4EVER: Efficient and Exact Rendering for Volumetric Video with 4D Ellipsoids

Florian Hahlbohm[1] Linus Franke[2] Vladislav Golyanik[3] Martin Eisemann[1] Marcus Magnor[1,4]

[1]TU Braunschweig [2]University of Würzburg [3]Max Planck Institute for Informatics, SIC [4]University of New Mexico

https://fhahlbohm.github.io/4ever



Figure 1. 4EVER reconstructs volumetric video with 4D ellipsoids and renders it in real time. For efficient training, we use a core/tail approximation inspired by hybrid transparency [21, 51], where only the core accounts for overlapping primitives, while our WebGL viewer (inset) is fully exact. We show a novel view with optical flow, depth, and core share, *i.e.*, the opacity resolved by the core (left). 4EVER improves novel views of moving content on established benchmarks over baselines [13, 38, 39, 42, 81] (right).

Abstract

Dynamic scene reconstruction methods based on 4D Gaussians suffer from popping, aliasing, and flickering because they inherit the rasterizer approximations of 3D Gaussian Splatting (3DGS). They project primitives instead of integrating them along the ray, blend them in a per-view global order, and sample them without regard to the pixel footprint or the frame rate. Exact Volumetric Ellipsoid Rendering (EVER) removes the first two approximations for static scenes, but it requires a ray tracer, and its hard-edged primitives represent fire and smoke poorly.

We present 4EVER, which builds on the observation that exact rendering is affordable in a rasterizer if it is limited to where it matters. First, to render every frame exactly, we represent the scene with 4D ellipsoids, whose temporal slices are 3D ellipsoids, and give them soft Epanechnikov edges and anti-aliasing in space and time. Second, to render in real time, our rasterizer integrates the nearest primitives of every pixel exactly, including their overlap, and blends the remainder in a per-tile depth order, and we regularize our models to concentrate the opacity in the nearest primitives. Third, to capture fast motion, we initialize moving content densely with velocities and give every primitive its own optimizer clock, so that short-lived primitives do not fade out between their observations. On established benchmarks, 4EVER reduces mean squared error over prior art by more than 10% on average, both on full frames and in moving regions. Complemented by a controlled comparison with 4D Gaussians and a WebGL viewer for mobile devices, 4EVER makes exact rendering practical for volumetric video.

1. Introduction

Volumetric video, the reconstruction of a dynamic scene from synchronized multi-view recordings, has become practical with primitive-based radiance fields. Following 3D Gaussian Splatting (3DGS) [32], dynamic scenes are represented by deforming or tracking a canonical set of 3D Gaussians [47, 74, 80], by equipping 3D Gaussians with an explicit motion and lifetime [42, 71], or by lifting the primitives to four dimensions, so that every frame is a temporal slice of a 4D representation [13, 78, 81]. All of these methods render with the rasterizer of 3DGS and inherit its approximations. Gaussians are projected with an affine approximation [89], their overlap along a ray is ignored, and their blending order is a per-view sort by center depth, which is inconsistent across views and causes popping [60]. Most

methods also sample the primitives without regard to the pixel footprint or the frame rate, which causes aliasing and flickering when the sampling rate deviates from that of the training data [83]. We show that a rasterizer can render volumetric video exactly where it matters and approximately elsewhere, in real time (Fig. 1).

Constant-density ellipsoids can instead be rendered *exactly* as a volume, overlap included [48], but Exact Volumetric Ellipsoid Rendering (EVER) relies on a ray tracer leading to hours of optimization time and achieving low frame rates. Furthermore, its hard-edged primitives represent volumetric effects such as fire and smoke poorly. Moreover, the benefits of accurate rendering have so far only been analyzed for static scenes [5, 29], although popping and flickering are especially disturbing in a video.

Our goal is to render volumetric video exactly and in real time on a single consumer GPU, and to measure what exactness changes for dynamic scenes. To this end, we present 4EVER, a method that reconstructs dynamic scenes with 4D ellipsoids and renders them in real time, exactly where it matters and approximately where it does not. We observe that exactness matters most for the nearest primitives of every pixel, because training can concentrate most of the opacity of a pixel in them (Fig. 1, core share). 4EVER builds on this observation with three elements. First, to render every frame exactly, we represent the scene with 4D ellipsoids, since the temporal slice of a 4D ellipsoid is a 3D ellipsoid that can be integrated in closed form. Since hard-edged ellipsoids represent volumetric effects poorly, we equip them with an Epanechnikov density profile, whose line integral remains closed-form for a single primitive and exact to second order where primitives overlap. We further make anti-aliasing part of the representation by adapting every primitive to the pixel footprint and the frame rate.

Second, to render in real time, our rasterizer builds on hybrid transparency [21, 51]. For every pixel, the K nearest primitives form a *core* that is integrated exactly, including the overlap between ellipsoids, while the remaining primitives form a *tail* that is blended in a per-tile depth order. The approximation is thereby confined to the tail, and dedicated training measures concentrate the opacity in the core.

Unlike hybrid transparency for Gaussians [21], our core resolves the overlap of primitives, and our tail is sorted rather than averaged, so it stays accurate even where the core holds little of the opacity.

Third, to capture fast motion, we initialize moving content densely with full 4D shapes and velocities and adapt confidence-aware densification [26] to 4D. Since a primitive that exists for only a few frames is rarely observed during training, the regularizers fade it out between its observations. We therefore give every primitive its own *observation clock* in Adam [35], which updates it only when it is observed, with a larger step the less often this happens.

We evaluate 4EVER on established benchmarks [41, 71] against recent baselines, all retrained under a common protocol that separates reconstruction quality from the photometric calibration of the held-out camera. 4EVER achieves the highest PSNR on both datasets, on full frames and in moving regions. Furthermore, a controlled comparison of both primitive types across render modes isolates the effect of each approximation and shows the benefits of accurate rendering on dynamic scenes. Our contributions are:

- A 4D ellipsoid representation whose temporal slices are 3D ellipsoids with constant or Epanechnikov density, anti-aliased by pixel footprint and frame rate (Sec. 3.2).
- A real-time hybrid transparency rasterizer that integrates the nearest primitives of every pixel exactly, including their overlap, with training-time measures that concentrate the opacity in the core (Sec. 3.3).
- A training scheme for 4D primitives with dense 4D shape initialization, confidence-aware densification adapted to 4D, and an observation clock for the optimizer (Sec. 3.4).
- A controlled analysis of exact versus approximate rendering on dynamic and static scenes under a protocol separating reconstruction from photometric calibration (Sec. 4.3).

Our implementation is available on our project page and includes both primitive types, all render modes, and a WebGL viewer based on stochastic transparency [15, 34, 72] that plays our reconstructions without the tail approximation, even on mobile devices.

2. Related Work

Primitive-Based Radiance Fields for Static Scenes. Neural radiance fields [1, 53] established differentiable volume rendering, and 3DGS [32] replaced ray marching with the splatting of anisotropic Gaussians [89] in a tile-based rasterizer. 3DGS has since been accelerated [23, 50] and extended with alternative densification [26, 33, 62], compact appearance [22, 52], and other primitives [19, 24, 25, 28, 45, 49, 65, 69]. These methods target static scenes, and most of them keep the rendering approximations of splatting, both of which we lift for volumetric video. Our observation clock generalizes visibility-masked Adam updates [12, 50] from primitives outside the current view to primitives unobserved for most of a sequence.

Accurate Rendering of Primitives. The affine projection and the global per-view sort of 3DGS cause distortion and popping [29, 60]. Evaluating Gaussians along rays [8, 54] removes the projection error and has been brought to rasterization for single Gaussians [3, 21, 30, 67, 76, 84]. Per-pixel sorting [60, 64] or order-independent transparency [27, 34, 66] removes the dependence on the global order, and hybrid transparency [21, 51] combines both by sorting the nearest fragments of a pixel and averaging the rest. Closest to our renderer, EVER [48] additionally integrates the overlap

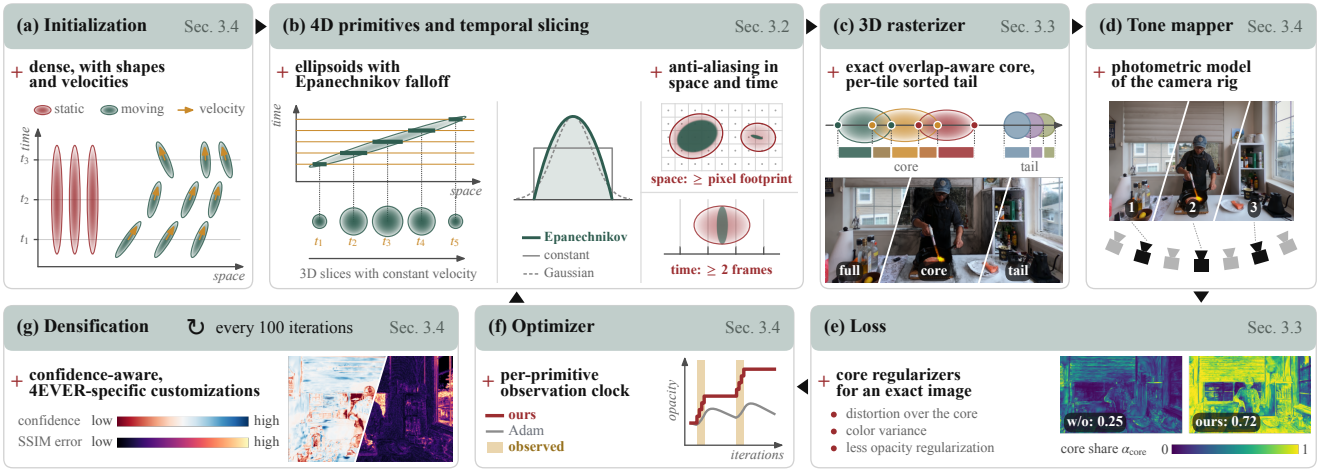

Figure 2. **Overview of 4EVER.** (a) We initialize 4D ellipsoids with shapes and velocities. (b) Their slices are 3D ellipsoids with an Epanechnikov falloff, anti-aliased in space and time. (c) We integrate the K nearest primitives per pixel exactly and blend the rest in sorted order. (d) A photometric model per camera [10] maps radiance to colors. (e) Regularizers raise the opacity α_{core} resolved by the core. (f) An observation clock updates primitives only when observed. (g) Densification [26] adds primitives at low confidence and high structural error.

of constant-density ellipsoids along the ray, which avoids distortion and popping but requires a hardware ray tracer. Other works integrate Gaussian or Epanechnikov kernels per primitive [2, 5, 8] or approximate the overlap with moment-based transmittance [56], and Softmax-GS [88] re-weights overlapping splats for sharper boundaries. Unlike EVER, we resolve the overlap of only the K nearest primitives, in a rasterizer, and soften the ellipsoids with an Epanechnikov profile. In contrast to hybrid transparency for Gaussians [21], our core integrates overlap and our tail is sorted per tile rather than averaged. As all these works consider static scenes, we extend the question of Celarek *et al*. [5], who found small gains from volume rendering at a high cost, to dynamic scenes, where ray tracing has so far only simulated camera effects [46].

Anti-Aliasing. Mip-Splatting [83] bounds Gaussians by the pixel footprint at the closest training camera, and analytic [43, 64] or multi-scale [79] integration refines the screen-space filter. Anti-aliasing of dynamic scenes has received less attention. Alias-free 4DGS [6] and Dhiman *et al*. [11] filter 4D Gaussians with a scale-adaptive and a motion-aware filter, respectively, to handle changes in spatial resolution, and RetimeGS [70] removes ghosting between frames with flow-supervised trajectories. These filters act on the image formation of Gaussians, whereas we filter the temporal slice of every primitive with the pixel footprint and bound its temporal extent by the frame spacing.

Dynamic Scene Reconstruction from Video. Dynamic radiance fields deform a canonical representation [58, 59], condition the field on time [41], or factorize space and time [4, 16]. With Gaussians, deformation fields [38, 74, 75, 77, 80] and per-frame tracking [47] were followed by 3D Gaussians with an explicit motion and lifetime, where the motion is polynomial [42], linear [71], rigid [82], or spline-based [70]. Native 4D Gaussians are instead conditioned on time [13, 18, 81] and have compact [85, 87] and hierarchical [78] variants, and several methods separate static from dynamic content [39, 57]. All of them render with the rasterizer of 3DGS and inherit the approximations that we address. SharpTimeGS [44] replaces the Gaussian temporal falloff with a flat-top lifespan to avoid fading, which the slice of a 4D ellipsoid provides by construction. Closest to our training scheme, FreeTimeGS++ [40] initializes velocities from dense flow at keyframes and corrects the colors of every training camera with a learned affine transformation. We instead initialize the full 4D shape of every primitive and model the cameras with PPISP [10], which restricts the correction to physically plausible camera effects (see Sec. C.2).

3. Method

Given synchronized videos of a dynamic scene from C calibrated cameras, with frame f captured at the normalized time $t_f \in [0, 1]$, our goal is to render the scene from a novel camera at any time $t \in [0, 1]$ in real time. We represent the scene with 4D primitives and optimize them with the video frames as supervision. See Fig. 2 for our pipeline.

3.1. Preliminaries

A primitive-based radiance field represents a scene with N primitives of density $\rho_i(\mathbf{x})$ and color $\mathbf{c}_i$. The color of a pixel is the volume rendering integral along its ray $\mathbf{o} + \lambda\mathbf{d}$,

$$\mathbf{C} = \int_0^\infty \sum_i \rho_i(\lambda)\, \mathbf{c}_i \exp\Big(-\int_0^\lambda \sum_j \rho_j(s)\, \mathrm{d}s\Big)\, \mathrm{d}\lambda, \quad (1)$$

where λ is the distance along the ray, $\rho_i(\lambda) = \rho_i(\mathbf{o} + \lambda\mathbf{d})$ the density of primitive i along it, and the exponential the transmittance $T(\lambda)$. 3DGS [32] instead projects Gaussians to the image plane with an affine approximation [89], assigns each a per-pixel opacity α_i, and alpha-composites them in the order of their center depths,

$$\mathbf{C} = \textstyle\sum_i \alpha_i T_i \mathbf{c}_i, \quad T_i = \prod_{j<i}(1-\alpha_j), \tag{2}$$

which ignores the overlap of primitives along the ray and depends on a global sort whose order can change abruptly between views. EVER [48] uses 3D ellipsoids of constant density σ_i. A ray enters and leaves the ellipsoid at λ_i^0 and λ_i^1, so the density along the ray is piecewise constant and Eq. (1) has a closed form. Between consecutive endpoints of the intersected primitives, the aggregate density $D = \sum_{i\in A}\sigma_i$ of the active set A is constant, and a segment of length δ contributes $(1 - e^{-D\delta})\, T \sum_{i\in A}\sigma_i \mathbf{c}_i / D$, with T at its start. Without overlap, this reduces to Eq. (2) with $\alpha_i = 1 - e^{-\tau}$ for the optical depth $\tau = \sigma_i(\lambda_i^1 - \lambda_i^0)$ of the chord.

3.2. 4D Ellipsoids

To render every frame exactly, we need a primitive whose temporal slice can be integrated along a ray in closed form. We therefore propose 4D ellipsoids, which extend the constant-density ellipsoids of EVER [48] to space-time and describe a 3D ellipsoid that moves at constant velocity for a finite lifetime. They have a mean $\boldsymbol{\mu}_i = (\boldsymbol{\mu}_i^x, \mu_i^t) \in \mathbb{R}^4$, four semi-axes, a 4D rotation given by a left- and a right-isoclinic quaternion [81], a density, and view-dependent appearance parameters. Each 4D shape matrix $\boldsymbol{\Sigma}_i$ has the spatial, temporal, and mixed blocks $\boldsymbol{\Sigma}_i^{xx}$, Σ_i^{tt}, and $\boldsymbol{\Sigma}_i^{xt}$. To render the frame at time t, we slice each ellipsoid at t, which yields a 3D ellipsoid with center $\boldsymbol{\mu}_{i|t}$ and shape matrix $k_i(t)\,\boldsymbol{\Sigma}_{i|t}$,

$$\boldsymbol{\mu}_{i|t} = \boldsymbol{\mu}_i^x + \boldsymbol{\Sigma}_i^{xt}\,(\Sigma_i^{tt})^{-1}\,(t - \mu_i^t), \tag{3}$$

$$\boldsymbol{\Sigma}_{i|t} = \boldsymbol{\Sigma}_i^{xx} - \boldsymbol{\Sigma}_i^{xt}\,(\Sigma_i^{tt})^{-1}\,\boldsymbol{\Sigma}_i^{tx}, \tag{4}$$

with $k_i(t) = 1 - (t-\mu_i^t)^2/\Sigma_i^{tt}$ (see Sec. A). The slice moves with the velocity $\boldsymbol{\Sigma}_i^{xt}/\Sigma_i^{tt}$ and keeps the shape $\boldsymbol{\Sigma}_{i|t}$ up to the scale $k_i(t)$. It shrinks toward the ends of its lifetime of $2\sqrt{\Sigma_i^{tt}}$ and disappears instead of fading, the geometric analog of the flat-top lifespan of SharpTimeGS [44]. A 4D Gaussian shares this parametrization but fades, since conditioning it on t gives the 3D Gaussian with mean $\boldsymbol{\mu}_{i|t}$ and covariance $\boldsymbol{\Sigma}_{i|t}$ whose opacity is scaled by the temporal marginal $\exp(-\frac{1}{2}(t-\mu_i^t)^2/\Sigma_i^{tt})$ [81]. A 4D ellipsoid has exactly the degrees of freedom of a 3D ellipsoid with a velocity and a lifetime, but its shape, lifetime, and velocity are optimized jointly as semi-axes and a rotation, which we found more stable than separate motion parameters. As the slice of a constant-density 4D ellipsoid has a constant density, the exact rendering of EVER applies to every frame. We use spherical harmonics for view-dependent appearance [32].

Density Parametrization. A constant density has a hard boundary, so content without sharp boundaries, such as fire, smoke, hair, or the blurred edges of fast motion, has to be assembled from many small primitives. Following Condor *et al*. [8], we also support the Epanechnikov profile $\rho_i(\mathbf{x}) = \sigma_i\,(1 - r^2)$ with the normalized radius r of the ellipsoid, whose slice at time t is again an Epanechnikov ellipsoid with peak density $k_i(t)\,\sigma_i$. Along a ray, the density is a parabola in λ, so a single primitive is still rendered exactly, and only the overlap of primitives is approximated (Sec. 3.3). EVER [48] caps the opacity along the shortest axis of a primitive at 0.99, so a primitive can still become fully opaque along its longest axis and block the gradients to everything behind it. We instead optimize the optical depth τ_i of the longest chord in log space and cap its opacity $1 - e^{-\tau_i}$ at 0.99, so that the cap holds along every chord. With the chord length ℓ_i, this gives $\sigma_i = \tau_i/\ell_i$ for the constant profile and $\sigma_i = {}^3\!/_2\,\tau_i/\ell_i$ for the Epanechnikov profile, which averages ${}^2\!/_3$ of its peak along the chord.

Spatial Anti-Aliasing. A primitive smaller than a pixel is either hit or missed by the pixel ray, which causes aliasing and starves thin primitives of gradients. We therefore impose the pixel footprint as a sampling condition on the temporal slice of every primitive, for both primitive types, rather than as a filter on the Gaussian image formation [6, 11]. Adapting Gaussian anti-aliasing [83] to ellipsoids, we dilate the conditional shape matrix of every primitive by the pixel footprint at its depth z, $\hat{\boldsymbol{\Sigma}}_{i|t} = k_i(t)\boldsymbol{\Sigma}_{i|t} + (z/f)^2\,\mathbf{I}$ with the focal length f, and rescale the density by $\sqrt{\det(k_i(t)\boldsymbol{\Sigma}_{i|t})/\det\hat{\boldsymbol{\Sigma}}_{i|t}}$, which preserves the rendered mass of the primitive. During optimization, we additionally bound the semi-axes from below by the footprint at the closest training camera [23, 83], which we found essential in 4D, where densification creates many tiny primitives that would otherwise die before they receive a gradient.

Temporal Anti-Aliasing. In time, the sampling condition is the frame rate instead of the pixel footprint. A primitive that exists for less than the frame spacing is visible in at most one frame and flickers at novel times. We therefore bound the temporal extent of every primitive from below by two frames, the Nyquist condition for the frame rate. With this bound, no primitive can appear and vanish between two frames, so the reconstruction can also be rendered at novel times, *e.g.*, in slow motion, without such flickering.

3.3. Efficient and Exact Rendering

Our renderer has to integrate overlapping primitives exactly while keeping the speed and the bounded per-pixel memory of a tile-based rasterizer. We extend existing fast tile-based rasterizers [21, 23] to primitives that are integrated along rays. In a first step, every primitive is sliced at time t and anti-aliased. For every covered pixel, we transform the ray

into the frame in which the ellipsoid is a unit sphere, given by a Cholesky factorization of its shape matrix, and intersect it with this sphere to obtain the entry and exit distances λ_i^0 and λ_i^1. We call the contribution of a primitive to a pixel a *fragment*. The fragments of a pixel are composited with a variant of hybrid transparency [21, 51] (Fig. 2c). The $K = 16$ nearest fragments form the *core*, which we integrate exactly, and all remaining fragments form the *tail*, which we composite behind the core in sorted order. Nearest refers to the entry distance λ_i^0 of an ellipsoid and to the depth of a Gaussian.

Exact Core. For ellipsoids, the core resolves the overlap of its fragments. We sort their $2K$ entry and exit points, and the pixel color follows from the segment integrals between consecutive points (Sec. 3.1). For a constant density, this is exact. For the Epanechnikov profile, the aggregate density of a segment is a quadratic, so its optical depth and transmittance are still exact. Only the split of the segment color among the active fragments requires $\int_0^1 e^{-\tau(x)} \rho_i(x)\, \mathrm{d}x$ with the segment coordinate x and a cubic optical depth $\tau(x)$, which we expand to second order around its linear part. The resulting pixel colors deviate from a numerical reference by 10^{-6} on average. The backward pass differentiates these integrals, so the geometry of a primitive receives gradients through its entry and exit points. We derive the integrals, the expansion, and all gradients in Sec. A, where we also visualize the error.

Sorted Tail. The tail is alpha-composited behind the core with Eq. (2), in a depth order that is established per tile. This order replaces the global sort by center depth of 3DGS at a small additional cost (Sec. 4). By construction, it is closer to exact rendering for large primitives, which cover many tiles at different depths. The tail only receives the transmittance that the core leaves, so both its approximation and its cost matter less the more opacity the core holds. We therefore skip the tail in the forward and the backward pass for every pixel whose core leaves a transmittance below 10^{-4}.

Reference Modes. Our full model uses the exact core and the sorted tail, which we call *hybrid with sorted tail* ("Ours" in the tables). To isolate each approximation in our experiments, the same rasterizer provides reference modes that share all other components. *Global sort* composites all primitives in the center-depth order of 3DGS with Eq. (2), which corresponds to EVER without overlap for ellipsoids. *Hybrid* (HT) is hybrid transparency as in HTGS [21], which keeps the core but accumulates the tail into a weighted average, which does not require sorting and is free of popping. For ellipsoids, *HT without overlap* selects and composites the core fragments by their chord midpoints.

Concentrating Opacity in the Core. A model trained with a sorted tail may represent a surface with many faint primitives that are spread over a long depth range, in which case the core holds only part of the opacity. We measure this with the *core share* $\alpha_{\mathrm{core}} = 1 - T_{\mathrm{core}}$, where T_{core} is the transmittance behind the core, averaged over all pixels. Under plain training, it stays between 0.23 and 0.69 on dynamic scenes (Tab. 2). We found the opacity regularizer of state-of-the-art densification approaches [26, 33] to amplify this issue and therefore lower its weight by a factor of ten. Three training measures raise the core share at no or small cost, and dense initialization raises it further. First, a distortion loss [1] over the segments of the core, evaluated in closed form with weights normalized over the core, pulls the mass of a pixel together. Second, when densification copies an elongated primitive, whose longest semi-axis exceeds 1.5× its second longest, the copies are placed side by side along that axis with their density preserved [9] instead of being stacked as thin layers of fragments. Third, we penalize the variance of the colors among the fragments of a ray [31] in the last third of training [72], which pulls tail fragments toward the core or merges them. Dense initialization (Sec. 3.4) helps because it starts from primitives that already sit on surfaces.

3.4. Initialization, Densification, and Training

Dense 4D Initialization. As in 4DGS [81], our initial point cloud is reconstructed from the first frame of the videos with multi-view stereo [63]. It is static, so the optimizer would have to grow all moving content by relocation. We therefore propose a dense initialization for 4D ellipsoids that covers the moving content and sets the 4D shape of every primitive. For every camera, we first detect the moving pixels, which deviate from the temporal median image by more than a robust noise estimate and typically cover a few percent of a frame. At every tenth frame, we triangulate these pixels against the three nearest cameras with learned dense correspondences [14], as in EDGS [36] for static scenes. The velocity of a point follows from warping it to the next keyframe within its camera and triangulating it there again. Every point becomes a 4D primitive with the keyframe time as temporal mean and semi-axes set from the keyframe interval and its nearest-neighbor distance. Its velocity is written into the mixed block of the shape matrix, $\mathbf{\Sigma}^{xt} = \mathbf{v}\,\Sigma^{tt}$, so that the primitive already moves correctly. Static content keeps the first-frame point cloud, with temporal means spread over the sequence.

Confidence-Aware Densification in 4D. We train under a fixed budget of primitives with the confidence-aware densification of Hofer *et al*. [26]. Every 100 iterations, the model grows by 5% toward the budget, and every 400 iterations, the 10% of primitives with the lowest rendered impact are relocated. New and relocated primitives are copied from sources sampled by their positional gradient, their learned confidence, and the structural error of their pixels. Copies share the optical depth of their source, which preserves the trans-

mittance of constant-density ellipsoids exactly (see Sec. A). They are placed within the spatial extent of their source at its time and inherit its time and velocity, since a temporal displacement would move static content outside the observed window, where it receives no gradient. We also relocate faint ellipsoids, since unlike faint Gaussians they are never culled by the rasterizer and would occupy core slots without contributing opacity. Finally, a *near-plane rule* relocates every primitive whose bounding sphere comes close to a training camera. Floaters in this region are invisible to the training views but can cover a held-out one, which we found to be a frequent issue with multi-view capture rigs.

Observation Clock. A primitive that is visible in only a few frames receives a gradient in few iterations, yet Adam [35] updates it every iteration. Regularizers and decaying moments therefore fade transient content out between its observations. We go beyond the visibility-masked Adam of static scenes [12, 50] and give every primitive its own clock. A primitive is updated only when it receives a non-zero gradient, its bias correction counts its own steps, and its step is scaled by $(1/p)^\gamma$ with its observation rate p, the share of recent iterations in which it had a non-zero gradient. We use $\gamma = 0.5$ for geometry and base color and $\gamma = 0$ for view-dependent appearance, so that rarely observed primitives stay close to view-independent. As the clock also stops the regularizers from fading unobserved primitives, we relocate primitives that received no update for 200 iterations. As the updates of a primitive no longer depend on how rarely it is observed, we can scale the batch size, *i.e.*, number of views per training iteration, with the length of a sequence, which makes our method applicable to sequences of any length.

Photometric Model of the Rig. The cameras of a rig may differ in exposure, white balance, and vignetting. We model these differences per training camera with PPISP [10], with a vignetting and a response curve per camera and an exposure and a color correction per frame, and apply the model to the rendered radiance before the loss.

Training Details. We optimize all parameters for 30k iterations with a batch size of four for sequences of 300 frames, using the fused Adam of Faster-GS [23] with our observation clock. The loss is the photometric loss of 3DGS, $0.8\,\mathcal{L}_1 + 0.2\,\mathcal{L}_{\text{D-SSIM}}$, between the output of the photometric model and the video frame, where every pixel is weighted by the rendered confidence [26]. We add the distortion loss and the color variance loss of Sec. 3.3 with weights 0.1 and 0.2, the latter from iteration 20k, and the scale regularizer of 3DGS-MCMC [33] with weight 0.01 until densification ends. We densify from iteration 600 to 24.9k. The semi-axes are optimized in log space and clamped to the spatial and the temporal bound of Sec. 3.2 after every step. Since a step in normalized time spans more frames in a longer sequence, we set the learning rate of the temporal means in frames, to 0.01 frames per step. Otherwise, we use the same learning rates for all scenes (see Sec. B.2).

4. Evaluation

4.1. Setup

We run all compared methods ourselves, which ensures that every method receives the same input images and is evaluated in the same way. Both affect the metrics and can even change the ranking of methods [37]. In particular, downscaling filters differ in how strongly they attenuate high frequencies, and smoother images are easier to fit, which inflates the metrics.

Datasets. We use the six scenes of the Neural 3D Video dataset [41] with 300 frames at 1352×1014 pixels, half the original resolution, and the first camera held out. We further use six 60-frame windows of SelfCap [71] at about 1885×1060 pixels (1062×1062 for the two bike windows), again with one camera held out. The dataset authors created these one-second windows by picking the hardest, *i.e.*, fastest-moving, parts of recordings of one to ten minutes, which is exactly what we seek to evaluate. Both datasets provide videos, which we convert to images with as little resampling as possible and without lossy compression to preserve their quality (see Sec. C.1). For the comparison of render modes on static scenes, we use the nine scenes of Mip-NeRF 360 [1] with the established protocol and the pre-downscaled images provided with the dataset.

Photometric Calibration. Photometric variation affects multi-view video even more than the single-camera captures of static scenes, as every camera of a rig has its own image signal processing. We therefore equip all baselines with the same PPISP model [10] that our method uses (Sec. 3.4). As this model has no parameters for the held-out camera, we fit a PPISP model to the frames of this camera for every method with the scene frozen (see Sec. C.2). This *calibrated* evaluation measures the quality of the reconstruction and not the ability of a method to explain photometric variation.

Metrics. We report PSNR, SSIM, and $\text{LPIPS}_{\text{VGG}}$ [86] over all frames of the held-out camera. As static content covers most of every frame, we also report mPSNR and mLPIPS [17], *i.e.*, PSNR and $\text{LPIPS}_{\text{Alex}}$ inside masks of the moving objects. A mask covers an object as a whole even if only a part of it moves, *e.g.*, an entire person while only an arm moves, which matches how humans perceive a dynamic scene (see Sec. C.3).

Baselines. We compare with 4DGS [81], 4D-Rotor GS [13], ST Gaussians [42], Ex4DGS [39], and MoRel [38]. 4DGS shares our 4D Gaussian parametrization, ST Gaussians is the strongest prior method on Neural 3D Video, Ex4DGS is the fastest, and MoRel a strong recent baseline. Since our protocol requires retraining, we cannot compare with

Table 1. Comparison on Neural 3D Video [41] and SelfCap [71]. ✓ and ✗ indicate whether a method avoids distortion artifacts (dist.), aliasing (AA), popping (pops), and overlap errors (overl.). Time includes the initialization and is measured on an RTX 4090. 4DGS is capped at 2.5M primitives, as it otherwise exceeds 24 GB of VRAM. The best and second best results are color coded.

Method	dist.	AA	pops	overl.	Neural 3D Video [41]						SelfCap [71]					
					mPSNR↑	mLPIPS↓	PSNR↑	SSIM↑	LPIPS↓	Time↓	mPSNR↑	mLPIPS↓	PSNR↑	SSIM↑	LPIPS↓	Time↓
4DGS [81]	✗	✗	✗	✗	32.06	0.075	33.12	0.944	0.193	2h23m	24.47	0.166	26.12	0.878	0.219	2h33m
4D-Rotor GS [13]	✗	✗	✗	✗	30.64	0.109	32.49	0.941	0.198	2h46m	24.45	0.193	26.84	0.899	0.192	5h48m
ST Gaussians [42]	✗	✗	✗	✗	32.08	0.067	33.44	0.946	0.192	2h26m	23.50	0.224	26.47	0.858	0.283	34m
Ex4DGS [39]	✗	✗	✗	✗	29.94	0.128	32.39	0.940	0.192	57m	23.14	0.257	26.81	0.856	0.247	1h01m
MoRel [38]	✗	✗	✗	✗	29.59	0.119	32.76	0.946	0.184	1h34m	22.51	0.273	26.94	0.883	0.225	2h23m
Ours	✓	✓	✓	✓	32.61	0.053	33.70	0.950	0.175	1h16m	25.22	0.152	27.77	0.897	0.194	30m

methods whose code was not released at the time of writing [6, 40, 44, 71, 82]. Beyond the input, dynamic methods differ in initialization, the number of views per iteration, hyperparameters, densification, parametrization, and rendering. We keep these as published, since they are part of each method, and therefore report training times that include the initialization. We further note that multiple baselines [13, 42, 81] use per-scene hyperparameters for improved results, which our method and the other baselines [38, 39] avoid.

Our Models. Our models use the configuration of Sec. 3.4, and our 4D models have a budget of one million primitives. For static scenes, we use the primitive budgets of Liu *et al*. [45]. We disable the observation clock for these scenes, as we found no benefit when primitives are observed far more frequently than the short-lived primitives of a dynamic scene.

4.2. Results

Table 1 compares our full model, *i.e*., 4D ellipsoids with the Epanechnikov profile, with the baselines. On Neural 3D Video, it achieves the best result in every quality metric, both on full frames and in the moving regions. On SelfCap, it leads in PSNR and in the moving regions, while 4D-Rotor GS is slightly ahead in SSIM and LPIPS. Its margin over the best baseline in mPSNR is largest on SelfCap, where the motion is fastest. Including the initialization, our model trains in about an hour and a quarter on Neural 3D Video and in about half an hour on SelfCap, whereas most baselines need more than two hours and only Ex4DGS is faster on Neural 3D Video. It renders at 252 FPS on Neural 3D Video (Tab. 2). Figure 1 (right) shows close-ups of fast-moving content from both datasets, each paired with a different baseline. The baselines blur such content or lose it entirely, *e.g*., the head of the jumping dog, the tongs, and the hair. Our reconstructions preserve it, although the finest structures under the fastest motion, such as single strands of hair, remain softer than in the ground truth. We show more qualitative results and per-scene metrics in Secs. D.3 and D.4 and demonstrate the improvements regarding rendering artifacts in our supplemental video.

4.3. Exact Versus Approximate Rendering

Table 2 compares the render modes of Sec. 3.3 for all three primitive types within our method. On Neural 3D Video, our mode achieves the highest PSNR for every primitive type, clearly above the global sort, while it is on par in the moving regions for ellipsoids and slightly behind for Gaussians, so its lead stems from the exact ordering of the static content. HT clearly falls behind in the moving regions for ellipsoids, likely because its order-independent tail fails when the core holds little of the opacity, a dependence that the sorted tail removes. Our core regularization raises the core share for both ellipsoid profiles, slightly improves PSNR, and accelerates rendering by skipping the tail for more pixels, at slightly increased training time. The Epanechnikov profile raises mPSNR over the constant density in every mode (Fig. 3) and matches Gaussians in our mode, at a lower core share and frame rate than the constant profile. Anti-aliasing has no notable effect on the quality at the training resolution.

On static scenes, in contrast, the global sort achieves the highest PSNR for every primitive type, the core regularization costs a little PSNR, and ellipsoids stay slightly behind Gaussians. Exactness thus does not pay off in per-frame metrics on static scenes (*cf*. [5]), whereas the per-pixel order of our renderer does on dynamic scenes. Such metrics do not capture popping, however, which our renderer avoids for all content that the core resolves (supplemental video).

4.4. Ablations

Table 3 ablates our training scheme for both primitive types and varies the initialization and the observation clock together, since the clock addresses the short-lived primitives that our initialization creates. Our initialization improves both primitive types on full frames and in the moving regions, with and without the clock. The clock degrades the moving regions with the static-only initialization, but with our initialization it improves Gaussians in all three metrics and leaves ellipsoids unchanged, while it also enables scaling to sequences of any length (Sec. 3.4). Among the densification methods, confidence-aware densification improves on the adaptive density control (ADC) of 3DGS for both primitive types and surpasses MCMC densification in the

Table 2. Render modes of our method for all three primitive types on dynamic and static scenes, with ✓ and ✗ as in Tab. 1. (✓) marks rows without core regularization, whose core resolves less of the opacity. Time, without initialization, and FPS are measured on an RTX 4090.

						Neural 3D Video [41] *(dynamic)*								Mip-NeRF 360 [1] *(static)*					
	Render mode	dist.	AA	pops	overl.	PSNR↑	SSIM↑	LPIPS↓	mPSNR↑	mLPIPS↓	α_{core}↑	Time↓	FPS↑	PSNR↑	SSIM↑	LPIPS↓	α_{core}↑	Time↓	FPS↑
Gaussian	Global sort w/o AA	✓	✗	✗	✗	33.05	0.947	0.178	32.68	0.053	–	32m	754	28.57	0.835	0.208	–	10m	344
	Global sort	✓	✓	✗	✗	33.21	0.947	0.178	32.70	0.054	–	32m	775	28.65	0.837	0.207	–	10m	348
	HT	✓	✓	✓	✗	33.56	0.948	0.173	32.29	0.054	0.68	37m	403	28.19	0.830	0.211	0.62	12m	232
	Ours	✓	✓	(✓)	✗	33.68	0.949	0.173	32.44	0.053	0.66	42m	296	28.35	0.833	0.209	0.54	14m	151
Constant	Global sort w/o AA	✓	✗	✗	✗	33.14	0.946	0.174	32.22	0.054	–	31m	885	28.30	0.833	0.218	–	10m	385
	Global sort	✓	✓	✗	✗	33.16	0.947	0.174	32.22	0.055	–	32m	867	28.34	0.834	0.217	–	10m	382
	HT w/o overlap	✓	✓	✓	✗	32.93	0.944	0.180	31.10	0.070	0.39	37m	412	26.86	0.808	0.246	0.26	13m	220
	HT	✓	✓	✓	✓	33.01	0.944	0.180	31.05	0.069	0.44	47m	362	26.84	0.808	0.246	0.22	15m	216
	Ours w/o core reg.	✓	✓	(✓)	(✓)	33.48	0.948	0.174	32.17	0.056	0.69	51m	286	28.21	0.833	0.217	0.44	16m	187
	Ours	✓	✓	✓	✓	33.63	0.948	0.176	32.25	0.051	0.97	55m	325	28.00	0.828	0.218	0.91	16m	261
Epanechnikov	Global sort w/o AA	✓	✗	✗	✗	33.13	0.947	0.178	32.52	0.056	–	36m	599	28.49	0.839	0.217	–	12m	317
	Global sort	✓	✓	✗	✗	33.05	0.947	0.178	32.51	0.057	–	37m	571	28.50	0.840	0.216	–	12m	315
	HT w/o overlap	✓	✓	✓	✗	33.15	0.946	0.180	31.52	0.072	0.11	43m	297	26.88	0.813	0.245	0.10	16m	171
	HT	✓	✓	✓	✓	33.08	0.946	0.180	31.42	0.073	0.12	56m	221	26.92	0.814	0.245	0.11	19m	138
	Ours w/o core reg.	✓	✓	(✓)	(✓)	33.56	0.949	0.176	32.48	0.057	0.23	58m	179	28.36	0.839	0.216	0.20	19m	107
	Ours	✓	✓	✓	✓	33.70	0.950	0.175	32.61	0.053	0.67	1h04m	252	28.23	0.836	0.211	0.69	18m	173

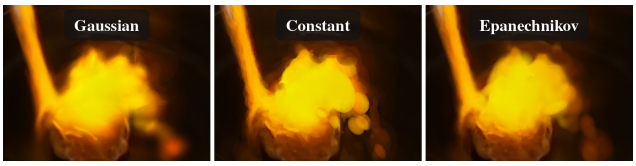

Figure 3. A flame at 4× resolution. Constant-density ellipsoids are clearly visible, an issue that the Epanechnikov profile reduces.

Table 3. Ablation of our initialization, observation clock, and the densification method (Sec. 3.4) on Neural 3D Video [41].

		Gaussian			Epanechnikov		
Initialization	Clock	mPSNR↑	mLPIPS↓	PSNR↑	mPSNR↑	mLPIPS↓	PSNR↑
static-only	✗	31.77	0.067	33.28	32.31	0.058	33.35
static-only	✓	31.43	0.071	33.18	32.03	0.063	33.47
Ours	✗	32.01	0.059	33.52	32.63	0.052	33.76
Ours w/ ADC [32]		32.19	0.053	33.15	31.93	0.063	32.95
Ours w/ MCMC [33]		31.99	0.062	33.68	32.28	0.057	33.58
Ours		32.44	0.053	33.68	32.61	0.053	33.70

moving regions.

5. Discussion

Does 4DGS Need Exact Volumetric Rendering? Per-frame metrics reward accurate rendering on dynamic scenes, where our mode achieves the highest PSNR for every primitive type (Tab. 2), but not on static scenes, in line with Celarek *et al*. [5]. Such metrics, however, do not capture the popping that accurate rendering removes, which is likely more disturbing in a video, where the camera and the content move, than in a still image. We therefore consider accurate rendering more important for volumetric video than for static scenes, and our rasterizer makes it affordable during training.

Exact Rendering at Inference. Our WebGL viewer slices the primitives at the current time, intersects every pixel ray with them, and replaces the core and tail of our rasterizer by stochastic transparency [15, 34]. In every sample, a fragment survives with a probability equal to its opacity at a depth drawn from its density along the ray, so that the mean over samples converges to Eq. (1), including the overlap of primitives. Alpha correction [15], stratified survival decisions, and the reprojection of previous frames [72] reduce the noise enough for interactive frame rates at few samples per frame, at the price of a small bias. Here, exactness only changes the depth at which a fragment is placed, so it costs little more than approximate blending. As exact rendering is this cheap at inference, the remaining challenge is to train models for it efficiently, which 4EVER addresses. We provide additional details in Sec. B.3.

Limitations. During training, the core does not hold all opacity, so part of every pixel is still composited approximately. This is a limitation of our implementation, as the core size is bounded by the per-pixel memory of our rasterizer, and our viewer resolves all fragments exactly, so only a mismatch between training and viewing remains. Our rasterizer is also slower than splatting Gaussians in a global order, which we accept since a volumetric video is reconstructed once and played many times, and rendering remains real time.

6. Conclusion

We presented 4EVER, which reconstructs volumetric video with 4D ellipsoids and renders it in real time, exactly where it matters. It achieves the best quality among recent methods, on full frames and in moving regions, while training faster than most of them. Our controlled comparison shows that accurate rendering pays off on dynamic scenes even in per-frame metrics and avoids artifacts such as popping that these metrics do not capture. As our rasterizer and viewer make it affordable during training and at inference, we consider accurate rendering a practical default for volumetric video.

Acknowledgments

We thank Mathias Ivanov, Jon Barron, and Thomas Köhler for helpful discussions on individual components of our method. We also thank Timon Scholz and Kjell Keune for their contributions to the software infrastructure supporting this work. This work was partially funded by the DFG projects "Real-Action VR" (ID 523421583) and "Increasing Realism of Omnidirectional Videos in Virtual Reality" (ID 491805996).

References

[1] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded anti-aliased neural radiance fields. In *CVPR*, pages 5460–5469, 2022. 2, 5, 6, 8, 17

[2] Hugo Blanc, Jean-Emmanuel Deschaud, and Alexis Paljic. RayGauss: Volumetric Gaussian-based ray casting for photorealistic novel view synthesis. In *WACV*, pages 1808–1817, 2025. 3

[3] Samuel Rota Bulò, Nemanja Bartolovic, Lorenzo Porzi, and Peter Kontschieder. Hardware-rasterized ray-based Gaussian splatting. In *CVPR*, pages 485–494, 2025. 2

[4] Ang Cao and Justin Johnson. HexPlane: A fast representation for dynamic scenes. In *CVPR*, pages 130–141, 2023. 3

[5] Adam Celarek, Georgios Kopanas, George Drettakis, Michael Wimmer, and Bernhard Kerbl. Does 3D Gaussian splatting need accurate volumetric rendering? *Comput. Graph. Forum*, 44(2):e70032, 2025. 2, 3, 7, 8

[6] Zilong Chen, Huan-ang Gao, Delin Qu, Haohan Chi, Hao Tang, Kai Zhang, and Hao Zhao. Alias-free 4D Gaussian splatting, 2025. 3, 4, 7

[7] Ho Kei Cheng, Seoung Wug Oh, Brian Price, Joon-Young Lee, and Alexander Schwing. Putting the object back into video object segmentation. In *CVPR*, pages 3151–3161, 2024. 18, 19

[8] Jorge Condor, Sebastien Speierer, Lukas Bode, Aljaz Bozic, Simon Green, Piotr Didyk, and Adrian Jarabo. Don't splat your Gaussians: Volumetric ray-traced primitives for modeling and rendering scattering and emissive media. *ACM TOG*, 44(1), 2025. 2, 3, 4

[9] Xiaobin Deng, Changyu Diao, Min Li, Ruohan Yu, and Duanqing Xu. Improving densification in 3D Gaussian splatting for high-fidelity rendering, 2025. 5, 16

[10] Isaac Deutsch, Nicolas Moënne-Loccoz, Gavriel State, and Zan Gojcic. PPISP: Physically-plausible compensation and control of photometric variations in radiance field reconstruction. In *CVPR*, pages 7289–7298, 2026. 3, 6, 18

[11] Ankit Dhiman, Kunal A. Kathare, Pranav Vignesh, Lokesh R. Boregowda, and Venkatesh Babu Radhakrishnan. Towards alias-free 4D Gaussian representations with motion-aware filtering. In *ECCV*, pages 480–499, 2026. 3, 4

[12] Renjie Ding, Yaonan Wang, Min Liu, Jialin Zhu, Jiazheng Wang, Jiahao Zhao, Wenting Shen, Feixiang He, and Xiang Chen. A step to decouple optimization in 3DGS. In *ICLR*, 2026. 2, 6

[13] Yuanxing Duan, Fangyin Wei, Qiyu Dai, Yuhang He, Wenzheng Chen, and Baoquan Chen. 4D-Rotor Gaussian Splatting: Towards efficient novel view synthesis for dynamic scenes. In *SIGGRAPH*, 2024. 1, 3, 6, 7, 20, 21

[14] Johan Edstedt, David Nordström, Yushan Zhang, Georg Bökman, Jonathan Astermark, Viktor Larsson, Anders Heyden, Fredrik Kahl, Mårten Wadenbäck, and Michael Felsberg. RoMa v2: Harder better faster denser feature matching. In *ECCV*, pages 664–682, 2026. 5

[15] Eric Enderton, Erik Sintorn, Peter Shirley, and David Luebke. Stochastic transparency. In *I3D*, pages 157–164, 2010. 2, 8, 17

[16] Sara Fridovich-Keil, Giacomo Meanti, Frederik Rahbæk Warburg, Benjamin Recht, and Angjoo Kanazawa. K-Planes: Explicit radiance fields in space, time, and appearance. In *CVPR*, pages 12479–12488, 2023. 3

[17] Hang Gao, Ruilong Li, Shubham Tulsiani, Bryan Russell, and Angjoo Kanazawa. Monocular dynamic view synthesis: A reality check. In *NeurIPS*, pages 33768–33780, 2022. 6, 19

[18] Zhongpai Gao, Benjamin Planche, Meng Zheng, Anwesa Choudhuri, Terrence Chen, and Ziyan Wu. 7DGS: Unified spatial-temporal-angular Gaussian splatting. In *ICCV*, pages 26316–26325, 2025. 3

[19] Shrisudhan Govindarajan, Daniel Rebain, Kwang Moo Yi, and Andrea Tagliasacchi. Radiant Foam: Real-time differentiable ray tracing. In *ICCV*, pages 4135–4145, 2025. 2

[20] Stefan Gumhold. Splatting illuminated ellipsoids with depth correction. In *Vis. Model. Vis.*, 2003. 17

[21] Florian Hahlbohm, Fabian Friederichs, Tim Weyrich, Linus Franke, Moritz Kappel, Susana Castillo, Marc Stamminger, Martin Eisemann, and Marcus Magnor. Efficient perspective-correct 3D Gaussian splatting using hybrid transparency. *Comput. Graph. Forum*, 44(2):e70014, 2025. 1, 2, 3, 4, 5, 15, 16, 17

[22] Florian Hahlbohm, Jorge Condor, Linus Franke, Martin Eisemann, and Marcus Magnor. Compact neural appearance models for efficient Gaussian splatting, 2026. 2, 16, 17

[23] Florian Hahlbohm, Linus Franke, Martin Eisemann, and Marcus Magnor. Faster-GS: Analyzing and improving Gaussian splatting optimization. In *CVPR*, pages 18946–18957, 2026. 2, 4, 6, 16

[24] Jan Held, Renaud Vandeghen, Abdullah Hamdi, Adrien Deliege, Anthony Cioppa, Silvio Giancola, Andrea Vedaldi, Bernard Ghanem, and Marc Van Droogenbroeck. 3D convex splatting: Radiance field rendering with 3D smooth convexes. In *CVPR*, pages 21360–21369, 2025. 2

[25] Jan Held, Renaud Vandeghen, Adrien Deliege, Abdullah Hamdi, Silvio Giancola, Daniel Rebain, Anthony Cioppa, Bernard Ghanem, Andrea Vedaldi, Andrea Tagliasacchi, and Marc Van Droogenbroeck. Triangle splatting for real-time radiance field rendering. In *3DV*, pages 1248–1257, 2026. 2

[26] Manuel Hofer, Thomas Köhler, Felix Windisch, Bernhard Kerbl, Markus Steinberger, and Lukas Radl. Efficient confidence-aware densification for 3DGS, 2026. 2, 3, 5, 6, 16, 19

[27] Qiqi Hou, Randall Rauwendaal, Zifeng Li, Hoang Le, Farzad Farhadzadeh, Fatih Porikli, Alexei Bourd, and Amir Said.

Sort-free Gaussian splatting via weighted sum rendering. In *ICLR*, 2025. 2

[28] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2D Gaussian splatting for geometrically accurate radiance fields. In *SIGGRAPH*, pages 32:1–32:11, 2024. 2

[29] Letian Huang, Jiayang Bai, Jie Guo, Yuanqi Li, and Yanwen Guo. On the error analysis of 3D Gaussian splatting and an optimal projection strategy. In *ECCV*, pages 247–263, 2024. 2

[30] Zixun Huang, Cho-Ying Wu, Yuliang Guo, Xinyu Huang, and Liu Ren. 3DGEER: 3D Gaussian rendering made exact and efficient for generic cameras. In *ICLR*, 2026. 2

[31] Shubhendu Jena, Amine Ouasfi, Mae Younes, and Adnane Boukhayma. Sparfels: Fast reconstruction from sparse unposed imagery. In *ICCV*, pages 27476–27487, 2025. 5

[32] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 3D Gaussian splatting for real-time radiance field rendering. *ACM TOG*, 42(4), 2023. 1, 2, 4, 8, 16

[33] Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Weiwei Sun, Yang-Che Tseng, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. 3D Gaussian splatting as Markov chain Monte Carlo. In *NeurIPS*, pages 80965–80986, 2024. 2, 5, 6, 8, 16

[34] Shakiba Kheradmand, Delio Vicini, George Kopanas, Dmitry Lagun, Kwang Moo Yi, Mark Matthews, and Andrea Tagliasacchi. StochasticSplats: Stochastic rasterization for sorting-free 3D Gaussian splatting. In *ICCV*, pages 26326–26335, 2025. 2, 8, 17

[35] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 2, 6

[36] Dmytro Kotovenko, Olga Grebenkova, and Björn Ommer. EDGS: Eliminating densification for efficient convergence of 3DGS. In *CVPR*, pages 41065–41076, 2026. 5, 16

[37] Jonas Kulhanek and Torsten Sattler. NerfBaselines: Consistent and reproducible evaluation of novel view synthesis methods. In *NeurIPS*, pages 127378–127403, 2025. 6

[38] Sangwoon Kwak, Weeyoung Kwon, Jun Young Jeong, Geonho Kim, Won-Sik Cheong, and Jihyong Oh. MoRel: Long-range flicker-free 4D motion modeling via anchor relay-based bidirectional blending with hierarchical densification. In *CVPR*, pages 37370–37379, 2026. 1, 3, 6, 7, 20, 21

[39] Junoh Lee, Changyeon Won, Hyunjun Jung, Inhwan Bae, and Hae-Gon Jeon. Fully explicit dynamic Gaussian splatting. In *NeurIPS*, pages 5384–5409, 2024. 1, 3, 6, 7, 20, 21

[40] Lucas Yunkyu Lee, Soonho Kim, Youngwook Kim, Sangmin Kim, and Jaesik Park. FreeTimeGS++: Secrets of dynamic Gaussian splatting and their principles, 2026. 3, 7, 18

[41] Tianye Li, Mira Slavcheva, Michael Zollhoefer, Simon Green, Christoph Lassner, Changil Kim, Tanner Schmidt, Steven Lovegrove, Michael Goesele, Richard Newcombe, and Zhaoyang Lv. Neural 3D video synthesis from multi-view video. In *CVPR*, pages 5511–5521, 2022. 2, 3, 6, 7, 8, 17, 18, 19, 20, 21

[42] Zhan Li, Zhang Chen, Zhong Li, and Yi Xu. Spacetime Gaussian feature splatting for real-time dynamic view synthesis. In *CVPR*, pages 8508–8520, 2024. 1, 3, 6, 7, 20, 21

[43] Zhihao Liang, Qi Zhang, Wenbo Hu, Lei Zhu, Ying Feng, and Kui Jia. Analytic-Splatting: Anti-aliased 3D Gaussian splatting via analytic integration. In *ECCV*, pages 281–297, 2024. 3

[44] Zhanfeng Liao, Jiajun Zhang, Hanzhang Tu, Zhixi Wang, Yunqi Gao, Hongwen Zhang, and Yebin Liu. SharpTimeGS: Sharp and stable dynamic Gaussian splatting via lifespan modulation. In *CVPR*, pages 11798–11807, 2026. 3, 4, 7

[45] Rong Liu, Dylan Sun, Meida Chen, Yue Wang, and Andrew Feng. Deformable beta splatting. In *SIGGRAPH*, pages 101:1–101:11, 2025. 2, 7

[46] Yi-Ruei Liu, You-Zhe Xie, Yu-Hsiang Hsu, Jing-En Huang, I-Sheng Fang, Yu-Lun Liu, and Jun-Cheng Chen. Every camera effect, every time, all at once: 4D Gaussian ray tracing for physics-based camera effect data generation. In *WACV*, 2027. 3

[47] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. Dynamic 3D Gaussians: Tracking by persistent dynamic view synthesis. In *3DV*, pages 800–809, 2024. 1, 3

[48] Alexander Mai, Peter Hedman, George Kopanas, Dor Verbin, David Futschik, Qiangeng Xu, Falko Kuester, Jonathan T. Barron, and Yinda Zhang. EVER: Exact volumetric ellipsoid rendering for real-time view synthesis. In *ICCV*, pages 4930–4939, 2025. 2, 4, 13

[49] Alexander Mai, Trevor Hedstrom, George Kopanas, Janne Kontkanen, Falko Kuester, and Jonathan T. Barron. Radiance meshes for volumetric reconstruction. In *CVPR*, pages 8267–8277, 2026. 2

[50] Saswat Subhajyoti Mallick, Rahul Goel, Bernhard Kerbl, Markus Steinberger, Francisco Vicente Carrasco, and Fernando De La Torre. Taming 3DGS: High-quality radiance fields with limited resources. In *SIGGRAPH Asia*, pages 2:1–2:11, 2024. 2, 6, 16

[51] Marilena Maule, João Comba, Rafael Torchelsen, and Rui Bastos. Hybrid transparency. In *I3D*, pages 103–118, 2013. 1, 2, 5

[52] Ewa Miazga, Jorge Condor, and Piotr Didyk. Beyond spherical harmonics: Rethinking appearance models for radiance reconstruction. *Comput. Graph. Forum*, 2026. 2

[53] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, pages 405–421, 2020. 2

[54] Nicolas Moenne-Loccoz, Ashkan Mirzaei, Or Perel, Riccardo de Lutio, Janick Martinez Esturo, Gavriel State, Sanja Fidler, Nicholas Sharp, and Zan Gojcic. 3D Gaussian Ray Tracing: Fast tracing of particle scenes. *ACM TOG*, 43(6), 2024. 2

[55] Thomas Müller. tiny-cuda-nn, 2021. 16

[56] Jan U. Müller, Robin Tim Landsgesell, Leif Van Holland, Patrick Stotko, and Reinhard Klein. Moment-based 3D Gaussian splatting: Resolving volumetric occlusion with order-independent transmittance, 2025. 3

[57] Seungjun Oh, Younggeun Lee, Hyejin Jeon, and Eunbyung Park. Hybrid 3D-4D Gaussian splatting for fast dynamic scene representation, 2025. 3

[58] Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo

Martin-Brualla. Nerfies: Deformable neural radiance fields. In *ICCV*, pages 5845–5854, 2021. 3

[59] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-NeRF: Neural radiance fields for dynamic scenes. In *CVPR*, pages 10313–10322, 2021. 3

[60] Lukas Radl, Michael Steiner, Mathias Parger, Alexander Weinrauch, Bernhard Kerbl, and Markus Steinberger. StopThePop: Sorted Gaussian splatting for view-consistent real-time rendering. *ACM TOG*, 43(4), 2024. 1, 2

[61] Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, and Laura Gustafson et al. SAM 2: Segment anything in images and videos. In *ICLR*, 2025. 18

[62] Samuel Rota Bulò, Lorenzo Porzi, and Peter Kontschieder. Revising densification in Gaussian splatting. In *ECCV*, pages 347–362, 2024. 2

[63] Johannes L. Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *ECCV*, pages 501–518, 2016. 5

[64] Michael Steiner, Thomas Köhler, Lukas Radl, Felix Windisch, Dieter Schmalstieg, and Markus Steinberger. AAA-Gaussians: Anti-aliased and artifact-free 3D Gaussian rendering. In *ICCV*, pages 27650–27659, 2025. 2, 3

[65] Cheng Sun, Jaesung Choe, Charles Loop, Wei-Chiu Ma, and Yu-Chiang Frank Wang. Sparse voxels rasterization: Real-time high-fidelity radiance field rendering. In *CVPR*, pages 16187–16196, 2025. 2

[66] Xin Sun, Iliyan Georgiev, Yun Fei, and Milos Hasan. Stochastic ray tracing of transparent 3D Gaussians. In *EGSR*, 2025. 2

[67] Chinmay Talegaonkar, Yash Belhe, Ravi Ramamoorthi, and Nicholas Antipa. Volumetrically consistent 3D Gaussian rasterization. In *CVPR*, pages 10953–10963, 2025. 2

[68] Zachary Teed and Jia Deng. RAFT: Recurrent all-pairs field transforms for optical flow. In *ECCV*, pages 402–419, 2020. 18

[69] Nicolas von Lützow and Matthias Nießner. LinPrim: Linear primitives for differentiable volumetric rendering. In *NeurIPS*, pages 24405–24431, 2025. 2

[70] Xuezhen Wang, Li Ma, Yulin Shen, Zeyu Wang, and Pedro V. Sander. RetimeGS: Continuous-time reconstruction of 4D Gaussian splatting. In *CVPR*, pages 7340–7350, 2026. 3

[71] Yifan Wang, Peishan Yang, Zhen Xu, Jiaming Sun, Zhanhua Zhang, Yong Chen, Hujun Bao, Sida Peng, and Xiaowei Zhou. FreeTimeGS: Free Gaussian primitives at anytime anywhere for dynamic scene reconstruction. In *CVPR*, pages 21750–21760, 2025. 1, 2, 3, 6, 7, 17, 18, 20, 21

[72] Felix Windisch, Thomas Köhler, Lukas Radl, Chris Wyman, Georgios Kopanas, Bernhard Kerbl, and Markus Steinberger. SteadySplats: Resampling of low-variance Gaussians for high-fidelity stochastic rendering, 2026. 2, 5, 8, 17

[73] World Labs. Spark, 2025. 17

[74] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 4D Gaussian splatting for real-time dynamic scene rendering. In *CVPR*, pages 20310–20320, 2024. 1, 3

[75] Jiahao Wu, Rui Peng, Zhiyan Wang, Lu Xiao, Luyang Tang, Jinbo Yan, Kaiqiang Xiong, and Ronggang Wang. Swift4D: Adaptive divide-and-conquer Gaussian splatting for compact and efficient reconstruction of dynamic scene. In *ICLR*, 2025. 3

[76] Qi Wu, Janick Martinez Esturo, Ashkan Mirzaei, Nicolas Moenne-Loccoz, and Zan Gojcic. 3DGUT: Enabling distorted cameras and secondary rays in Gaussian splatting. In *CVPR*, pages 26036–26046, 2025. 2

[77] Jiawei Xu, Zexin Fan, Jian Yang, and Jin Xie. Grid4D: 4D decomposed hash encoding for high-fidelity dynamic Gaussian splatting. In *NeurIPS*, pages 123787–123811, 2024. 3

[78] Zhen Xu, Yinghao Xu, Zhiyuan Yu, Sida Peng, Jiaming Sun, Hujun Bao, and Xiaowei Zhou. Representing long volumetric video with temporal Gaussian hierarchy. *ACM TOG*, 43(6), 2024. 1, 3

[79] Zhiwen Yan, Weng Fei Low, Yu Chen, and Gim Hee Lee. Multi-scale 3D Gaussian splatting for anti-aliased rendering. In *CVPR*, pages 20923–20931, 2024. 3

[80] Ziyi Yang, Xinyu Gao, Wen Zhou, Shaohui Jiao, Yuqing Zhang, and Xiaogang Jin. Deformable 3D Gaussians for high-fidelity monocular dynamic scene reconstruction. In *CVPR*, pages 20331–20341, 2024. 1, 3

[81] Zeyu Yang, Hongye Yang, Zijie Pan, and Li Zhang. Real-time photorealistic dynamic scene representation and rendering with 4D Gaussian splatting. In *ICLR*, 2024. 1, 3, 4, 5, 6, 7, 12, 16, 20, 21

[82] Suwoong Yeom, Joonsik Nam, Seunggyu Choi, Lucas Yunkyu Lee, Sangmin Kim, Jaesik Park, Joonsoo Kim, Kugjin Yun, Kyeongbo Kong, and Suk-Ju Kang. TRiGS: Temporal rigid-body motion for scalable 4D Gaussian splatting. In *ECCV*, pages 230–248, 2026. 3, 7

[83] Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. Mip-Splatting: Alias-free 3D Gaussian splatting. In *CVPR*, pages 19447–19456, 2024. 2, 3, 4

[84] Zehao Yu, Torsten Sattler, and Andreas Geiger. Gaussian Opacity Fields: Efficient adaptive surface reconstruction in unbounded scenes. *ACM TOG*, 43(6), 2024. 2

[85] Yuheng Yuan, Qiuhong Shen, Xingyi Yang, and Xinchao Wang. 1000+ FPS 4D Gaussian splatting for dynamic scene rendering. In *NeurIPS*, pages 83135–83153, 2025. 3

[86] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, pages 586–595, 2018. 6

[87] Xinjie Zhang, Zhening Liu, Yifan Zhang, Xingtong Ge, Dailan He, Tongda Xu, Yan Wang, Zehong Lin, Shuicheng Yan, and Jun Zhang. MEGA: Memory-efficient 4D Gaussian splatting for dynamic scenes. In *ICCV*, pages 27828–27838, 2025. 3

[88] Chen Ziwen, Peng Wang, Hao Tan, Zexiang Xu, and Li Fuxin. Softmax-GS: Generalized Gaussians learning when to blend or bound. In *CVPR Findings*, pages 508–517, 2026. 3

[89] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. EWA volume splatting. In *IEEE Vis.*, pages 29–38, 2001. 1, 2, 4

A. Derivations

A.1. Temporal Slice of a 4D Ellipsoid

A 4D ellipsoid with mean $\boldsymbol{\mu} = (\boldsymbol{\mu}^x, \mu^t)$ and shape matrix $\boldsymbol{\Sigma}$ consists of all points $\mathbf{y} = (\mathbf{x}, t)$ of space-time whose squared normalized radius

$$r^2(\mathbf{y}) = (\mathbf{y} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{y} - \boldsymbol{\mu}) \tag{5}$$

is at most one. To slice it at time t, we separate space and time in this quadratic form. Let $\boldsymbol{\Sigma}^{xx}$, Σ^{tt}, and $\boldsymbol{\Sigma}^{xt}$ be the spatial, temporal, and mixed blocks of $\boldsymbol{\Sigma}$. With the velocity $\mathbf{v} = \boldsymbol{\Sigma}^{xt}/\Sigma^{tt}$ and the matrix

$$\boldsymbol{\Sigma}_{|t} = \boldsymbol{\Sigma}^{xx} - \boldsymbol{\Sigma}^{xt} (\Sigma^{tt})^{-1} \boldsymbol{\Sigma}^{tx}, \tag{6}$$

the shape matrix factors into

$$\boldsymbol{\Sigma} = \begin{pmatrix} \mathbf{I} & \mathbf{v} \\ \mathbf{0}^\top & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{\Sigma}_{|t} & \mathbf{0} \\ \mathbf{0}^\top & \Sigma^{tt} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{v}^\top & 1 \end{pmatrix}, \tag{7}$$

which is verified by multiplying out the blocks. The outer factors are shears along time. The inverse of the left one maps $\mathbf{y} - \boldsymbol{\mu}$ to $(\mathbf{x} - \boldsymbol{\mu}_{|t},\, t - \mu^t)$ with the center $\boldsymbol{\mu}_{|t} = \boldsymbol{\mu}^x + \mathbf{v}\,(t - \mu^t)$, *i.e.*, to coordinates that move with the primitive. In these coordinates, the quadratic form is block diagonal, and Eq. (5) becomes

$$r^2(\mathbf{x}, t) = r^2_{|t}(\mathbf{x}) + \frac{(t - \mu^t)^2}{\Sigma^{tt}}, \tag{8}$$

with the spatial part $r^2_{|t}(\mathbf{x}) = (\mathbf{x} - \boldsymbol{\mu}_{|t})^\top \boldsymbol{\Sigma}^{-1}_{|t} (\mathbf{x} - \boldsymbol{\mu}_{|t})$. The points of the slice thus satisfy $r^2_{|t}(\mathbf{x}) \le k(t)$ with $k(t) = 1 - (t - \mu^t)^2/\Sigma^{tt}$. This is the 3D ellipsoid with center $\boldsymbol{\mu}_{|t}$ and shape matrix $k(t)\,\boldsymbol{\Sigma}_{|t}$, as stated in Eqs. (3) and (4). It shrinks toward the ends of the lifetime and is empty for $k(t) \le 0$ (Fig. 4a).

Density of the Slice. A constant density σ inside the 4D ellipsoid is the same constant inside every slice. For the Epanechnikov profile $\rho = \sigma\,(1 - r^2)$, Eq. (8) gives

$$\rho(\mathbf{x}, t) = \sigma \left(k(t) - r^2_{|t}(\mathbf{x})\right) = k(t)\,\sigma \left(1 - \frac{r^2_{|t}(\mathbf{x})}{k(t)}\right), \tag{9}$$

where $r^2_{|t}/k(t)$ is the squared normalized radius of the slice. The slice is therefore again an Epanechnikov ellipsoid, whose peak density $k(t)\,\sigma$ decreases toward the ends of the lifetime while its extent shrinks (Fig. 4b). For a 4D Gaussian $\exp(-1/2\, r^2)$, the same separation yields the 3D Gaussian with covariance $\boldsymbol{\Sigma}_{|t}$ and the factor $\exp(-1/2\,(t - \mu^t)^2/\Sigma^{tt})$, which keeps its extent and only fades [81].

Stable Evaluation. Equation (6) subtracts two nearly equal matrices for a fast primitive, whose spatial extent $\boldsymbol{\Sigma}^{xx}$ is dominated by its motion. In single precision, the difference can turn indefinite, and the affected primitives flicker between training iterations. We therefore evaluate $\boldsymbol{\Sigma}_{|t}$ without subtracting matrices. Let $\mathbf{m}_1, \ldots, \mathbf{m}_4$ be the rows of $\mathbf{M} = \mathbf{RS}$, where $\mathbf{R}$ is the 4D rotation and $\mathbf{S}$ the diagonal matrix of the semi-axes $s_1, \ldots, s_4$, so that $\boldsymbol{\Sigma} = \mathbf{M}\mathbf{M}^\top$. We remove from every spatial row its component along the temporal row,

$$\tilde{\mathbf{m}}_j = \mathbf{m}_j - \frac{\mathbf{m}_j^\top \mathbf{m}_4}{\|\mathbf{m}_4\|^2}\, \mathbf{m}_4, \quad j \in \{1, 2, 3\}, \tag{10}$$

and obtain the entries $\tilde{\mathbf{m}}_j^\top \tilde{\mathbf{m}}_l$ of $\boldsymbol{\Sigma}_{|t}$, a Gram matrix that is positive semi-definite by construction. As the shears in Eq. (7) have unit determinant, $\det \boldsymbol{\Sigma} = \det(\boldsymbol{\Sigma}_{|t})\,\Sigma^{tt}$, and with $\det \boldsymbol{\Sigma} = (s_1 s_2 s_3 s_4)^2$ the determinant of the slice is $\det(k(t)\,\boldsymbol{\Sigma}_{|t}) = k(t)^3\,(s_1 s_2 s_3 s_4)^2/\Sigma^{tt}$. It enters the spatial anti-aliasing (Sec. 3.2), where the mass of both density profiles is proportional to $\sigma\sqrt{\det \mathbf{A}}$ for a shape matrix $\mathbf{A}$, so rescaling the density by $\sqrt{\det(k(t)\,\boldsymbol{\Sigma}_{|t})/\det \hat{\boldsymbol{\Sigma}}_{|t}}$ preserves the mass under the dilation.

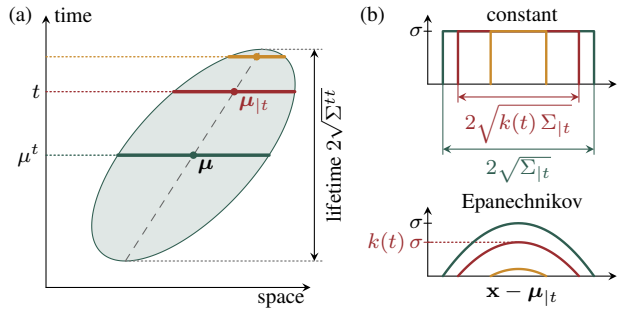

Figure 4. **Temporal slice of a 4D ellipsoid, shown with one spatial dimension.** (a) The slice at time t is centered at $\boldsymbol{\mu}_{|t}$, which moves with the velocity $\mathbf{v}$ along the dashed line, and its extent shrinks with $\sqrt{k(t)}$ until it vanishes at the ends of the lifetime. (b) Density across the three marked slices. A constant density keeps its value σ, whereas the peak of an Epanechnikov profile decreases to $k(t)\,\sigma$.

A.2. A Primitive Along a Ray

After slicing and anti-aliasing, a primitive is a 3D ellipsoid with center $\boldsymbol{\mu}_{|t}$, shape matrix $\hat{\boldsymbol{\Sigma}}_{|t}$, and density σ, to which the slice and the anti-aliasing contribute the factors of Sec. A.1. Let $\mathbf{L}$ be the Cholesky factor of $\hat{\boldsymbol{\Sigma}}_{|t} = \mathbf{L}\mathbf{L}^\top$. The map $\mathbf{x} \mapsto \mathbf{L}^{-1}(\mathbf{x} - \boldsymbol{\mu}_{|t})$ turns the ellipsoid into the unit sphere and the ray into $\tilde{\mathbf{o}} + \lambda\tilde{\mathbf{d}}$ with $\tilde{\mathbf{o}} = \mathbf{L}^{-1}(\mathbf{o} - \boldsymbol{\mu}_{|t})$ and $\tilde{\mathbf{d}} = \mathbf{L}^{-1}\mathbf{d}$, while λ remains the distance along the ray. The squared normalized radius along the ray is the parabola

$$r^2(\lambda) = \|\tilde{\mathbf{o}} + \lambda\tilde{\mathbf{d}}\|^2 = q + \|\tilde{\mathbf{d}}\|^2\,(\lambda - \bar{\lambda})^2, \tag{11}$$

with the minimum $q = \|\tilde{\mathbf{o}}\|^2 - (\tilde{\mathbf{o}}^\top\tilde{\mathbf{d}})^2/\|\tilde{\mathbf{d}}\|^2$ at $\bar{\lambda} = -\tilde{\mathbf{o}}^\top\tilde{\mathbf{d}}/\|\tilde{\mathbf{d}}\|^2$. The ray hits the ellipsoid if $q < 1$, and it

then enters and leaves at

$$\lambda^0 = \bar{\lambda} - h, \quad \lambda^1 = \bar{\lambda} + h, \quad h = \frac{\sqrt{1-q}}{\|\tilde{\mathbf{d}}\|}, \tag{12}$$

so $\bar{\lambda}$ is the midpoint and h the half length of its chord. With the position $w = (\lambda - \bar{\lambda})/h \in [-1, 1]$ on the chord, Eq. (11) reads $r^2 = q + (1-q)\,w^2$, and the density along the ray is

$$\rho(\lambda) = \begin{cases} \sigma & \text{(constant)}, \\ \sigma\,(1-q)\,(1-w^2) & \text{(Epanechnikov)}. \end{cases} \tag{13}$$

The Epanechnikov profile is thus a parabola along every ray, with the peak $\sigma\,(1-q)$ at the midpoint of the chord. Integrating Eq. (13) over the chord gives the optical depth

$$\tau = 2\,\sigma h \quad \text{and} \quad \tau = \tfrac{4}{3}\,\sigma\,(1-q)\,h \tag{14}$$

of the fragment for the two profiles. A fragment that is composited without overlap, as in the tail and in the reference modes without overlap, enters Eq. (2) with $\alpha = 1 - e^{-\tau}$, which is exact for a single fragment of either profile.

Density Parametrization. The same integrals define the density of a primitive from the optical depth τ of a reference chord through its center, where $q = 0$. EVER [48] uses the chord along the shortest semi-axis and caps its opacity $1 - e^{-\tau}$ at 0.99 with a scaled sigmoid. A chord along the longest semi-axis is longer by the ratio of the two semi-axes, and so is its optical depth, so an elongated primitive can still become opaque and block the gradients to everything behind it. We therefore use the chord along the longest semi-axis, where $2h$ is the chord length ℓ, and Eq. (14) gives $\sigma = \tau/\ell$ and $\sigma = {}^3\!/_2\,\tau/\ell$ (Sec. 3.2). We cap the opacity of this chord at 0.99 as well, and as no chord of a static primitive has a larger optical depth, the cap holds along every ray. We optimize $\log\tau$ and enforce the cap by clamping this parameter after every optimizer step instead of using a sigmoid.

A.3. Exact Core

The core of a pixel consists of K fragments with densities σ_i, colors $\mathbf{c}_i$, and chords $[\lambda_i^0, \lambda_i^1]$. Their $2K$ sorted endpoints divide the ray into segments, and within a segment the active set A of fragments that cover it does not change (Fig. 5). Consider the segment $[\lambda_a, \lambda_b]$ of length $\delta = \lambda_b - \lambda_a$ and let T_a be the transmittance at its start.

Constant Densities. The density on the segment is the constant $D = \sum_{i\in A}\sigma_i$, so the transmittance decays as $T(\lambda) = T_a\,e^{-D(\lambda-\lambda_a)}$, and the segment contributes

$$\Delta\mathbf{C} = \int_{\lambda_a}^{\lambda_b} \sum_{i\in A} \sigma_i\,\mathbf{c}_i\,T(\lambda)\,\mathrm{d}\lambda = T_a\left(1 - e^{-D\delta}\right)\bar{\mathbf{c}} \tag{15}$$

to Eq. (1), with the density-weighted mean color $\bar{\mathbf{c}} = \sum_{i\in A}\sigma_i\mathbf{c}_i/D$ of its active set. The transmittance $T_b = T_a\,e^{-D\delta}$ remains for the next segment. A segment thus behaves like a single fragment with the opacity $1 - e^{-D\delta}$ and the color $\bar{\mathbf{c}}$, and the core is the alpha blending of Eq. (2) over its $2K-1$ segments. The forward pass visits the sorted endpoints once and updates D and $\sum_{i\in A}\sigma_i\mathbf{c}_i$ whenever a fragment enters or leaves. No approximation is involved, and the rendered core matches a numerical reference up to rounding (Fig. 6). EVER [48] computes the same integral with a ray tracer, whose official implementation does not separate intersections at exactly the same distance along a ray. This distorts the gradients with respect to the endpoints of coinciding primitives, such as the copies that densification creates, whereas ours remain exact (Sec. A.4).

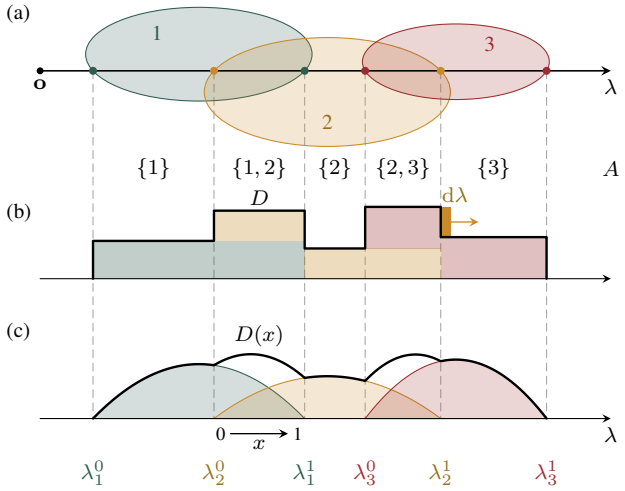

Figure 5. **Fragments along a ray.** (a) The chords of three overlapping primitives. Their sorted endpoints divide the ray into segments with a fixed active set A. (b) For constant densities, the aggregate density D is constant on every segment. Moving the exit point of a fragment adds a slab of its density, here for λ_2^1. (c) For Epanechnikov profiles, D is a quadratic in the coordinate x of every segment.

Epanechnikov Profiles. With the Epanechnikov profile, fragment i has the density $\rho_i(\lambda) = p_i\,(1 - w_i^2)$ along the ray, with the peak $p_i = \sigma_i\,(1-q_i)$ and the chord position $w_i = (\lambda - \bar{\lambda}_i)/h_i$ of Sec. A.2. The segments and their active sets are the same, but the density now varies within a segment (Fig. 5c). We describe the segment $[\lambda_a, \lambda_b]$ by the coordinate $x \in [0, 1]$ with $\lambda = \lambda_a + \delta\,x$. The chord position of an active fragment is linear in x, $w_i = w_i^a + x\,\delta/h_i$ with its value $w_i^a = (\lambda_a - \bar{\lambda}_i)/h_i$ at the start of the segment, so its density $p_i\,(1 - w_i^2)$ is the quadratic

$$\rho_i(x) = \rho_{i,0} + \rho_{i,1}\,x + \rho_{i,2}\,x^2, \tag{16}$$

whose coefficients $\rho_{i,n}$ follow by expanding the square. The aggregate density is then also a quadratic, $D(x) = \sum_{i\in A}\rho_i(x) = D_0 + D_1\,x + D_2\,x^2$ with $D_n = \sum_{i\in A}\rho_{i,n}$.

The optical depth between the start of the segment and x

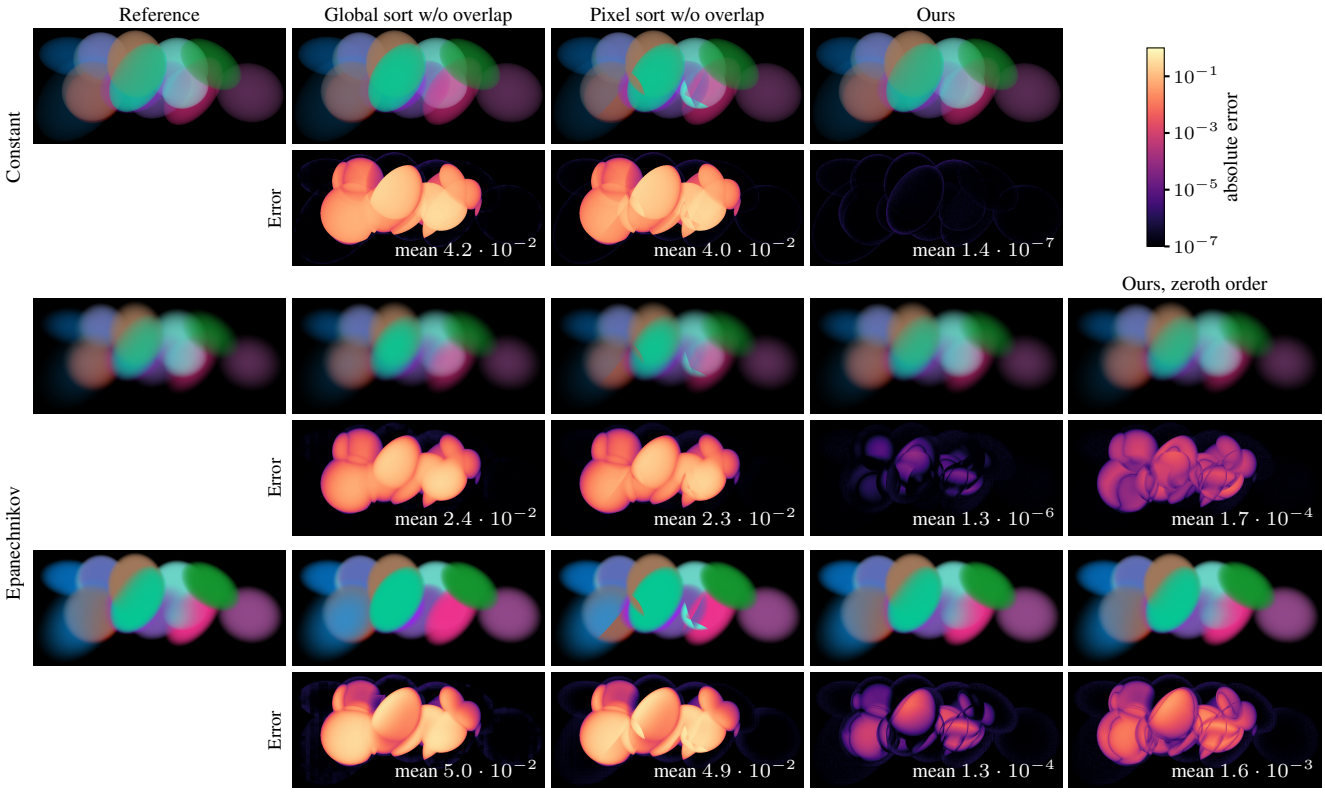

Figure 6. **Accuracy of the render modes.** Renderings and per-pixel errors against a float64 quadrature of Eq. (1) (left) on synthetic scenes of 16 overlapping ellipsoids with constant densities (top), with Epanechnikov profiles (middle), and with more opaque Epanechnikov profiles (bottom). The numbers give the mean over all covered pixels. Global sort and pixel sort without overlap, which is HT without overlap with an empty tail, composite whole fragments and ignore their overlap. Our core needs no expansion for constant densities and is exact up to rounding. For Epanechnikov profiles, its second-order expansion reduces the error of the zeroth order, which keeps only the linear part of the optical depth, by one to two orders of magnitude, and both errors grow with the opacity of the primitives.

is the integral of $\delta\, D$ up to x, the cubic

$$\tau(x) = \delta \left(D_0\, x + \tfrac{1}{2} D_1\, x^2 + \tfrac{1}{3} D_2\, x^3 \right). \tag{17}$$

The optical depth $\Delta\tau = \tau(1)$ of the whole segment is therefore exact, and so are its opacity $1 - e^{-\Delta\tau}$ and the transmittance $T_b = T_a\, e^{-\Delta\tau}$ behind it.

The segment contributes $\Delta\mathbf{C} = T_a \sum_{i \in A} u_i\, \mathbf{c}_i$ to Eq. (1) with the weights

$$u_i = \delta \int_0^1 e^{-\tau(x)}\, \rho_i(x)\, \mathrm{d}x, \tag{18}$$

which sum to the opacity of the segment. With the cubic $\tau(x)$ in the exponent, this integral has no closed form. We split the optical depth into its linear part and a remainder,

$$\begin{aligned} \tau(x) &= \Delta\tau\, x + \varepsilon(x), \\ \varepsilon(x) &= \beta\, (x^2 - x) + \gamma\, (x^3 - x), \end{aligned} \tag{19}$$

with $\beta = \tfrac{1}{2}\, \delta D_1$ and $\gamma = \tfrac{1}{3}\, \delta D_2$. The remainder vanishes at both ends of the segment and is small if the density varies little across it. We expand its exponential to second order,

$$e^{-\tau(x)} \approx e^{-\Delta\tau\, x} \left(1 - \varepsilon(x) + \tfrac{1}{2}\, \varepsilon(x)^2 \right). \tag{20}$$

The factor in parentheses is a polynomial $\sum_{m=0}^{6} s_m\, x^m$ of degree six with the coefficients $(s_0, \ldots, s_6) = (1,\ -a,\ \tfrac{1}{2}\, a^2 - \beta,\ a\beta - \gamma,\ \tfrac{1}{2}\, \beta^2 + a\gamma,\ \beta\gamma,\ \tfrac{1}{2}\, \gamma^2)$ and $a = -(\beta + \gamma)$. It is positive for every ε, so no weight can become negative, which a first-order expansion does not guarantee. Inserting Eqs. (16) and (20) into Eq. (18) leaves integrals of a monomial times an exponential,

$$u_i \approx \tilde{u}_i = \delta \sum_{n=0}^{2} \rho_{i,n}\, F_n, \quad F_n = \sum_{m=0}^{6} s_m\, E_{m+n}, \tag{21}$$

with the exponential moments $E_n = \int_0^1 x^n\, e^{-\Delta\tau\, x}\, \mathrm{d}x$. The three sums F_n are shared by all fragments of a segment. Finally, we normalize the weights so that they sum to the exact opacity,

$$\Delta\mathbf{C} = T_a \left(1 - e^{-\Delta\tau} \right) \bar{\mathbf{c}}, \quad \bar{\mathbf{c}} = \frac{\sum_{i \in A} \tilde{u}_i\, \mathbf{c}_i}{\sum_{i \in A} \tilde{u}_i}, \tag{22}$$

which has the form of Eq. (15). The expansion therefore only affects how the color of a segment is divided among its active fragments, whereas opacity and transmittance are exact. It is also exact for a segment with a single active fragment and for constant densities, where $\varepsilon = 0$.

The zeroth moment is $E_0 = (1 - e^{-\Delta\tau})/\Delta\tau$, and integration by parts links consecutive moments by $n\,E_{n-1} = \Delta\tau\,E_n + e^{-\Delta\tau}$. Solved for E_n, this recurrence runs upward from E_0, but it suffers from cancellation for small $\Delta\tau$. For $\Delta\tau < 8$, we therefore compute the highest moment from the series $E_N = e^{-\Delta\tau} \sum_{m\geq 0} \Delta\tau^m\, N!/(N+m+1)!$, whose terms are all positive, and run the recurrence downward. The forward pass needs the moments up to E_8 and the backward pass also E_9, since $\mathrm{d}E_n/\mathrm{d}\Delta\tau = -E_{n+1}$.

Figure 6 compares the render modes with a float64 quadrature of Eq. (1) on synthetic scenes of 16 overlapping ellipsoids, where every fragment is part of the core. With the second-order expansion, the pixel colors of the Epanechnikov core deviate by $1.3 \cdot 10^{-6}$ on average, compared to $1.7 \cdot 10^{-4}$ for the zeroth order, which keeps only the linear part of the optical depth. Both errors grow with the opacity of the primitives, as ε scales with their optical depths. The modes that ignore overlap are off by 2 to $5 \cdot 10^{-2}$ on the same scenes, and the core for constant densities stays at the rounding error of 10^{-7}. The second-order terms add 4% to the time of a training step in our tests.

Gaussians. For Gaussians, we evaluate every fragment at its maximal response along the ray, which gives an opacity and a depth, and alpha-composite the core in the order of these depths, ignoring overlap [21]. An exact core is possible with a similar approximation as for the Epanechnikov profile. The optical depth of a Gaussian along a ray, however, is an error function, which then has to be evaluated for every active fragment at both ends of every segment and also enters the weights of the fragments. These evaluations are expensive compared to the polynomials of the Epanechnikov profile, so we leave an exact core for Gaussians to future work.

A.4. Gradients

The core is an alpha blending of its segments (Sec. A.3), so the gradients of the pixel color $\mathbf{C}$ follow from the chain rule, first with respect to the segments, then to the fragments, and finally to the primitives.

Segments. We number the segments from front to back. Segment s has the optical depth $\Delta\tau_s$, the color $\bar{\mathbf{c}}_s$, and the transmittance T_s at its start, so it enters the pixel color with the blending weight $W_s = T_s\,(1 - e^{-\Delta\tau_s})$ and leaves the transmittance $T_{s+1} = T_s\,e^{-\Delta\tau_s}$. Let $\mathbf{C}_{>s} = \mathbf{C} - \sum_{j\leq s} W_j\,\bar{\mathbf{c}}_j$ be the part of the pixel color that originates behind the segment, including the tail and the background. As this part is proportional to T_{s+1}, the derivatives with respect to the color and the optical depth of the segment are

$$\frac{\partial \mathbf{C}}{\partial \bar{\mathbf{c}}_s} = W_s, \qquad \frac{\partial \mathbf{C}}{\partial \Delta\tau_s} = T_{s+1}\,\bar{\mathbf{c}}_s - \mathbf{C}_{>s}. \tag{23}$$

More optical depth adds the color of the segment at the transmittance behind it and attenuates everything that follows.

Constant Densities. The optical depth of a segment is $\Delta\tau_s = D_s\,\delta_s$, and its density D_s and color $\bar{\mathbf{c}}_s$ are those of Eq. (15) for its active set A_s. For a fragment i of this set,

$$\frac{\partial \bar{\mathbf{c}}_s}{\partial \mathbf{c}_i} = \frac{\sigma_i}{D_s}, \quad \frac{\partial \bar{\mathbf{c}}_s}{\partial \sigma_i} = \frac{\mathbf{c}_i - \bar{\mathbf{c}}_s}{D_s}, \quad \frac{\partial \Delta\tau_s}{\partial \sigma_i} = \delta_s. \tag{24}$$

Summing over the segments S_i on which the fragment is active gives its blending weight and the gradient of its density,

$$\frac{\partial \mathbf{C}}{\partial \mathbf{c}_i} = \sum_{s\in S_i} W_s\,\frac{\sigma_i}{D_s}, \tag{25}$$

$$\frac{\partial \mathbf{C}}{\partial \sigma_i} = \sum_{s\in S_i} \Big(\frac{W_s}{D_s}\,(\mathbf{c}_i - \bar{\mathbf{c}}_s) + \delta_s\,\frac{\partial \mathbf{C}}{\partial \Delta\tau_s}\Big). \tag{26}$$

The first term of Eq. (26) changes the color of a segment and the second its opacity.

The endpoints of a fragment change the lengths of the segments. Moving the exit point λ_i^1 outward by $\mathrm{d}\lambda$ adds a slab of the fragment (Fig. 5b), which to first order is a segment of its own with the optical depth $\sigma_i\,\mathrm{d}\lambda$ and the color $\mathbf{c}_i$, and moving the entry point λ_i^0 outward removes one. With Eq. (23), this gives

$$\begin{aligned} \frac{\partial \mathbf{C}}{\partial \lambda_i^1} &= \sigma_i\,\big(T(\lambda_i^1)\,\mathbf{c}_i - \mathbf{C}_{>\lambda_i^1}\big), \\ \frac{\partial \mathbf{C}}{\partial \lambda_i^0} &= -\sigma_i\,\big(T(\lambda_i^0)\,\mathbf{c}_i - \mathbf{C}_{>\lambda_i^0}\big), \end{aligned} \tag{27}$$

where $T(\lambda)$ is the transmittance at λ and $\mathbf{C}_{>\lambda}$ the part of the pixel color that originates behind it. The same result follows from the lengths of the two segments that share the endpoint, as these segments differ only by the fragment. Equation (27) also holds if the endpoints of several fragments coincide, as for the copies that densification creates, whereas differentiating the lengths of the sorted segments would attribute a shared endpoint to all of these fragments. A fragment that is composited as a whole by Eq. (2) is a segment of its own, so Eq. (23) applies with its optical depth τ and its color $\mathbf{c}_i$.

Epanechnikov Profiles. The parabola of Eq. (13) vanishes at both ends of the chord, so the endpoints contribute no slab, and the geometry acts through the peak p_i, the midpoint $\bar{\lambda}_i$, and the half length h_i instead. Without overlap, the optical depth $\tau = {}^4\!/_3\,p_i\,h_i$ of Eq. (14) depends on p_i and h_i only. In the core, we differentiate the rule of Eqs. (21) and (22) rather than the exact integral, so that the gradients are consistent with the rendered image. The rule is a rational function of the

coefficients $\rho_{i,n}$ and of the moments E_n, whose derivatives are again moments, so all of its derivatives have a closed form. They are propagated to p_i, $\bar{\lambda}_i$, and h_i through Eq. (16).

From Fragments to Primitives. The remaining steps apply the chain rule. The fragment quantities depend on $\mathbf{L}$ and $\boldsymbol{\mu}_{|t}$ through the ray in the frame of the unit sphere (Sec. A.2), and these depend on the parameters of the 4D ellipsoid through the slice (Sec. A.1). The half length h of Eq. (12) has an unbounded derivative for grazing rays, and so has the extent of a slice at the ends of the lifetime. We therefore skip rays with $q \geq 0.99$ and slices with $k(t) \leq 0.01$.

A.5. Splitting a Primitive into Copies

Densification replaces a source by n copies, the source itself and $n-1$ new primitives, which share its optical depth τ (Sec. 3.4). Every copy receives τ/n and keeps the semi-axes of the source, and therefore its chord length ℓ. By Sec. A.2, the density is proportional to τ/ℓ for both profiles, so every copy has the density σ/n. As densities add up in Eq. (1), n coinciding copies have the density of the source at every point, and the split changes neither the transmittance nor the color of any ray. The copies are then displaced within the extent of the source, which redistributes this density. The opacity of a Gaussian instead scales its response at every pixel, so the same split preserves the transmittance only at the center of the Gaussian, which is why 3DGS-MCMC [33] also adjusts the scales.

B. Implementation Details

B.1. Rasterizer

Both primitive types and all render modes are implemented in one CUDA extension per primitive type, which shares its structure with Faster-GS [23]: a per-primitive preprocess kernel, instance creation with the exact tile test, a radix sort of the instances for the modes that order them, the blend kernel, and a fused Adam optimizer. The preprocess kernel is compiled at runtime with NVRTC [55], so that the appearance model, the primitive type, and the render mode are compile-time constants, and the fused appearance evaluation of Hahlbohm *et al*. [22] is inlined into it. The global sort uses 16×16 pixel tiles and the per-bucket backward pass of Mallick *et al*. [50]. The hybrid modes keep the K core fragments of every pixel in registers and use 8×8 tiles. For the sorted tail, the sort key of a primitive in a tile is a 16-bit logarithmic depth between the near and the far plane. As in HTGS [21], a fragment of a Gaussian enters the core only with an opacity of at least 0.05, and fainter fragments are composited with the tail. Our ellipsoid modes do not make this approximation. The core of the hybrid mode with sorted tail is also kept in registers and its per-slot gradient information is written to per-pixel buffers in the forward pass, together with a per-bucket bit mask that marks the core fragments of every pixel. The backward pass replays the instances of a tile from bucket checkpoints like the global sort, takes the gradient of a core fragment from the stored slot instead of recomputing it, and treats the remaining fragments as tail. Fragments that straddle the near or far plane are clipped exactly to the planes, with the gradient of the clipped endpoint set to zero, rather than dropped. The near-plane rule and the 3D filter run as fused kernels.

B.2. Training Details

For all settings that the main paper does not state, our default configuration uses the values of Hahlbohm *et al*. [22], with two exceptions. The learning rate of the opacity is 0.025, and the weight of the opacity regularizer is 0.001. The 60-frame windows of SelfCap are trained with one view per iteration. Static scenes are trained with one view per iteration as well and are initialized from the sparse SfM points of the dataset. Unless stated otherwise, all settings are shared by static and dynamic scenes, both primitive types, and all render modes.

Initialization. The dense initialization (Sec. 3.4) assigns every point the standard deviations of a Gaussian, and the semi-axes of an ellipsoid are 3.33 times these standard deviations, which is the extent at which a Gaussian falls below the opacity threshold of $1/255$ of 3DGS [32]. It triangulates 1000 of the moving pixels per camera and keyframe. For a moving point, the temporal standard deviation is half the keyframe interval and the spatial one is half the root mean squared distance to its three nearest neighbors, so its slice has semi-axes of 1.66 times this distance and its lifetime spans 3.33 keyframe intervals. A static point has a spatial standard deviation of $5 \cdot 10^{-4}$ times its distance to the nearest training camera [36], a temporal mean drawn uniformly from $[-0.1, 1.1]$ [81], and a temporal variance of 0.2 [81]. All primitives start with an opacity of 0.1. If the initialization exceeds the budget of one million primitives, we keep a random subset of the static points and scale them by the square root of the ratio of the point counts, so that the model starts at the budget and does not grow. This applies to all scenes of Neural 3D Video, where the initialization holds 1.1 to 1.4 million points and about 50 to 90% of the static points are kept, whereas the SelfCap windows start with 0.4 to 0.9 million primitives.

Densification Schedule. The schedule for growing the model and relocating primitives follows Hofer *et al*. [26], to whom we refer for details. Copies of an elongated source are placed side by side along its longest axis [9] (Sec. 3.3), which for a moving source also shifts them in time. Every copy is 10% longer than an even division of this axis, so that neighboring copies overlap.

Bounds for Moving Primitives. The opacity cap and the temporal bound of Sec. 3.2 are enforced on the semi-axes of the 4D ellipsoid. The cap takes twice the longest of the three

Table 4. **CUDA rasterizer and WebGL viewer.** FPS and PSNR of our models on flame salmon (Neural 3D Video [41]) and flowers (Mip-NeRF 360 [1]). The WebGL viewer renders one stochastic sample per frame on a MacBook Pro and an iPhone 18 Pro.

		Flame salmon		Flowers	
	Renderer	FPS↑	PSNR↑	FPS↑	PSNR↑
Gaussian	CUDA, RTX 5090	359	30.58	236	22.15
	CUDA, RTX 4090	279		179	
	WebGL, M1 Max	73	29.16	47	21.42
	WebGL, A20 Pro	29		22	
Constant	CUDA, RTX 5090	454	30.06	413	21.99
	CUDA, RTX 4090	344		315	
	WebGL, M1 Max	112	29.17	56	21.28
	WebGL, A20 Pro	53		25	
Epanechnikov	CUDA, RTX 5090	379	30.44	317	22.06
	CUDA, RTX 4090	257		219	
	WebGL, M1 Max	52	29.64	39	21.40
	WebGL, A20 Pro	20		17	

spatial semi-axes as the chord length ℓ, and the temporal bound keeps the temporal semi-axis at one frame or more. For a static primitive, these are the longest chord of every slice and half the lifetime, so both bounds hold as stated. A moving primitive is tilted in space-time (Fig. 4a), so its temporal semi-axis also extends in space and its spatial semi-axes also extend in time. Its slice can then be longer than ℓ along the direction of motion, so that the opacity along this direction can exceed the cap, and its lifetime $2\sqrt{\Sigma^{tt}}$ can be shorter than two frames. We found neither to be an issue in practice.

B.3. Inference Rendering

CUDA Rasterizer. The inference kernels keep the core of the hybrid modes without overlap in half precision (premultiplied color and opacity as pairs of 16-bit floats, as in HTGS [21]), which saves registers and lets the GPU run more threads concurrently. In all render modes, they only compute the image and omit what the forward pass of training stores for the backward pass (Sec. B.1), *i.e.*, the bucket checkpoints and the per-pixel gradient information of the core, which avoids the memory traffic of these buffers.

WebGL Viewer. Our WebGL viewer builds on the web viewer of Hahlbohm *et al*. [22], which is based on Three.js, and extends it to the ray-based evaluation of both primitive types in 3D and 4D. An exported model stores 24 bytes of geometry per 4D primitive in a compact layout similar to Spark [73], namely a half-precision spatial mean, a 16-bit temporal mean, 8-bit logarithmic codes for the four semi-axes, two 24-bit quaternions for the 4D rotation, and the 8-bit base color and opacity. Whenever the time changes, a slice pass evaluates one texel per primitive in a fragment shader and writes the center, the shape matrix, and the density scale of its slice (Sec. A.1) to floating-point textures. A second pass of this kind evaluates the view-dependent colors whenever the camera moves. The render pass draws one silhouette quad per primitive [20], built from the Cholesky factor of the shape matrix of the slice. As in Sec. A.2, the ellipsoid is the unit sphere in this frame, so the fragment shader obtains the chord of its ray and the optical depth of Eq. (14) from a few dot products, with the anti-aliasing, the grazing cutoff, and the clipping to the near and far plane of the CUDA rasterizer. As the slices change with every frame of a video, the viewer does not sort 4D primitives and composites them with alpha-corrected stochastic transparency [15] in all render modes, whereas 3D models trained with the global sort are sorted on the CPU and blended back to front. Per sample, a first pass draws eight stratified survival decisions for every fragment and keeps the nearest surviving depth of each draw. A second pass accumulates the colors of all fragments, weighted by their opacity and by the share of the eight depths that do not lie in front of them, and a resolve scales the result to the exact coverage of the pixel. For models trained with overlap, the depths are drawn from the distribution of the first scattering event inside the chord, *i.e.*, with free-flight distance sampling as in StochasticSplats [34], which accounts for the overlap of the primitives. The image is averaged over the samples while the view is still, and under camera motion and playback, a temporal reprojection as in SteadySplats [72], extended by the displacement of the slice centers between frames, continues the average. The viewer thus renders the per-pixel order without the core and tail of our render mode, which differs only where more than K fragments overlap before a pixel saturates.

In Tab. 4, the CUDA rasterizer renders at 179 to 454 FPS. With one sample per frame, the WebGL viewer reaches 39 to 112 FPS on a laptop and 17 to 53 FPS on a phone. Its PSNR, which is nearly identical on both devices, is 0.7 to 1.4 dB below the CUDA rasterizer, and averaging more samples recovers part of this difference.

C. Evaluation Setup

C.1. Frame Extraction

Neural 3D Video [41] and SelfCap [71] provide videos, and every resampling or lossy compression step between a video and the training images removes detail, which makes the images easier to fit and inflates the metrics, while the reconstructions contain less high-frequency detail. We therefore decode every video once, resample every frame only once with a Lanczos filter, and store it as a lossless PNG file. For Neural 3D Video, this single step is the downscaling to half resolution. For SelfCap, the frames also have to be undistorted, so the single step maps every decoded frame directly to the pixel grid of a COLMAP undistortion followed by a downscaling by a factor of two, with no downscaling for the bike windows. This step resamples the luma and chroma planes of the video separately, so that the chroma, which



Figure 7. **Moving-object masks.** We compute mPSNR and mLPIPS inside masks that cover every moving object as a whole, shown for the held-out cameras of Neural 3D Video [41] (top) and SelfCap [71] (bottom) with other content darkened. The masks of Neural 3D Video are obtained automatically from optical flow [68] and SAM 2 [61], and those of SelfCap semi-automatically with Cutie [7].

the videos store at half resolution, is never upsampled first. All methods, *i.e.*, ours and the baselines, load the extracted frames without further resampling and thus see identical images. We also found the videos of SelfCap to shake by a few pixels from frame to frame, most strongly in the dance sequences. Our attempts to remove this shake, namely aligning every frame with a homography or a translation and fitting a camera pose per frame, did not help, so all methods use the uncorrected frames.

C.2. Photometric Calibration

We use PPISP [10], which models only the photometric effects of a physical camera and is thus better constrained than an affine color correction per camera [40]. It describes a camera with a vignetting, a response curve, and an exposure and a color correction per frame, which a controller predicts from the rendering for novel views. In a rig, every camera has its own processing, and the cameras differ far more from each other than a camera changes over time. With one model for all cameras, as in the standard setup for a single camera, the per-frame exposure and color corrections absorb these differences and are almost constant per camera. The controller cannot infer such an offset from a rendering and outputs a constant for the held-out camera that misses its offset. We therefore give every training camera its own vignetting and response curve, keep the exposure and the color correction per frame, and train no controller. In early experiments, we found that these per-camera models also improve the reconstruction itself.

Held-Out Camera. The held-out camera then has no parameters. For all results on Neural 3D Video and SelfCap in the main paper, we therefore calibrate it for every method, *i.e.*, we freeze the scene and fit a fresh single-camera model with per-frame exposure and color, one vignetting, and one response curve to its frames, with the photometric loss and regularizers of training for 5k steps. This calibration separates the quality of the reconstruction from the error of guessing the response of an unseen camera. Without calibration, we use the per-parameter median over the training cameras, and Tabs. 7 and 8 compare both for every scene and method. In early experiments, we found the median to be a better guess than the controller of a single-camera model or the three nearest training cameras.

Table 5. **Core size and core regularization.** Means over the scenes of Neural 3D Video [41] for 4D ellipsoids with the Epanechnikov profile in our mode, with the metrics of Tab. 2. The upper block varies the number K of core fragments, and the lower block removes one of the measures of Sec. 3.3 at a time.

Configuration	PSNR$^\uparrow$	SSIM$^\uparrow$	LPIPS$^\downarrow$	$\alpha_{\text{core}}{}^\uparrow$	Time$^\downarrow$
Ours ($K = 16$)	33.70	0.950	0.175	0.67	1h04m
$K = 32$	33.75	0.950	0.175	0.93	2h33m
$K = 64$	33.69	0.950	0.176	0.99	6h39m
w/o distortion loss	33.77	0.950	0.175	0.59	55m
w/o long-axis split	33.78	0.950	0.175	0.57	1h05m
w/o color variance loss	33.68	0.950	0.175	0.58	1h03m
opacity regularizer 0.01	33.83	0.950	0.175	0.41	1h06m
w/o core regularization	33.56	0.949	0.176	0.23	58m

Table 6. **Anti-aliasing.** Mean PSNR over the scenes of Neural 3D Video [41] for the held-out camera at the training resolution (1×), at the native resolution of the videos (2×), and at half the training resolution (0.5×). We remove the density rescaling, the lower bound of the semi-axes (3D filter), and the lower bound of the temporal extent (temporal floor) of Sec. 3.2 one at a time and all at once, which is the configuration w/o AA of Tab. 2.

Configuration	1×	2×	0.5×
Ours	33.70	33.00	33.58
w/o density rescaling	33.84	32.19	31.54
w/o 3D filter	33.88	33.10	33.78
w/o temporal floor	33.82	33.11	33.68
w/o all three (w/o AA in Tab. 2)	33.80	31.43	31.34

C.3. Moving-Object Masks

Our evaluation uses dynamic region masks that cover every moving object as a whole (Fig. 7), *e.g.*, a person together with the objects they hold while only an arm moves. Thresholding the change of every pixel cannot provide such masks, as it only marks the pixels that change. For Neural 3D Video, we therefore segment object instances in the videos of the held-out camera and decide per instance whether it moves, without any manual input. Optical flow [68] between frames five apart seeds SAM 2 [61] instances at keyframes, and

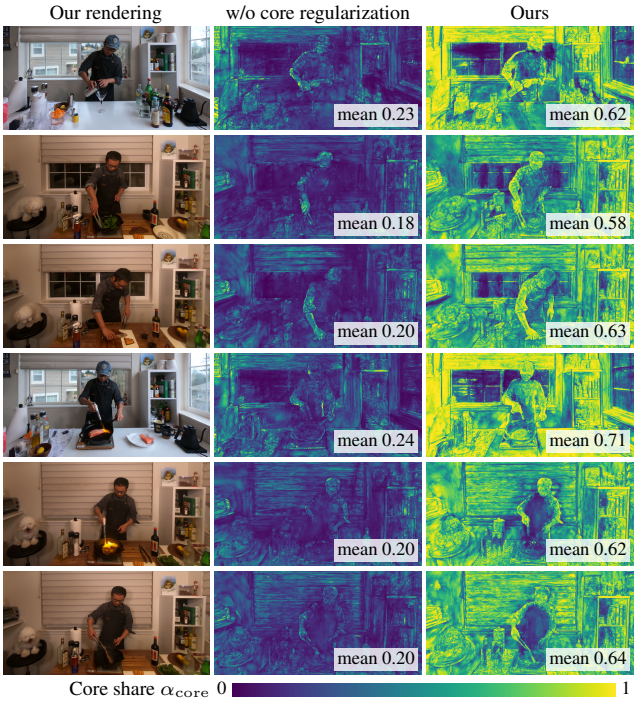

Figure 8. **Core share.** Per-pixel core share α_{core} and its mean for one test view per scene of Neural 3D Video [41], for the last and the first row of Tab. 5, *i.e.*, without and with core regularization.

each instance is tracked over the sequence. An instance is kept if it ever moves in place or changes its silhouette, judged by robust change statistics against a per-object background estimate, and is then masked in every frame. The final masks cover 6 to 10% of the frame and are closed and dilated by a few pixels in space and time, so that the boundary band, where reconstruction errors concentrate, lies inside the region. For SelfCap, the automatic masks were less reliable, so we annotated the masks of the held-out camera semi-automatically with Cutie [7]. Following DyCheck [17], mPSNR is computed over the masked pixels and mLPIPS on the masked image with AlexNet.

D. Further Experiments and Results

D.1. Core Share

Table 5 extends the comparison of Tab. 2 to larger cores and to the four individual measures of our core regularization (Sec. 3.3), *i.e.*, the distortion loss, the long-axis split, the color variance loss, and the weight of the opacity regularizer, which we lower from the 0.01 of Hofer *et al*. [26] to 0.001. The last row reverts all four measures, which is the configuration without core regularization of Tab. 2. Compared to it, our default model raises the core share (Fig. 8) at slightly higher PSNR and training time.

D.2. Anti-Aliasing

Table 6 ablates the density rescaling, the 3D filter, and the temporal floor at three resolutions. All configurations perform similarly at the training resolution. At the others, the PSNR drops clearly without the density rescaling. The 3D filter and the temporal floor target spatiotemporal extrapolation from the training views, *e.g.*, close-ups or slow motion, and removing either does not lower the PSNR here.

D.3. Qualitative Results

Figures 9 and 10 compare all methods of Tab. 1 on test views of Neural 3D Video and SelfCap. The methods differ most on fast-moving content, such as hands, hair, and the dog.

D.4. Per-Scene Results

Tables 7 and 8 list the per-scene numbers behind Tab. 1.

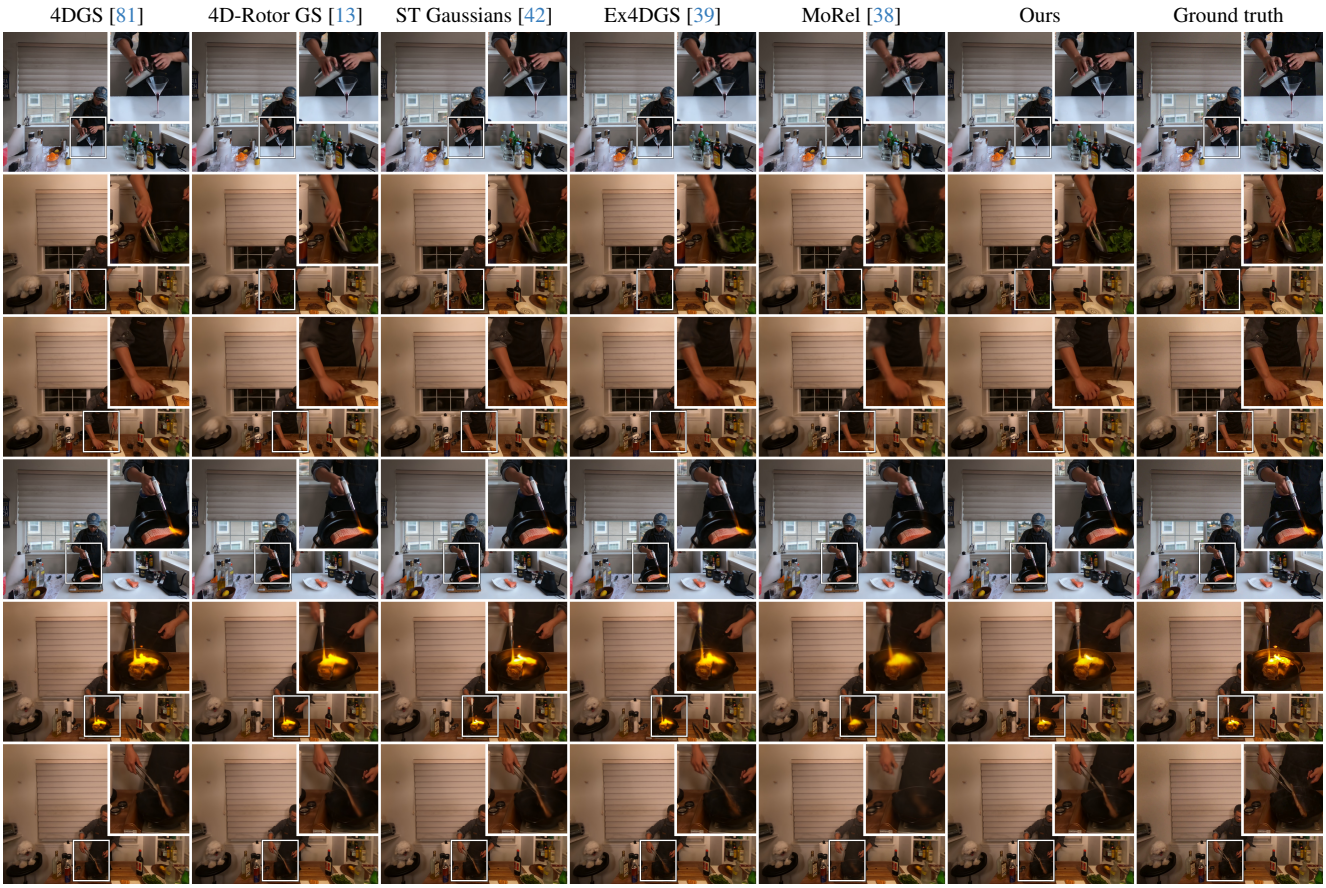

Figure 9. Qualitative comparison on test views of Neural 3D Video [41]. Best viewed digitally and zoomed in.

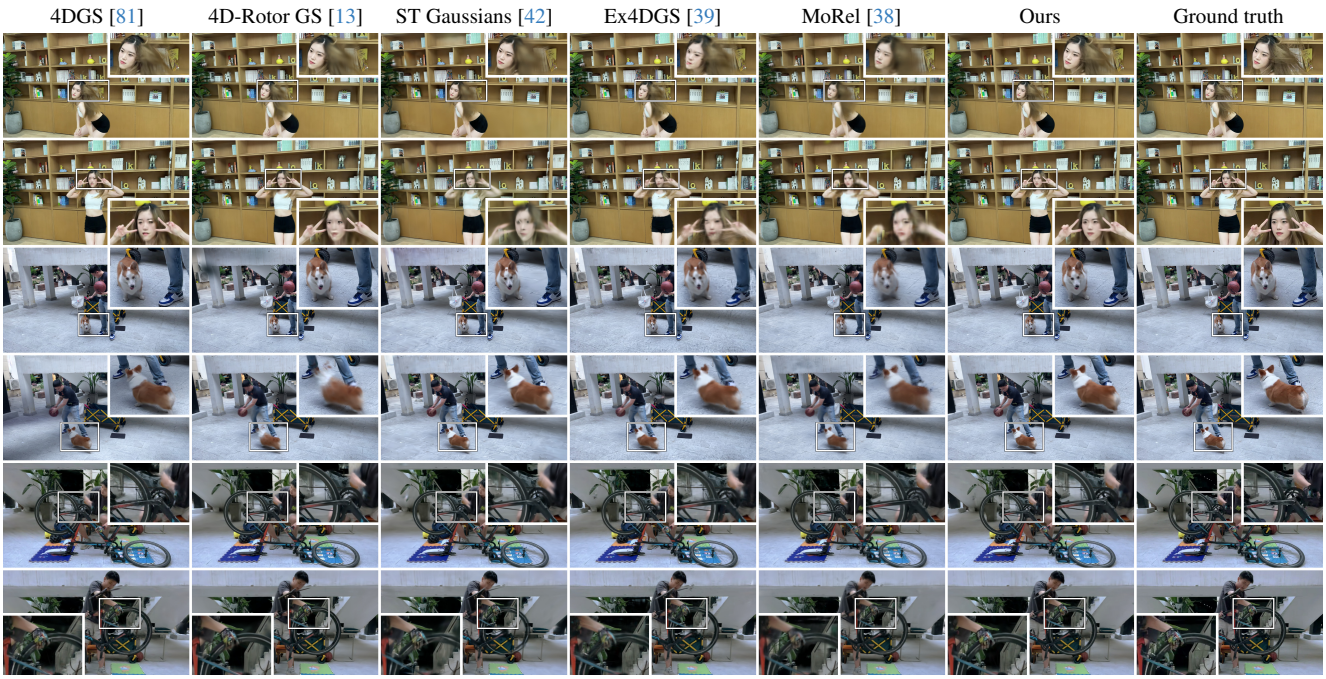

Figure 10. Qualitative comparison on test views of SelfCap [71]. Best viewed digitally and zoomed in.

Table 7. Per-scene results on the Neural 3D Video dataset [41] under our protocol, with the methods of Tab. 1. We report mPSNR and mLPIPS, PSNR with the median response of the rig (no calib.), and calibrated PSNR, SSIM, and LPIPS. Time, which includes the initialization as in Tab. 1, and FPS are measured on an RTX 4090. The best and second best results of every scene are color coded.

Scene	Method	mPSNR↑	mLPIPS↓	No calib.↑	PSNR↑	SSIM↑	LPIPS↓	Time↓	FPS↑
Coffee martini	4DGS [81]	31.05	0.059	28.79	28.84	0.912	0.203	2h35m	81
	4D-Rotor GS [13]	29.25	0.090	28.42	28.52	0.909	0.207	2h43m	5
	ST Gaussians [42]	30.79	0.053	27.39	28.80	0.912	0.205	1h21m	477
	Ex4DGS [39]	27.47	0.137	29.16	29.20	0.911	0.203	58m	75
	MoRel [38]	29.15	0.083	29.44	29.73	0.923	0.182	1h40m	35
	Ours	31.92	0.039	29.25	29.69	0.923	0.183	1h16m	255
Cook spinach	4DGS [81]	31.37	0.102	33.15	34.25	0.954	0.198	2h19m	139
	4D-Rotor GS [13]	29.71	0.151	32.70	33.47	0.950	0.203	2h50m	9
	ST Gaussians [42]	30.85	0.096	31.45	34.44	0.956	0.195	2h55m	522
	Ex4DGS [39]	29.31	0.164	32.29	33.10	0.949	0.195	57m	121
	MoRel [38]	28.41	0.176	32.46	33.43	0.953	0.190	1h32m	46
	Ours	31.27	0.078	33.54	34.67	0.958	0.178	1h17m	251
Cut roasted beef	4DGS [81]	34.39	0.082	33.26	35.15	0.958	0.193	2h17m	136
	4D-Rotor GS [13]	32.23	0.130	33.11	34.16	0.953	0.199	2h47m	9
	ST Gaussians [42]	34.45	0.074	29.14	35.07	0.958	0.193	2h51m	540
	Ex4DGS [39]	31.77	0.141	32.84	33.93	0.952	0.195	54m	131
	MoRel [38]	31.14	0.153	33.09	34.10	0.954	0.192	1h31m	46
	Ours	35.01	0.054	33.81	35.39	0.961	0.174	1h16m	253
Flame salmon	4DGS [81]	28.15	0.075	29.35	29.47	0.917	0.194	2h31m	52
	4D-Rotor GS [13]	27.63	0.096	28.97	29.11	0.917	0.197	2h43m	6
	ST Gaussians [42]	28.33	0.070	26.96	29.77	0.923	0.191	1h40m	422
	Ex4DGS [39]	26.85	0.128	29.23	29.40	0.916	0.190	56m	75
	MoRel [38]	26.69	0.093	29.89	30.29	0.928	0.172	1h41m	35
	Ours	29.09	0.054	30.26	30.44	0.930	0.170	1h14m	257
Flame steak	4DGS [81]	32.03	0.067	34.24	35.23	0.961	0.183	2h20m	143
	4D-Rotor GS [13]	31.07	0.097	33.79	34.57	0.958	0.190	2h46m	9
	ST Gaussians [42]	32.24	0.058	29.69	35.92	0.964	0.182	2h53m	537
	Ex4DGS [39]	30.10	0.110	32.88	34.27	0.957	0.184	58m	147
	MoRel [38]	29.75	0.107	33.43	34.66	0.961	0.177	1h33m	45
	Ours	32.20	0.048	35.04	35.70	0.964	0.170	1h17m	253
Sear steak	4DGS [81]	35.38	0.064	33.99	35.75	0.962	0.186	2h18m	136
	4D-Rotor GS [13]	33.92	0.091	33.66	35.09	0.959	0.191	2h47m	9
	ST Gaussians [42]	35.80	0.054	30.44	36.61	0.965	0.183	2h54m	545
	Ex4DGS [39]	34.12	0.087	32.41	34.47	0.956	0.186	55m	147
	MoRel [38]	32.37	0.102	32.64	34.34	0.956	0.190	1h29m	55
	Ours	36.16	0.043	34.98	36.33	0.965	0.173	1h18m	243

Table 8. Per-scene results on the SelfCap dataset [71] under our protocol, with the columns and color coding of Tab. 7.

Scene	Method	mPSNR↑	mLPIPS↓	No calib.↑	PSNR↑	SSIM↑	LPIPS↓	Time↓	FPS↑
Dance 1	4DGS [81]	26.44	0.191	22.68	27.50	0.912	0.198	2h59m	103
	4D-Rotor GS [13]	25.46	0.236	23.35	27.50	0.919	0.174	6h40m	3
	ST Gaussians [42]	23.71	0.271	20.51	25.53	0.896	0.254	40m	354
	Ex4DGS [39]	24.12	0.275	22.57	26.97	0.909	0.185	1h16m	96
	MoRel [38]	22.62	0.329	23.77	26.16	0.916	0.186	2h50m	39
	Ours	26.73	0.167	22.12	27.76	0.919	0.176	34m	205
Dance 2	4DGS [81]	23.54	0.225	22.42	26.96	0.908	0.206	2h58m	105
	4D-Rotor GS [13]	23.96	0.251	22.51	27.00	0.915	0.179	6h22m	3
	ST Gaussians [42]	22.04	0.329	20.55	25.50	0.895	0.248	40m	368
	Ex4DGS [39]	21.36	0.324	22.06	26.20	0.904	0.188	1h14m	101
	MoRel [38]	20.60	0.356	23.05	25.68	0.905	0.203	2h46m	36
	Ours	24.46	0.202	21.75	26.98	0.913	0.180	33m	207
Corgi 1	4DGS [81]	25.05	0.115	26.46	27.07	0.853	0.201	2h57m	94
	4D-Rotor GS [13]	24.66	0.143	21.83	22.28	0.886	0.187	9h15m	2
	ST Gaussians [42]	24.74	0.158	24.48	26.50	0.800	0.336	37m	481
	Ex4DGS [39]	22.51	0.301	25.08	26.07	0.789	0.300	1h03m	203
	MoRel [38]	23.46	0.234	26.02	27.89	0.856	0.249	2h39m	49
	Ours	25.90	0.112	26.73	27.98	0.885	0.192	33m	206
Corgi 2	4DGS [81]	20.85	0.247	19.09	20.69	0.842	0.217	2h58m	94
	4D-Rotor GS [13]	20.87	0.313	27.15	27.87	0.880	0.194	8h10m	2
	ST Gaussians [42]	21.84	0.273	24.62	26.66	0.798	0.341	39m	469
	Ex4DGS [39]	21.30	0.342	26.05	26.72	0.800	0.293	1h04m	194
	MoRel [38]	20.00	0.370	25.55	26.44	0.848	0.260	2h36m	50
	Ours	22.17	0.244	26.85	27.74	0.876	0.203	33m	206
Bike 1	4DGS [81]	25.70	0.095	26.22	27.07	0.869	0.249	1h42m	175
	4D-Rotor GS [13]	25.87	0.093	26.85	27.87	0.893	0.207	2h30m	9
	ST Gaussians [42]	25.67	0.104	26.43	27.91	0.888	0.248	25m	674
	Ex4DGS [39]	25.02	0.128	26.55	27.43	0.865	0.256	44m	253
	MoRel [38]	24.37	0.143	26.84	27.88	0.881	0.235	1h47m	57
	Ours	26.09	0.080	27.27	27.99	0.890	0.208	22m	271
Bike 2	4DGS [81]	25.22	0.125	25.97	27.40	0.882	0.241	1h47m	159
	4D-Rotor GS [13]	25.87	0.121	26.44	28.53	0.903	0.209	1h51m	13
	ST Gaussians [42]	23.02	0.210	25.41	26.75	0.871	0.272	26m	665
	Ex4DGS [39]	24.51	0.175	25.79	27.45	0.870	0.259	44m	269
	MoRel [38]	24.03	0.204	25.78	27.56	0.890	0.219	1h43m	59
	Ours	25.99	0.104	26.66	28.15	0.900	0.204	22m	284