

Gaussian Representations for Video

Supplementary Material

In this supplementary material, we show visualizations of GaRV’s decoding power in different applications (§1), analyze method choices (§2), provide the derivation of slicing gradients (§3), describe evaluation metrics (§4), and discuss alternative methods (§5). The supplemental zip file also contains sample videos decoded from a GaRV representation. Please view the ‘index.html’ file in a web browser or look at individual files inside the ‘video’ folder.

1. Visualizations

1.1. Frame Decoding

We show sample frames for all 7 UVG videos for HNeRV [5] and GaRV in Figure 1. GaRV is better at preserving low-frequency details such as posts, while HNeRV [5] is better at preserving high-frequency details such as water droplets. We also show sample frames and errors for DAVIS videos in Figure 2 to highlight what types of areas GaRV represents well.

1.2. Spatially Variable Resolution

GaRV can easily control where to spend the bitrate by combining Gaussians from two different encodings. This can enable low-bit rate high-resolution experiences by dynamically changing bitrate spatially at stream time. Figure 3 demonstrates an example of selecting Gaussians based on a 3D cylinder. This could be extended to follow saliency maps or user gaze instead of an arbitrary path. No other video representation has the capability of this level of control. The closest would be at encode time downsampling/blurring the unimportant regions. However, unique to GaRV is the ability to explicitly set the bitrate per region.

1.3. Direct Crops

Unlike existing video representations, GaRV can directly decode any arbitrary cropped view of a video. Figure 4 shows an example on the ‘ShakeNDry’ UVG sequence.

1.4. Line Scans

By performing a slice over x or y instead of t , we can directly render a line scan. Figure 5 shows an example of a line scan at the middle y value.

1.5. Denoising

As described in §4.6 of the main paper, GaRV has some built-in denoising capabilities. Figure 6 shows sample noisy and denoised frames from both GaRV and HNeRV [5].

1.6. Frame Interpolation

As described in §4.5 of the main paper, GaRV can interpolate frames by using a t value between two supervised frames. Figure 7 shows rendering a frame at an unsupervised timestep with the frames immediately before / after.

2. Ablation

2.1. 3D vs 2D Gaussians

We compare leveraging GaRV’s temporal information sharing through time slicing to simply having a unique set of 2D Gaussians for each frame. Based on GaussianImage [60], we fit 2D Gaussians to each of the 3900 frames. Table 1 shows temporal sharing from 3D Gaussians is significantly more efficient than using frame-wise 2D Gaussians. Figure 8 provides sample frames for 2D vs 3D representations.

Method	Parameters↓	PSNR↑	FPS↑	VRAM↓
2D	10M	21.78	1000	200 MB
	<u>150M</u>	<u>31.27</u>	<u>500</u>	600 MB
3D	10M	32.23	<u>500</u>	<u>160 MB</u>

Table 1. **3D vs 2D Gaussian comparison.** Using the same representation capacity, 3D Gaussians have significantly better quality at a cost to FPS. Scaling the 2D representation 15x is required to reach similar performance to 3D Gaussians.

2.2. Learning Rate

Learning rate selection is important for both convergence speed and quality. Global learning rates too high (0.01) and too small (0.0001) converge to a sub-optimal solution. Figure 9a shows how 0.005 finds a middle ground for reaching an acceptable frame quality rate in fewer iterations. Unlike neural counterparts, the parameter range differs between each Gaussian attribute, so leveraging per-attribute learning rates can significantly improve encoding time. Figure 9b shows how the scale’s learning rate is the largest factor for encoding time. We leverage smaller learning rates for position and color, and larger learning rates for scale and rotation, speeding up GaRV’s encoding time.

2.3. Color

Activation. Since each pixel’s color must be on the range $[0, 1]$, one might consider restricting a Gaussian’s color values to be on the same range, such as through a sigmoid. However, learning ‘negative’ color is important for improving temporal information reduction. Consider a video with

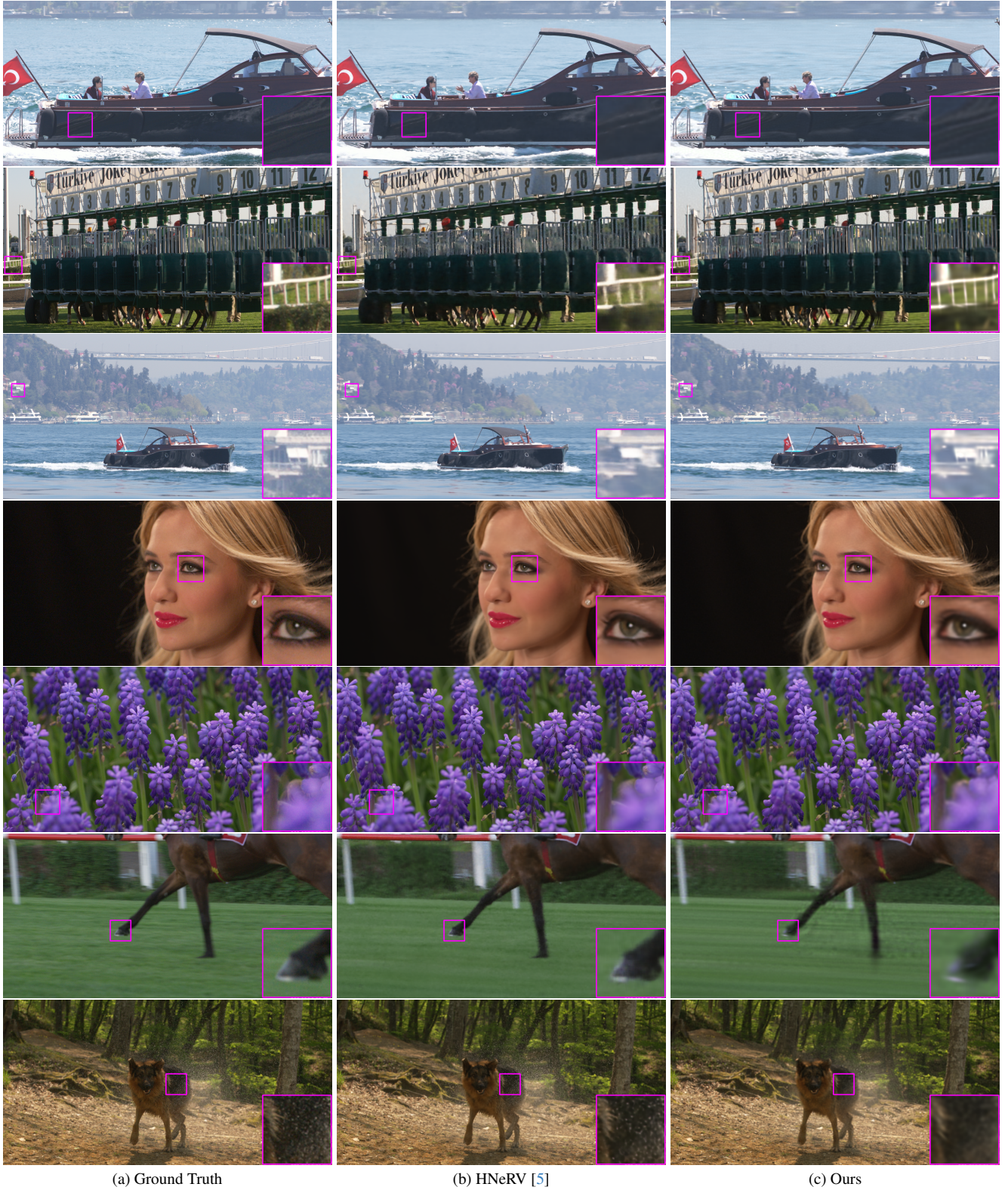


Figure 1. **Visual comparison of frames from UVG dataset.** Each representation uses approximately 0.06 bpp. GaRV preserves some details HNeRV cannot (top 3 rows), while blurring high frequency detail and motion that HNeRV handles better (bottom 2 rows).

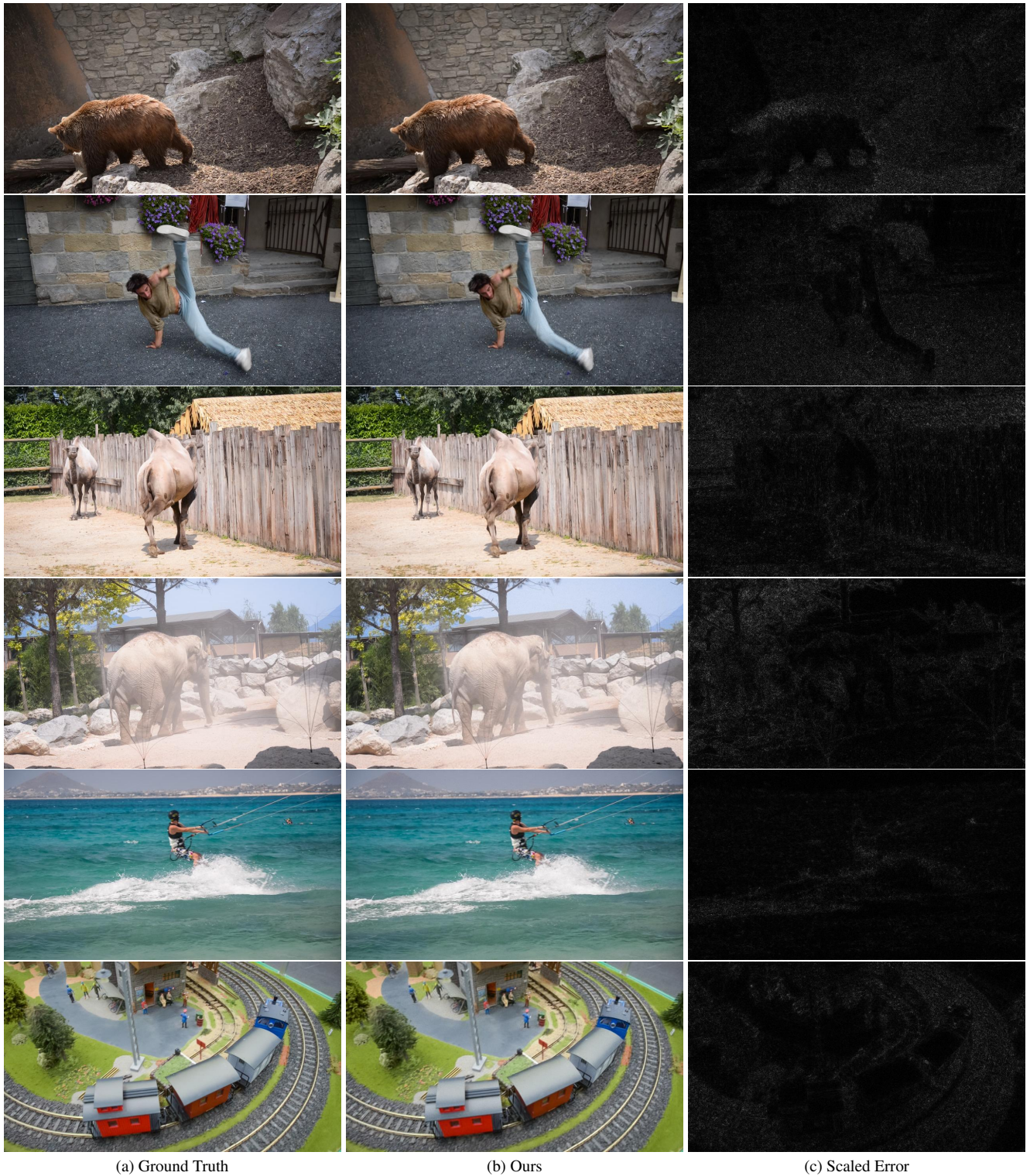


Figure 2. **Visual comparison of frames from DAVIS dataset.** Each representation uses approximately 1 bpp. GaRV performs well with temporal redundancy. Reconstruction error is concentrated in areas with high frequency changes such as gravel, trees, and waves.

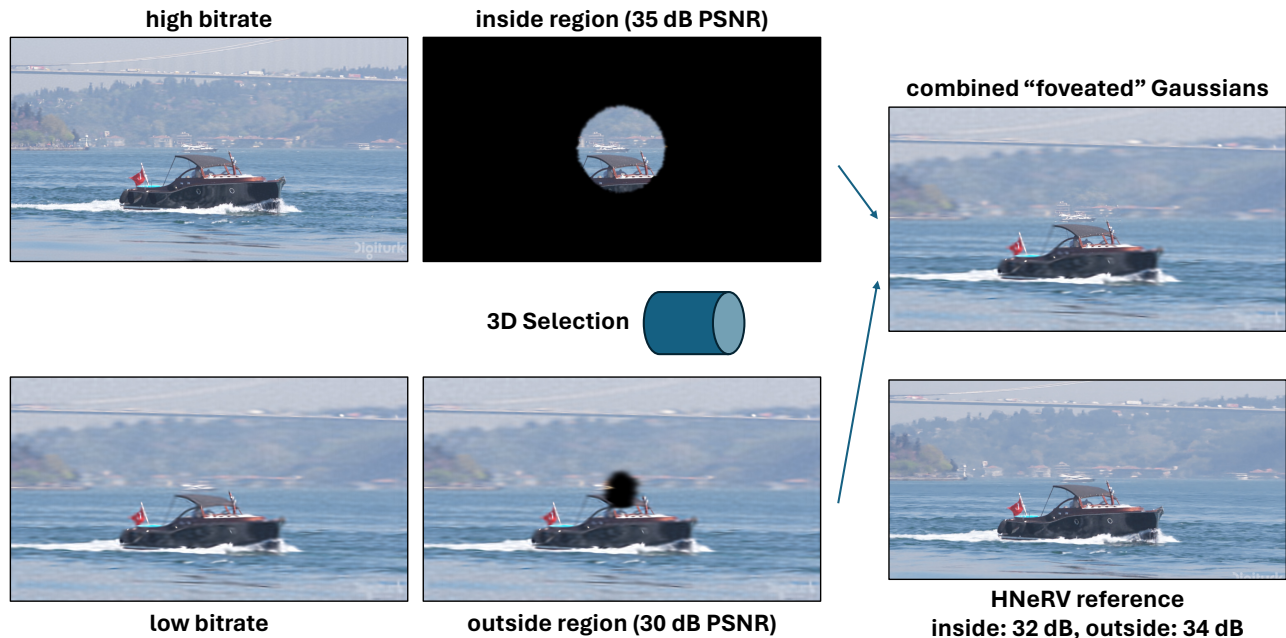


Figure 3. **Spatial resolution control visualization.** By mixing Gaussians from a high bitrate encoding that are within a region of interest, with Gaussians from a low bitrate encoding everywhere else, GaRV produces spatially dependent resolution.

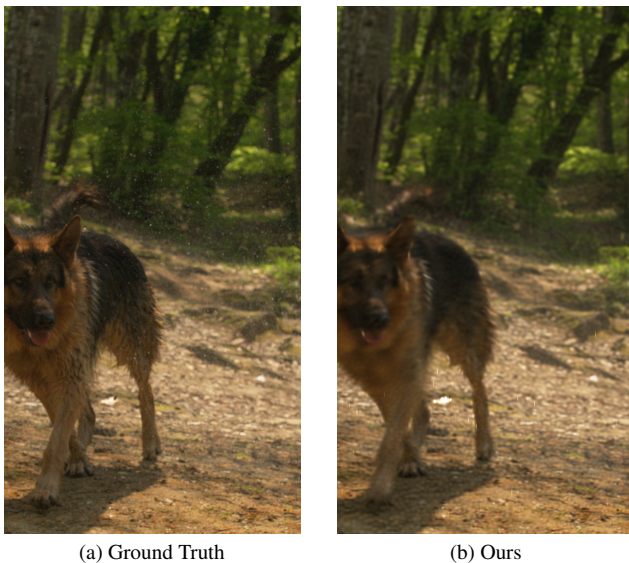


Figure 4. **Crop decoding.** GaRV can efficiently decoding arbitrary crops of video frames through a simple Gaussian-region intersection test.

one color for every frame except one. Without negative color, three groups of Gaussians are needed — before the unique frame, for the unique frame, and after the unique

frame. With negative color, we just need two sets where the set of Gaussians for the unique frame are the delta between that frame and the rest. Empirically, using ‘sigmoid’ results in a 2dB drop in quality with all else fixed.

Time Varying. GaRV is restricted to a pure-Gaussian approach, so complex temporal changes (such as high-frequency ones), may be difficult to capture. Similar to 4DGS [58], using a time-evolved color value may be useful for adding a different style of representation capacity. Although it breaks line-scan based decoding, we evaluate representing color as a d -degree polynomial up to a cubic. Each increase in degree, adds a million parameters, but only 0.1dB in frame quality. On the other hand, adding the equivalent number of Gaussians is more efficient. With 3M more parameters, +0.7dB when added as full Gaussians and only 0.3dB when added as color-polynomial coefficients. This suggests GaRV’s bottleneck is not temporal but spatial.

3. Slicing Gradient

We refer to 3DGS [23] and GaussianImage [60] for gradient computation details for rasterization and 3D Gaussian formation. Here we detail computing the gradient of the 3D Gaussian attributes through our ‘time slicing’ operation.



Figure 5. **Line scan decoding.** Since GaRV supports slicing along any axis, we can decode a row or column’s change over time at the same framerate of decoding traditional xy -planes (about 500 FPS).

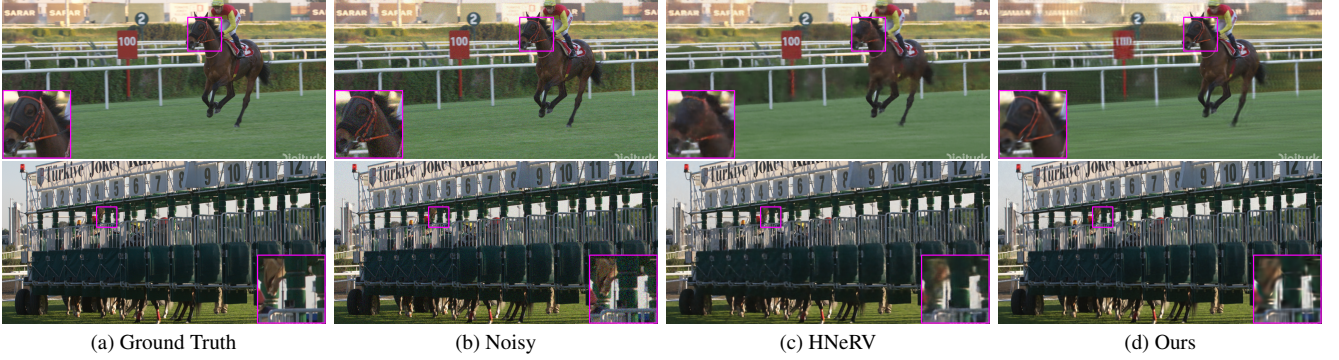


Figure 6. **Additive white-noise removal visualization from UVG dataset.** Top: ‘Jockey’ sequence. Bottom: ‘ReadySteadyGo’ sequence.

Recall from the main paper,

$$\mu_{xy|t} = \mu_{xy} + \Sigma_{xy,t} \Sigma_{t,t}^{-1} (t - \mu_t) \quad (1)$$

$$\Sigma_{xy|t} = \Sigma_{xy,xy} - \Sigma_{xy,t} \Sigma_{t,t}^{-1} \Sigma_{t,xy} \quad (2)$$

For notation sake, let $\hat{\mu} = \mu_{xy|t}$ be the 2D and $\hat{\Sigma} = \Sigma_{xy|t}$ be the 2×2 covariance matrix. The gradient through the 2D mean is given as,

$$\frac{\partial \hat{\mu}_x}{\partial \mu_x} = \frac{\partial \hat{\mu}_y}{\partial \mu_y} = 1 \quad (3)$$

$$\frac{\partial \hat{\mu}_x}{\partial \mu_t} = -\Sigma_{x,t} \Sigma_{t,t}^{-1} \quad (4)$$

$$\frac{\partial \hat{\mu}_y}{\partial \mu_t} = -\Sigma_{y,t} \Sigma_{t,t}^{-1} \quad (5)$$

$$\frac{\partial \hat{\mu}_x}{\partial \Sigma_{t,t}} = -\Sigma_{xy,t} (t - \mu_t) \Sigma_{t,t}^{-2} \quad (6)$$

$$\frac{\partial \hat{\mu}_x}{\partial \Sigma_{x,t}} = \frac{\partial \hat{\mu}_y}{\partial \Sigma_{y,t}} = \Sigma_{t,t}^{-1} (t - \mu_t) \quad (7)$$

All other partial gradients of $\hat{\mu}$ are 0. The gradient through the 2×2 covariance matrix is given as,

$$\frac{\partial \hat{\Sigma}_{i,i}}{\partial \Sigma_{i,i}} = 1 \quad (8)$$

$$\frac{\partial \hat{\Sigma}}{\partial \Sigma_{t,t}} = -\Sigma_{xy,t} \Sigma_{t,t}^{-2} \Sigma_{t,xy} \quad (9)$$

$$\frac{\partial \hat{\Sigma}_{i,i}}{\partial \Sigma_{i,t}} = \Sigma_{t,i} \Sigma_{t,t}^{-1} \quad (10)$$

$$\frac{\partial \hat{\Sigma}_{i,i}}{\partial \Sigma_{t,i}} = \Sigma_{i,t} \Sigma_{t,t}^{-1} \quad (11)$$

where $i \in \{x, y\}$. All other partial gradients of $\hat{\Sigma}$ are 0. These are chained with the gradients of the loss function with respect to the 2D Gaussian attributes to update the 3D Gaussian attributes.

4. Evaluation Metrics

Peak Signal-to-Noise Ratio (PSNR). First calculate the mean squared error between the original raw video frames \mathcal{I} and each decoded frame $\hat{\mathcal{I}}$:

$$MSE = \frac{1}{N} \sum_i (\mathcal{I}_i - \hat{\mathcal{I}}_i) \quad (12)$$



Figure 7. **Frame interpolation.** For low-motion videos (top), GaRV has strong interpolation capabilities. For high-motion (bottom), GaRV may distort object edges – see the letters/numbers.

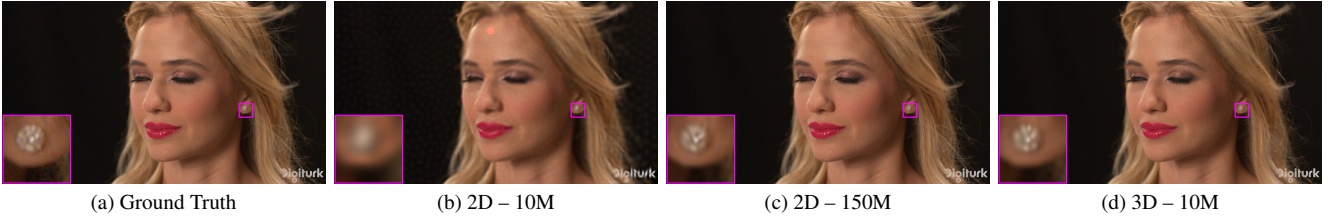


Figure 8. **3D vs 2D Gaussian visualization** on ‘Beauty’ sequence. Notice that 2D Gaussians need **15x** the number of parameters to reach 3D Gaussian performance.

Second, use the standard PSNR definition:

$$PSNR = 10 \log_{10} \frac{1^2}{MSE} \quad (13)$$

Bits-per-pixel (bpp). Recall, a video has (width) $W \times$ (height) $H \times$ (number of frames) T pixels. After quantization and attribute grid compression, the total bits used can be counted.

$$bpp = \frac{\text{total bits}}{\text{number of pixels}} \quad (14)$$

GPU Memory Usage (VRAM). We used `torch.cuda.max_memory_allocated()` in PyTorch [37] to measure the maximum memory used while decoding a frame and reset the statistic between tests.

Frames per Second (FPS) / Decode Time. We wrap frame decoding with `torch.cuda.Events`, then call `torch.cuda.synchronize` before measuring elapsed time. We take the average over 100 video decode repetitions to reduce system noise. Decode Time is then the average elapsed time to decode a single frame, and FPS is one over the decode time.

5. Alternative Gaussian Methods

5.1. Dynamics

Many researchers have been interested in extending 3D Gaussian Splatting to support scenes with motion for applications such as robotics [30] and content creation [12]. Given a multi-view video setup, one could fit a new 3DGS scene for every time step. Unfortunately, this leaves discrete disconnected sets of Gaussians, making any temporal analysis difficult. Instead, researchers have proposed allowing the Gaussian attributes to change smoothly over time [20, 53, 57]; similar to how color changes smoothly over view angle with spherical harmonics.

Dynamic Gaussian Splatting methods work by adding another function f such that $f(i, t)$ is the attributes of the i th Gaussian at time t . The exact choice of f is an interesting research problem. f may decide to only vary some attributes and leave others fixed. f may be learned or written explicitly. In [57], researchers train a neural network to predict Gaussian attributes at each time step, a deformation field with position and time as input. In [20], researchers opted to only allow a sparse set of Gaussians vary with a

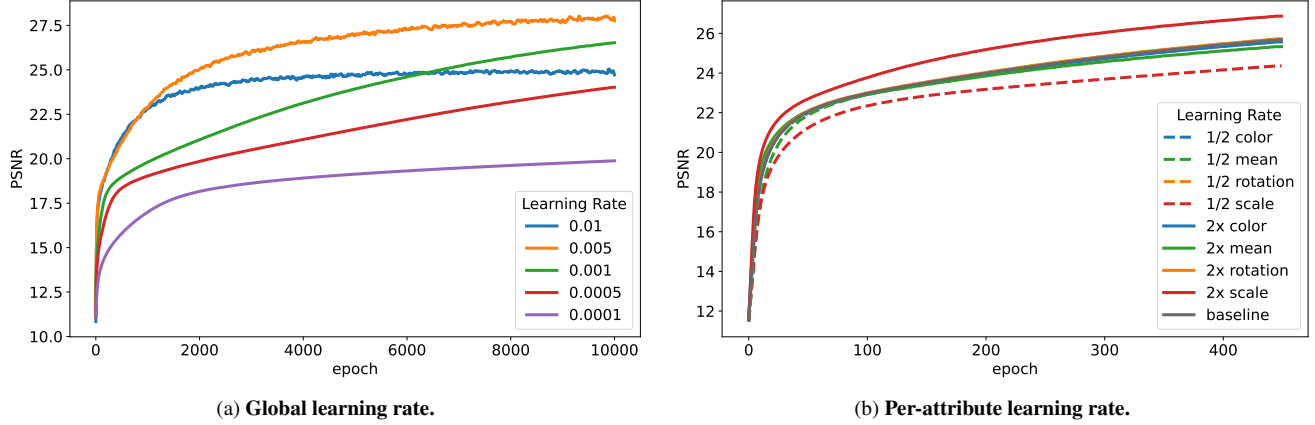


Figure 9. **Learning rate experiments** on the ‘ReadySteadyGo’ sequence. Left: using a global learning rate of 0.005 quickly recovers frame quality without plateauing to a sub-optimal solution. Right: changing learning rate by attribute significantly improves encoding time. The scale’s learning rate has the largest effect on fitting quality.

deformation field, with the reset using interpolated motion. In [53], researchers improved the encoding of Gaussians for the deformation field with neural voxels. Existing Gaussian video methods approach dynamics in a similar way: add another function that allow Gaussian attributes to change.

Instead of adding an additional component to modify Gaussians, researchers have proposed increasing from 3D to 4D [58]. Each Gaussian probability function then has 4 parameters x, y, z, t , which the rasterizer can evaluate at t . Effectively, each 3D Gaussian can move linearly over time. GaRV is similar to this approach, rather than introduce a bulky and slow mechanism for adding motion, we add a dimension to 2D Gaussians to support time.

5.2. Kernels

Recent 3DGS advancements show promise at integrating a non-Gaussian kernel into the rasterizer to improve quality while maintaining speed. GaRV focuses on a pure Gaussian setting, and lays the groundwork for future work to evaluate which techniques from 3D scenes may apply to video.

Instead of adding parameters by scaling the number of Gaussians, adding parameters to each Gaussian to learn some new ability may improve efficiency. Gabor filters [54] could improve high-frequency details. Generalized Exponential Splatting [14] could improve sharp edges. Radial Kernels [21] and 3D Convexes [17] could improve complex shapes.