.0931	587.1047	109.1363	106.1914	760.76	36.62	106.58
Hemibrain (vDeltaA)	Size (MB)	4.200	OOM	147.590	5.169	OOM	145.065	4.194
	3D PSNR (dB)	35.34	OOM	35.86	35.57	OOM	30.06	32.68
	Comp. (s)	2998	OOM	240.4151	4493.3396	OOM	308.16	4095.52
	Decomp. (s)	5.1207	OOM	285.3800	1795.7484	OOM	1964.00	2505.96

Qualitative Comparison. Although 3D PSNR captures the overall reconstruction trend, it can overestimate visual fidelity for sparse volumes because most voxels correspond to background or empty space. Therefore, rendering quality is also an important evaluation criterion

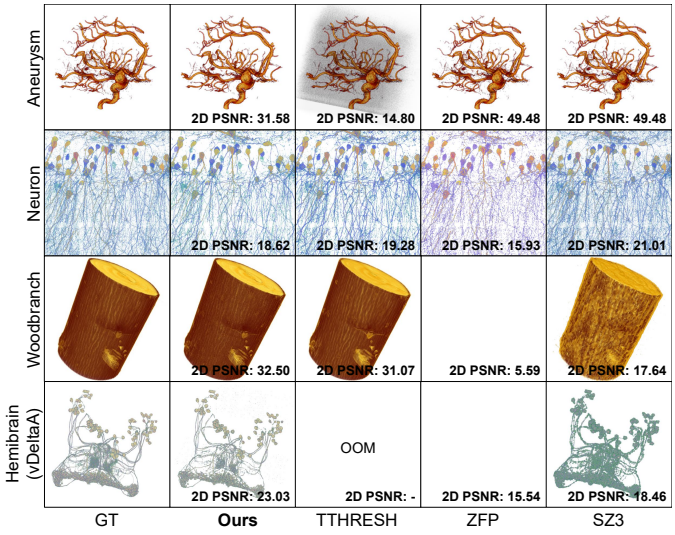


Fig. 3: Qualitative results of traditional compression methods. Woodbranch and Hemibrain are downsampled (×8 and ×64) to enable DVR (GT, TTHRESH, ZFP, SZ3) rendering without OOM.

Table 6: Time comparison for compression/decompression and single-frame rendering. For ours, compression time indicates sequential training time on a single GPU. Rendering time is reported as seconds per frame. Total time is the sum of the above, denoting the time required to render a single frame.

Dataset	Metric	Ours	TTHRESH	ZFP	SZ3
Aneurysm	Comp. (s)	388	23.75	0.15	0.53
	Decomp. (s)	-	1.75	0.15	0.13
	Render (s)	0.0045	0.0073	0.0073	0.0073
	Total (s)	388.00	25.51	0.31	0.67
Breast	Comp. (s)	756	11.08	0.23	1.23
	Decomp. (s)	-	2.55	0.23	0.25
	Render (s)	0.0047	0.0047	0.0047	0.0047
	Total (s)	756.00	13.63	0.46	1.48
Neuron	Comp. (s)	1320	159.94	4.02	17.11
	Decomp. (s)	-	25.65	3.43	3.23
	Render (s)	0.0076	0.0110	0.0110	0.0110
	Total (s)	1320.01	185.60	7.46	20.35
BigBrain	Comp. (s)	32320	4233.14	44.57	134.62
	Decomp. (s)	0.6731	714.90	38.98	36.46
	Render (s)	0.0206	0.0693	0.0693	0.0693
	Total (s)	32320.02	4948.11	83.62	171.15
Woodbranch	Comp. (s)	45408	3406.16	149.18	355.31
	Decomp. (s)	-	587.10	109.14	106.19
	Render (s)	0.0230	OOM	OOM	OOM
	Total (s)	45408.02	-	-	-
Hemibrain (vDeltaA)	Comp. (s)	191872	OOM	240.42	4493.34
	Decomp. (s)	-	OOM	285.38	1795.75
	Render (s)	0.0072	OOM	OOM	OOM
	Total (s)	191872.01	-	-	-
Hemibrain (lLN2P)	Comp. (s)	440250	OOM	501.98	OOM
	Decomp. (s)	-	OOM	541.65	OOM
	Render (s)	0.0115	OOM	OOM	OOM
	Total (s)	440250.01	-	-	-

for sparse volumetric data. As shown in Appendix Figure 3, ZFP reconstructs the small Aneurysm dataset nearly losslessly at a substantially smaller size, but its renderings become visibly blocky from Neuron onward and fail to preserve foreground structures on Woodbranch and Hemibrain. This discrepancy is more clearly reflected by 2D PSNR. For example, ZFP achieves only 15.93, 5.59, and 15.54 dB on Neuron, Woodbranch, and Hemibrain, respectively, compared with 18.62, 32.50, and 23.03 dB for our method. SZ3 remains competitive on Neuron, but its rendered results on large datasets deviate visibly from GT, consistent with its substantially lower 2D PSNR on Woodbranch and Hemibrain. TTHRESH achieves slightly higher 2D PSNR on Neuron, whereas our method is better on Woodbranch. On Hemibrain, TTHRESH runs out of memory. Taken together, these results highlight a broader limitation of regular-grid compression for sparse volumetric data. Under aggressive memory budgets, regular-grid compression tends to smooth, quantize, or remove sparse foreground structures. In contrast, our adaptive representation allocates its capacity according to the spatial distribution and complexity of the data, allowing it to preserve sparse and visually salient structures more effectively under the same storage budget. This advantage is particularly meaningful for highly sparse and spatially nonuniform volumes.

Time Comparison. To further analyze runtime cost, Appendix Table 6 reports compression (training) time, decompression time, single-frame rendering time, and their total. The main cost of ESVR is incurred offline during optimization-based training, which can take up to several days when the largest volumes are trained sequentially on a single GPU. Once trained, however, ESVR requires no separate decompression stage: primitives are directly rasterized at query time, enabling millisecond-level rendering with low VRAM usage even for extremely large volumes. Traditional compressors encode much faster, but require full-volume decompression before rendering, after which the decompressed volume is visualized using DVR. Decompression time also grows significantly with data size, reaching tens to hundreds of seconds for datasets from Neuron onward. Furthermore, DVR runs out of memory on Woodbranch and Hemibrain. Therefore, ESVR is not intended as a fast one-time encoder, but instead offers a different trade-off: a higher offline preprocessing cost in exchange for a compact, directly renderable representation that supports interactive visualization in memory-limited settings where full-volume decompression or DVR becomes impractical.

D.3 Comparison with VDB-based Methods

To better position ESVR with respect to sparse volumetric representations, we compare our method with OpenVDB [31] and NanoVDB [32] in terms of compression efficiency, reconstruction quality, memory usage, and rendering performance. OpenVDB is an open-source implementation of the VDB sparse volume data structure [31]. In this section, we use the term VDB-based representation to refer to sparse hierarchical grid representations following this data structure. OpenVDB denotes the CPU-oriented implementation used for conversion and storage, while NanoVDB converts the OpenVDB hierarchy into a compact GPU-friendly format optimized for traversal and rendering. We do not include NeuralVDB [19], which is a representative neural compression approach for VDB volumes, in our quantitative evaluation because there is currently no publicly available implementation suitable for reproducible evaluation. Furthermore, NeuralVDB primarily focuses on neural compression of VDB volumes, whereas our evaluation targets directly renderable representations for which model size, reconstruction quality, rendering performance, and memory consumption can be measured consistently.

Compression Results. Appendix Table 7 summarizes the results on both lossless OpenVDB/NanoVDB conversion and lossy fixed-rate VDB compression following Zellmann et al. [55]. In the lossless setting, OpenVDB and NanoVDB preserve the original voxel values exactly, resulting in infinite PSNR. However, their storage requirements remain substantially larger than those of ESVR. For example, the lossless NanoVDB representation requires 5.7 GB for BigBrain and 35.2 GB for Woodbranch, whereas ESVR represents the same datasets using only 7.6 MB and 4.3 MB, respectively. Although VDB structures efficiently remove empty regions, they remain fundamentally grid-based representations and therefore must preserve voxel information at the original grid resolution.

We further evaluate lossy VDB compression under a size-matched setting with ESVR. While lossy OpenVDB/NanoVDB can achieve a similar storage size, reconstruction quality degrades substantially. For example, on Breast, ESVR achieves 46.00 dB PSNR at 2.9 MB, whereas lossy NanoVDB achieves only 27.51 dB at 3.3 MB. Similar trends can be observed across all datasets. These results indicate that

Table 7: Compression results on OpenVDB and NanoVDB under lossless/lossy setting. Lossy VDB results are generated under a Size-matched setting with ESVR.

Dataset	Metric	Ours	Lossless		Lossy	
			OpenVDB	NanoVDB	OpenVDB	NanoVDB
Aneurysm	Size	2.6MB	0.7MB	6.7MB	0.7MB	2.59MB
	PSNR	51.54	∞	∞	∞	41.94
Breast	Size	2.9MB	2.8MB	17.6MB	2.8MB	3.3MB
	PSNR	46.00	∞	∞	∞	27.51
Neuron	Size	2.8MB	1.1GB	1.8GB	2.6MB	3.1MB
	PSNR	39.56	∞	∞	32.83	32.23
BigBrain	Size	7.6MB	1.8GB	5.7GB	7.8MB	7.4MB
	PSNR	21.53	∞	∞	7.54	7.52
Woodbranch	Size	4.3MB	18.8GB	35.2GB	4.8MB	4.3MB
	PSNR	40.60	∞	∞	28.43	28.41
Hemibrain	Size	4.2MB	0.5GB	2.2GB	4.1MB	4.2MB
(vDeltaA)	PSNR	35.34	∞	∞	30.11	30.06
Hemibrain	Size	5.8MB	1.0GB	4.5GB	6.0MB	6.4MB
(lLN2P)	PSNR	37.03	∞	∞	29.46	29.43

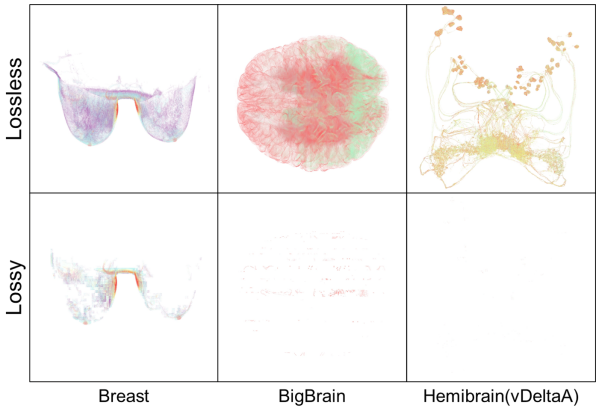


Fig. 4: Rendering results of volumes converted into lossless/lossy VDB structure.

matching the compact representation size of ESVR with a grid-based VDB representation requires aggressive voxel removal and quantization. The visual impact of this compression is shown in Appendix Figure 4. Although some datasets exhibit only moderate PSNR degradation numerically, the rendered results reveal severe structural loss. In particular, many thin and sparse structures disappear entirely, leaving large empty regions in the rendered volume. This behavior becomes increasingly pronounced for high-resolution datasets because VDB compression ultimately operates by deciding which voxel blocks should be retained or discarded. As volume resolution increases, preserving fine structures under a strict storage budget becomes progressively more difficult.

Table 8: Rendering performance and VRAM usage on lossless/lossy compression using NanoVDB.

Dataset	Metric	Ours	Lossless	Lossy
Aneurysm	FPS	223.19	204.55	239.28
	VRAM	1.64GB	1.56GB	1.55GB
Breast	FPS	213.49	145.75	257.77
	VRAM	1.64GB	1.57GB	1.55GB
Neuron	FPS	131.08	32.32	387.15
	VRAM	1.64GB	3.39GB	1.56GB
BigBrain	FPS	48.64	10.76	372.18
	VRAM	1.74GB	7.29GB	1.56GB
Woodbranch	FPS	43.42	OOM	425.28
	VRAM	1.71GB	OOM	1.56GB
Hemibrain	FPS	137.95	90.31	462.94
(vDeltaA)	VRAM	1.69GB	3.75GB	1.56GB
Hemibrain	FPS	86.76	69.15	469.26
(lLN2P)	VRAM	1.67GB	6.07GB	1.56GB

Rendering Performance. Appendix Table 8 reports NanoVDB rendering performance. For rendering, we implement a simple OptiX-based HDDA ray-marching renderer for NanoVDB and report FPS and peak VRAM usage. We omit OpenVDB rendering performance because its CPU-oriented traversal model is not directly comparable to our GPU-based renderer, and instead use NanoVDB as the relevant GPU-based baseline for interactive rendering. Lossless NanoVDB generally renders more slowly than ESVR while requiring substantially higher memory usage. In contrast, lossy NanoVDB often achieves higher FPS than ESVR. However, this speedup must be interpreted together with the quality degradation shown in Appendix Table 7 and Appendix Figure 4. Because aggressive compression removes a large fraction of the original voxel data, the renderer traverses significantly fewer active regions, resulting in higher rendering performance at the expense of structural fidelity.

These observations highlight a fundamental difference between the two representations. VDB-based methods remain resolution-dependent sparse grids whose memory and compression behavior are tied to the underlying voxel discretization. Consequently, their effectiveness decreases as volume resolution increases or as the volume becomes less sparse. This trend is visible in Woodbranch, where the denser occupancy leads to noticeably weaker compression compared to the highly sparse Hemibrain datasets. In contrast, ESVR is a primitive-based representation. Instead of preserving individual voxels, it approximates volumetric structures using a compact set of anisotropic ellipsoidal primitives. As a result, its storage complexity and rendering performance depend primarily on the number of primitives rather than the original grid resolution. This enables ESVR to maintain compact model sizes while preserving sparse anisotropic structures and supporting interactive rendering. Overall, ESVR provides a more balanced trade-off between storage size, reconstruction fidelity, memory consumption, and rendering performance for extremely large sparse volumetric datasets.

E Ablation

E.1 Primitive Type Comparison

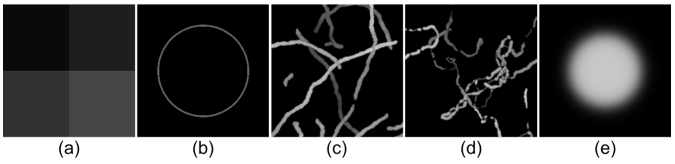


Fig. 5: 2D slice of synthetic datasets used for primitive type comparison. (a) 8area, (b) EmptySphere, (c) SynthTube, (d) SynthWiggle, (e) DiffuseScalar

To further evaluate the expressiveness of our differentiable ellipsoids, we construct five synthetic datasets, as shown in Appendix Figure 5, and compare them with Gaussian primitives in terms of FPS, PSNR, and average primitive hits per voxel. Appendix Figure 5 (a) shows a dense volume divided into eight chunks (8area), while Appendix Figure 5 (b) represents an empty sphere with only a surface (EmptySphere). Appendix Figures 5 (c) and (d) correspond to tube-like structures (SynthTube and SynthWiggle) and (e) sphere with diffuse boundaries (DiffuseScalar). As shown in Appendix Table 9, our method achieves lower PSNR than Gaussians in dense data. However, it consistently provides significantly higher FPS and fewer average primitive hits per voxel, demonstrating improved computational efficiency. In contrast, results on EmptySphere show that our differentiable ellipsoids better capture surface structures than Gaussians. Furthermore, on SynthTube and SynthWiggle, our method achieves both higher PSNR and faster rendering speed, indicating strong expressiveness for sparse and anisotropic structures. Notably, DiffuseScalar's smooth boundary blending with Gaussian tails achieves superior reconstruction quality compared to ellipsoid overlaps. While our ellipsoid representation demonstrates advantages in FPS and primitive hits as intended,

it exhibits fundamental limitations in expressiveness for representing smooth boundaries.

Table 9: Ablation study on primitive types (Ellipsoid vs. Gaussian) before and after pruning on synthetic datasets. Sparsity improvement is measured through average hits per voxel and primitive count reduction. Avg PrimHits denotes the average number of primitives per voxel, computed over voxels with non-zero primitive hits.

Dataset	Metric	Ellipsoid		Gaussian	
		Before	*After*	*Before*	*After*
8area	Size (MB)	4.18 (97k)	1.27 (29k)	4.25 (99k)	1.27 (29k)
	3D PSNR (dB)	41.73	42.67	45.86	47.93
	FPS	59.98	60.12	3.60	6.32
	Avg PrimHits	7.84	5.68	15.44	9.97
EmptySphere	Size (MB)	6.66 (155k)	2.26 (53k)	6.55 (152k)	2.23 (52k)
	3D PSNR (dB)	46.13	47.40	38.10	43.91
	FPS	167.33	291.26	9.25	18.90
	Avg PrimHits	9.79	3.53	26.47	13.50
SynthTube	Size (MB)	2.61 (61k)	0.89 (21k)	2.74 (64k)	0.93 (22k)
	3D PSNR (dB)	71.51	66.25	63.02	64.66
	FPS	183.78	268.57	19.08	50.26
	Avg PrimHits	14.99	9.29	34.69	19.26
SynthWiggle	Size (MB)	1.42 (33k)	0.48 (11k)	1.45 (34k)	0.49 (11k)
	3D PSNR (dB)	73.46	67.88	65.18	65.18
	FPS	226.76	329.07	31.79	88.39
	Avg PrimHits	12.56	7.31	24.89	12.28
DiffuseScalar	Size (MB)	4.54 (105K)	1.35 (29k)	4.35 (106k)	1.36 (31k)
	3D PSNR (dB)	48.90	48.18	52.28	53.84
	FPS	124.63	157.74	5.04	12.76
	Avg PrimHits	5.71	5.01	13.79	8.25

E.2 Pruning strategy

We additionally provide qualitative results in Appendix Figure 6 to support the pruning strategy. As shown in the figure, FA-based scoring better preserves fine vascular structures and high-frequency details, while intensity and scale cues help maintain dense regions. Ellipsoids with higher FA scores tend to cluster around structurally informative regions, demonstrating that FA effectively captures structural importance. These observations are consistent with the quantitative results shown in Table 8 in the main text, where combining FA with intensity and scale cues yields the best reconstruction quality.

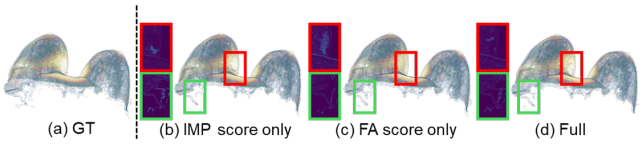


Fig. 6: Qualitative results highlighting the effect of the FA score on rendering quality of Breast dataset.

E.3 Learnable K

While we use k as a fixed hyperparameter in the main setting, we additionally evaluate a learnable k per ellipsoid, as well as a smoother configuration with $k = 3$ in Appendix Table 10. We observe that the effectiveness of learnable k depends on the dataset. In some cases (e.g., Aneurysm and BigBrain), a fixed k achieves higher PSNR than the learnable variant, as the learned values tend to saturate around $k \approx 5$. In contrast, for datasets with denser regions such as Woodbranch, the learnable k yields the highest PSNR. However, the PSNR improvement is marginal, while rendering performance degrades due to the need to compute adaptive cutoff regions for each primitive. This introduces a trade-off between PSNR and FPS depending on the dataset. Given the relatively small PSNR gains compared to the significant drop in FPS, we adopt a fixed k as a hyperparameter in our final design.

Table 10: Comparison of different sharpness (k) settings of differentiable ellipsoids. Each cell reports 3D PSNR (dB) / number of primitives / FPS. For BigBrain and Woodbranch, we use one of the chunks for evaluation.

Dataset	k=5	k=3	Learnable k
Aneurysm	**50.96** / 33.2k / **250.85**	48.73 / 32.7k / 196.10	49.76 / 33.8k / 182.12
BigBrain	**18.65** / 47.3k / **153.38**	18.54 / 42.6k / 142.71	18.52 / 43.8k / 132.51
Woodbranch	41.02 / 11.9k / **238.70**	41.30 / 12.7k / 235.18	**41.33** / 12.5k / 225.31

E.4 Boundary Sharpness

We evaluate the effect of boundary sharpness controlled by parameter k in Appendix Figure 7. Raw evaluation metrics (L_1 loss, DSSIM loss, and PSNR) are recorded every 100 iterations. Since the loss is computed on subvolumes, directly plotting the raw loss and PSNR values does not accurately reflect convergence and may appear to fluctuate. To improve visual clarity, a moving average filter is applied to the raw logs. Specifically, a window size of $n = 10$ is used for Appendix Figure 7. As shown in Appendix Figure 7, soft boundaries $k = 5$ yield stable convergence and high PSNR, while increasing k progressively slows and destabilizes optimization as gradients become increasingly localized near sharp boundaries. At $k = 5000$, PSNR saturates immediately and fails to improve, confirming that hard boundaries suppress the gradient signal needed for effective parameter updates. This behavior contrasts with EVER, where hard ellipsoids remain trainable through analytic transmittance integration. In our volumetric setting, near-zero spatial gradients outside the ellipsoid footprint make hard boundaries unsuitable.

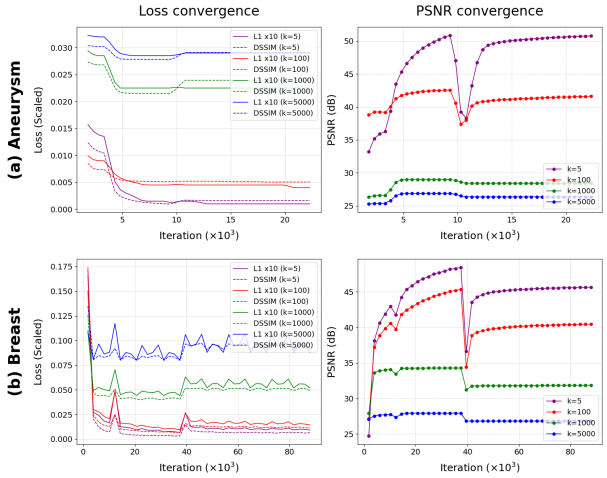


Fig. 7: Ablation on boundary sharpness (k) of differentiable ellipsoids on Aneurysm and Breast datasets.

E.5 Effect of Depth Bins

We investigate the impact of the number of depth bins used in the per-primitive ray sampling renderer. Appendix Figure 8 reports rendering performance (FPS), GPU memory usage, and rendering quality for 1, 4, 8, and 16 depth bins. Using multiple depth bins generally improves rendering performance by increasing intra-ray parallelism. However, the benefit saturates when the number of bins increases too much. Across all tested datasets, 4 and 8 bins achieve the highest rendering performance, whereas 16 bins consistently reduces FPS and can even perform worse than the single-bin configuration due to the additional overhead of maintaining and compositing per-bin intermediate buffers. GPU memory consumption exhibits a similar trend. The 4-bin configuration requires the least VRAM across all datasets, while the 16-bin configuration increases memory usage because of the larger amount of intermediate storage required for per-bin accumulation and compositing. To assess rendering quality, we use the single-bin renderer as a reference and report the relative mean squared error (MSE).

The relative MSE remains low for all tested configurations, indicating that depth binning introduces only minor image-space differences. No monotonic relationship is observed between the number of bins and rendering quality. Instead, the optimal bin count varies across datasets because changing the number of bins alters the placement of depth-bin boundaries along each ray, which can slightly affect sampling and alpha compositing near those boundaries. Consequently, the observed differences depend more strongly on primitive distributions, overlap patterns, and dataset characteristics than on the bin count itself. Based on these results, we use 8 depth bins as the default configuration throughout the paper, providing a robust balance between rendering performance, memory consumption, and image quality. For memory-constrained scenarios, 4 bins offer a more memory-efficient alternative while maintaining comparable visual fidelity.

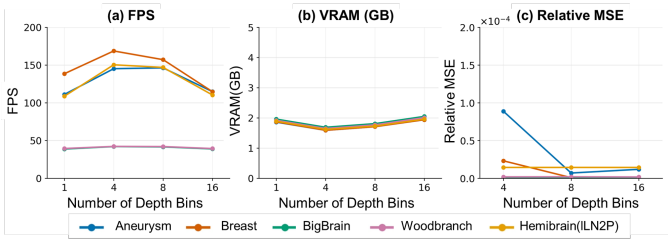


Fig. 8: Effect of depth bin count on (a) rendering performance (FPS), (b) GPU memory usage, and (c) rendering quality.