Gaussian Representations for Video

Sachin Shah[1*] Anustup Choudhury[2] Guan-Ming Su[2]
Jaclyn Pytlarz[2] Christopher A. Metzler[1] Trisha Mittal[2]
[1]University of Maryland, College Park [2]Dolby Laboratories

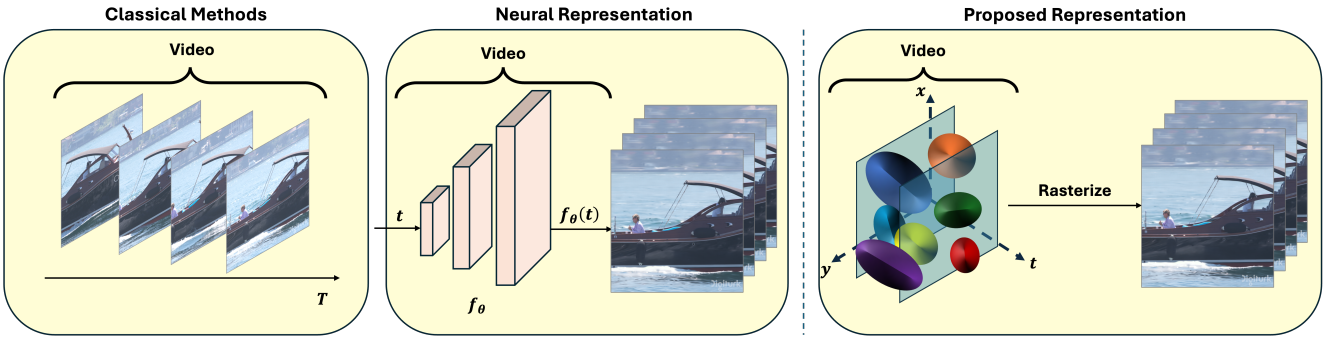

Figure 1. **Various representations of video.** *(Left)*: Explicitly represent video as **sequences of frames** (e.g. HEVC [47]). *(Middle)*: Implicitly represent video as **neural parameters** (e.g. NeRV [4], HNeRV [5], E-NeRV [29]). *(Right)*: Proposed implicit representation for video as a collection of **3D Gaussians**.

Abstract

We introduce Gaussian representations for videos (GaRV), a novel video encoding and decoding scheme based upon 3D Gaussians. Unlike traditional representations, which encode videos as sequences of frames, or neural representations, which encode videos within the weights of a neural network, we encode videos as a collection of 3D Gaussians within a space-time volume. The key advantage of our approach is that it enables efficient and flexible rasterization-based video decoding. With a slight drop in overall compression rate, GaRV offers an ***8-50****× improvement in decoding time and* ***2.5-15****× reduction in GPU memory compared with neural counterparts. Existing Gaussian video techniques require* ***2-30****× more disk space, while also using more GPU resources than GaRV. Moreover, GaRV offers unique flexibility in how and when pixels are decoded: One can non-sequentially decode frames/regions without penalty and can selectively decode regions at high-resolution to enable low-cost foveated video decoding.*

1. Introduction

Data can be thought of as the observed values of an underlying function. Recovering this function can enable novel applications since we can query at unobserved values. 2D Images belong to the family of functions with spatial coordinates $\in \mathbb{R}^2$ as input and color $\in \mathbb{R}^3$ as output, where we observe the output at pixel-aligned coordinates. 3D Scenes can belong to a function family mapping spatial coordinates $\in \mathbb{R}^3$ to a density $\in \mathbb{R}$ and color $\in \mathbb{R}^3$. 2D Video can either belong to a function family mapping spatial coordinates $\in \mathbb{R}^2$ and time $\in \mathbb{R}$ to color $\in \mathbb{R}^3$ or mapping only time $\in \mathbb{R}$ to a full-frame $\mathbb{R}^{W\times H\times 3}$. Finding these functions given limited data is an inherently ill-posed problem. There is some probability the unobserved values are random and usually no simple function family parametrization.

As shown in Figure 1, there are many possible function families for video. Before the digital era, video was simply an analog sequence of discrete frames. As digital video gained popularity, the need for more efficient representations quickly arose. Since each frame is not substantially different, we can exploit spatial and temporal similarities to reduce file sizes. After decades of hand-designing, we have formats such as HEVC [47] and AV1 [15], which push the boundaries of compression efficiency. Although these formats excel at what they were designed for — video compression — they become inflexible for downstream applications since they rely on multiple frames for decoding.

Recently, based on the idea that neural networks are "universal function approximators" [19], neural networks

*Work done during internship at Dolby Laboratories.

have been proposed as a function family which can produce sufficiently reasonable outputs for unobserved inputs. These 'implicit neural representations' (INRs) can be fit on data, such that the finalized network weights implicitly represent their data. In the context of 2D imagery, neural networks can predict the color value of a spatial coordinate [43, 45]. Unlike pixel-explicit counterparts, these network representations can produce 'super-resolved' imagery by evaluating the network at non-pixel aligned values. In the context of 3D scenes, neural networks can predict the color intensity of a point along a ray such that a ray-tracer querying the network would produce a view of the scene [32]. Unlike explicit counterparts like meshes, this representation can easily be formed from a collection of views instead of requiring a skilled 3D artist. In the context of 2D video, neural networks can predict the color values of full 2D-frames [4]. Unlike pixel-explicit counterparts, these network representations can interpolate between frames.

Although these methods excel in information recovery, they are often too slow or resource intensive for extracting content. Recently, using a more constrained function family (a weighted mixture of Gaussians) has shown promise at achieving similar quality while being significantly faster to decode data by sacrificing disk space. The function is formed as $f(\mathbf{x}) = \sum_i c_i p_i\left(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}\right)$, where p_i is the probability density function and c_i is the weight of the ith Gaussian. The mean $\boldsymbol{\mu}$, covariance $\boldsymbol{\Sigma}$, and weights are the learnable parameters to represent the input data. Most notably is 3D Gaussian Splatting (3DGS) [23], an alternative to NeRF, that uses 3D Gaussians to represent 3D objects. More recently, 2D Gaussians have been proposed for image compression with 1000 FPS decoding — GaussianImage [60].

However, while they show great promise at representing 3D and 2D scenes, existing Gaussian representations are ill-suited for representing video. GaussianImage [60] does not contain enough capacity to capture the time dimension. In theory 3DGS has enough dimensions, however, existing methods require "splatting" Gaussians onto the camera plane, losing the extra dimension. Recent work extends 3DGS to dynamic 3D scenes [53, 58]. Applying these techniques directly to single view video is difficult, not only because they require substantially more storage, but also because depth is ambiguous in single-view data. Concurrent to our work, researchers have experimented with overcoming this difficulty for fitting dynamic 3D Gaussian-based representations to single-view video [3, 46, 48]. Although successful in preserving frame quality, these methods sacrifice substantial disk space and rely on higher-order functions, which limit their speed and decode flexibility.

Rather than forcing dynamic 3D Gaussian Splatting frameworks to accommodate single-view data, our approach modifies static 3D Gaussians to directly represent video instead of 3D scenes. This work answers the question:

	Designed Once		Video Optimized	
	Classical[47]	**Learned[27]**	**Neural[4]**	**Gaussian (Ours)**
Encoding	**Fast**	Fair	Slow	Slow
Decoding	Fair	Slow	Fair	**Very Fast**
Disk Size	Fair	**Low**	Fair	Fair
Control	Low	Low	Fair	**High**

Table 1. **Video representation trade-offs.** Our representation trades off disk-space for significantly faster decoding and flexibility than existing approaches.

Can 3D Gaussians alone efficiently represent 2D video?
Overall, our contributions are:

- We propose GaRV, a novel learned representation for 2D time varying content through 3D Gaussians. Our method includes an encoding method for fitting Gaussians and a slice & rasterize decoder to recover 2D frames.
- Compared to implicit neural counterparts, GaRV is **8-50**× faster and uses **2.5-15**× less memory while only sacrificing marginal frame quality. GaRV also is **2-30**× smaller on disk than existing Gaussian video representations, while being **1.75-3.5**× faster.
- GaRV shows promise at enabling new tasks such as spatially variable resolution, as well as standard applications such as compression, denoising, and interpolation.

2. Related Works

2.1. Traditional Video Compression

Hand-crafted video codecs have been developed over the last two decades, methods such as MPEG [26], H.264 [51], and H.265 [47] offer state-of-the-art compression ratios while preserving frame quality. Their widespread popularity has earned these methods hardware level support on many devices to improve encoding and decoding efficiency. In general, these approaches use an inter-frame mechanism for encoding frames. Every few frames, a full key frame is encoded, then every subsequent frame is stored as the change from the previous frame until the next key frame. This helps reduce temporal redundancies since most pixels are relatively similar over time. These codecs have years of human expertise built into them; however, deep learning techniques have recently been proposed to further optimize these approaches [28]. For example, [41] demonstrated the ability to replace motion decoding with a learned neural network to improve decoded frame quality. These approaches show promise in improving the video compression ratio, but require lots of data to generalize well and can be significantly slower to decode than the hand-crafted counterparts.

Both approaches still have an explicit frame based representation, which can limit downstream applications. For example, since frames are reliant on prior decoded frames, it remains challenging to leverage parallel decoding.

2.2. Implicit Neural Networks for Video

Implicit neural networks have seen great success in a wide range of domains such as 3D modeling [32, 50], image compression [9, 43, 45], imaging through aberrations [11], image-to-image translation [44], deformation representation [8], and medical imaging [33]. INRs have also been proposed as a candidate for video representation [4]. Instead of designing a compression algorithm once, NeRV instead describes a function family whose parameters can be optimized per-video to represent frames. NeRV style INRs opt to treat video as a function over time, with a full-frame as output. Since its conception, numerous improvements have been proposed [2, 5, 7, 13, 29, 40, 42, 49, 49, 52, 55, 56, 59, 61]. Of note, E-NeRV [29] proposes a decoupling of spatial and temporal context to reduce redundant parameters. H-NeRV [5] observed index embedding was a key capacity limitation, so the authors proposed a hybrid approach where frame embeddings can be learned. Lastly, Boosting-NeRV [59] provides a general framework to improve the performance of existing NeRV-style models.

One major application NeRV works target is video compression [25]. Since INR parameter count becomes the number of bits necessary to store a video, video compression turns into a model compression problem. NeRV can also perform interesting downstream capabilities such as frame interpolation, since the function can be evaluated at any time [5]. Unfortunately, these INR methods require large computations for decoding, making them impractical on many devices. For instance, a 3M E-NeRV model [29] requires more than 50K floating point operations per-pixel.

2.3. Gaussian Content Representations

Gaussian representations are making breakthroughs in information representation. Unlike INRs, Gaussian representations are explicit and significantly faster to decode information from. In the context of 3D scenes, 3DGS [23] showed similar render view quality to Neural Radiance Field methods [32], but with a fraction of the time needed to render a single view. In the context of images, GaussianImage showed 3DGS can be streamlined with standard compression ideas [60]. GaussianImage presents a compact image codec with extremely fast decoding, while matching quality of handcrafted codecs like JPEG as well as INR methods. Recent works extend 3DGS to dynamic 3D scenes captured with multi-view video through 4D Gaussians [58] or dynamic 3D Gaussians according to a deformation function [53, 57] or control points [20].

Researchers have proposed modifications to dynamic 3DGS to overcome the single-view ambiguity problem. Splatter-a-Video [48] models video by moving 3D Gaussians with a polynomial Fourier basis, and relies on large motion and depth priors, which can easily fail for rapid complex motion, texture-less regions, or out-of-domain scenes. VeGaS [46] models video by moving 3D Gaussians with a polynomial, and restricts Gaussians to be flat on a plane to effectively remove depth. GaussianVideo [3] models video by moving 3D Gaussians with a spline and polynomial, and introduces a learnable camera model, which restricts the types of video fittable to natural scenes. These representations are still large and slow since they perform an expensive 'splat' projection to produce frames from 3D.

Our approach is similar to 4D Gaussian Splatting [58], which has a multi-stage pipeline: 4D to 3D through time conditioning, 3D to 2D through splatting projection, then 2D depth-dependent rasterization. Instead of overcoming the ambiguity problem to make 4DGS work on video, our method streamlines these ideas for time varying 2D data directly, similar to how GaussianImage [60] streamlined 3DGS for 2D data. Specifically, we propose time conditioning as a "slicing" mechanism for extracting 2D planes from a 3D volume as a replacement for "splatting".

3. Method

Our approach treats video as a 3-dimensional volume in the space-time x, y, t axes, where each frame is the intensity on an xy-plane at time t. In this way, the volume has discrete solutions at x, y pixel aligned coordinates and t aligned to frame timestamps. Our goal is to fit a mixture of 3D Gaussians to the color intensities at each (width) $W\times$ (height) $H\times$ (number of frames) T pixel grid point.

A brief overview of our approach is shown in Figure 2. Our approach starts with forming static 3D Gaussians from a factorized representation in §3.1. Then, we build up the process of decoding 2D frames from 3D Gaussians in §3.2. We first describe an easier problem of decoding 2D frames from 2D Gaussians in §3.2.1, then upgrade to 3D Gaussians by introducing a novel "**Time Slicing**" 3D to 2D transformation method to produce a set of 2D Gaussians that can be rasterized to a frame in §3.2.2.

3.1. 3D Gaussian Formulation

Similar to 3DGS [23], our 3D Gaussians are described by a position $\boldsymbol{\mu} \in \mathbb{R}^3$, 3D covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{3\times3}$, and color $\boldsymbol{c} \in \mathbb{R}^3$ (RGB), however, we operate on the x, y, t axes instead of x, y, z. We parameterize position as 3 real values, μ_x, μ_y, μ_t. Unlike 3DGS, video has no-view dependent effects to model, so color is simply parameterized as 3 real values, r, g, b representing the red, green, and blue channels respectively. Since the covariance matrix must be positive semi-definite (PSD), and constraining gradient descent to produce PSD matrices is error prone, we use a Cholesky decomposition [18]. Here, covariance is factorized into a lower triangular matrix $\mathbf{L} \in \mathbb{R}^{3\times3}$ such that $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^\top$, where $\mathbf{L}^\top$ is the conjugate transpose of $\mathbf{L}$. GaussianImage [60] found the Cholesky decomposition to be more stable and smaller than the Rotation + Scale factorization used

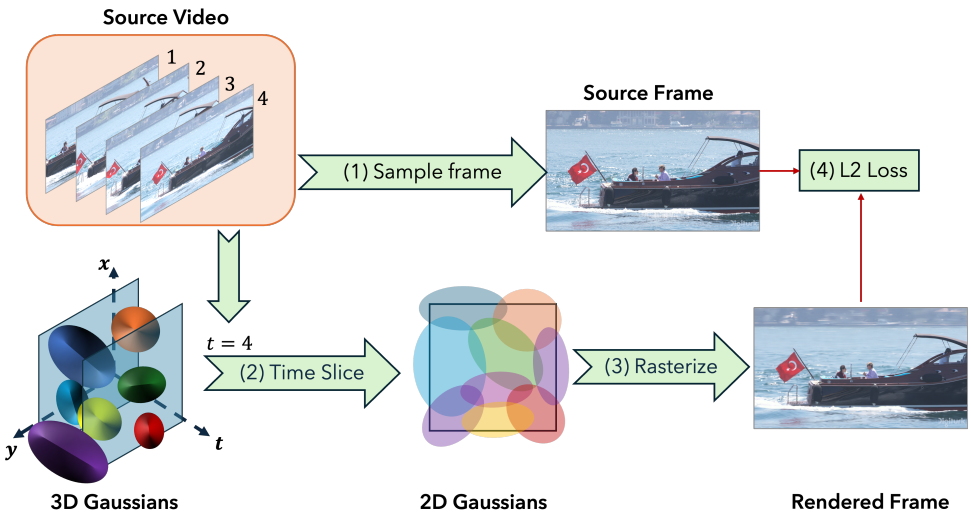

Figure 2. **GaRV Encoding.** Our representation is fit iteratively per-video. For each iteration, a single frame is sampled from the source video (1). The 2D Gaussian time slice is computed (2), then rasterized to form a decoded frame (3). Finally, the L2 loss can be computed between the original frame, and the decoded frame (4). This process is repeated until convergence.

in 3DGS. Since $\mathbf{L}$ must be lower-triangular, there are only 6 non-zero learnable parameters, $\{l_1, \ldots, l_6\}$. The parameters along the diagonal relate to the scale of the Gaussian, while the parameters off the diagonal represent the rotation.

3.2. Frame Extraction

In this section, we describe the process of recovering the frame intensity at a specific time step t from the 3D volume.

Conceptually, Gaussians are treated as 3D ellipsoids which can intersect with an xy-plane at a fixed time step (§3.2.2). The ellipsoid-plane intersection becomes the pixel values of a frame (§3.2.1). In this way, as the time value is increased, the xy-plane we intersect with *slides* along, and the intersection also changes depending on the ellipsoid parameters. Intuitively, Gaussians with long scales along the time axis and no rotation, represent static regions since the ellipsoid-plane intersection would not change shape drastically over time. On the other hand, Gaussians with a rotation about the xy-plane would form linearly moving dots since the intersection shape would move. Therefore, dynamic video regions can be represented as smaller piecewise linear movements such that each component is represented as a Gaussian with some rotation.

3.2.1. 2D Rasterization

First, consider a simpler problem where we want to decompose a 2D grid into a collection of 2D Gaussians. We can use the simplified accumulated summation rasterizer proposed in GaussianImage [60]. A single pixel's color $f(x, y)$ can be computed as a weighted average of the probability densities p of each Gaussian at coordinate (x, y) multiplied by the Gaussian's corresponding color $\boldsymbol{c}$:

$$f(x, y) = \sum_{i}^{N} p_i(x, y)\boldsymbol{c_i} \tag{1}$$

where N is the total number of Gaussians. Similar to 3DGS, the rasterizer can be tile-based where only Gaussians with a 99% confidence interval intersecting a tile are processed, significantly speeding up rendering, since we can compute $p_i(x, y)$ more sparsely, and approximate the term as 0 for Gaussians with low tile intersection.

3.2.2. Time Slicing for 3D Upgrade

To accommodate a 3D volume instead of a 2D grid, we could naively extend Eqn. (1) to 3 dimensions as follows:

$$f(x, y, t) = \sum_{i}^{N} p_i(x, y, t)\boldsymbol{c_i} \tag{2}$$

With t fixed, $\mathcal{I}_t(x, y) = f(x, y, t)$ is the function that represents a video frame. Then, each frame is the sum of the intersections between each 3D Gaussian and the xy-frame. We can make this relationship more explicit, by treating p_i as the joint probability density function of x, y and t. We can then factorize $p_i(x, y, t)$ into a conditional probability $p_i(x, y \mid t)$ and marginal probability $p_i(t)$, resulting in,

$$f(x, y, t) = \sum_{i}^{N} p_i(x, y \mid t)p_i(t)\boldsymbol{c_i} \tag{3}$$

We refer to this as **Time Slicing**, since $p_i(x, y \mid t)$ represents the 2D Gaussian intersection shape of a 3D Gaussian and the xy-plane at time t, and $p_i(t)$ represents the size of the intersection. This formulation has two major advantages:

1. *Faster per-pixel computation.* In Eqn. (2), if we precompute $\mathbf{\Sigma}^{-1}$, each Gaussian requires 26 floating point operations (FLOPs) — 3 for multiplying by $\mathbf{c_i}$, 3 for $\mathbf{d} = \{x, y, t\} - \boldsymbol{\mu}$, and 20 for $\mathbf{d}^\top \mathbf{\Sigma}^{-1} \mathbf{d}$. In our factorization as shown in Eqn. (3), $p_i(t)$ can be evaluated once, and pre-multiplied with $\mathbf{c_i}$, as well as the 2D Gaussian attributes $\boldsymbol{\mu'} \in \mathbb{R}^2$ and $\mathbf{\Sigma'} \in \mathbb{R}^{2\times2}$ that represent $p_i(x, y \mid t)$. So each Gaussian requires only 14 FLOPs per-pixel — 3 for multiplying by the marginal and color $p_i(t)\mathbf{c_i}$, 2 for $\mathbf{d} = \{x, y\} - \boldsymbol{\mu'}$, and 9 for $\mathbf{d}^\top \mathbf{\Sigma'}^{-1} \mathbf{d}$.
2. *Point culling.* In the Gaussian preprocessing stage, we can cull points if $p_i(t)$ is too small (frame-wise-cull). Similar to 3DGS, we can treat $p_i(x, y \mid t)$ as a 2D Gaussian and use the same culling technique to reduce the number of Gaussians considered per tile (tile-wise-cull). Both methods reduce the number of Gaussians summed in Eqn. (3), increasing speed, with minimal impact on quality since the excluded Gaussians have little weight.

The conditional $p_i(x, y \mid t)$ can be derived from the properties of the multivariate Gaussian. Let

$$\mathbf{\Sigma} = \begin{bmatrix} \mathbf{\Sigma}_{xy,xy} & \mathbf{\Sigma}_{t,xy} \\ \mathbf{\Sigma}_{xy,t} & \mathbf{\Sigma}_{t,t} \end{bmatrix} \tag{4}$$

be a block subdivision of covariance, where $\mathbf{\Sigma}_{xy,xy}$ is the 2×2 covariance of the pair (x, y) and $\mathbf{\Sigma}_{t,t}$ is the variance of t. The conditioned mean and covariance is given as:

$$\boldsymbol{\mu}_{xy|t} = \boldsymbol{\mu}_{xy} + \mathbf{\Sigma}_{xy,t}\mathbf{\Sigma}_{t,t}^{-1}\left(t - \mu_t\right) \tag{5}$$

$$\mathbf{\Sigma}_{xy|t} = \mathbf{\Sigma}_{xy,xy} - \mathbf{\Sigma}_{xy,t}\mathbf{\Sigma}_{t,t}^{-1}\mathbf{\Sigma}_{t,xy} \tag{6}$$

3.3. Video Encoding

Each video is 'encoded' into our representation through an optimization process. We optimize the 3D Gaussian attributes to minimize the difference between each video's frame collection $\{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_T\}$ and the resulting rendered frames $\{\hat{\mathcal{I}}_1, \hat{\mathcal{I}}_2, \ldots, \hat{\mathcal{I}}_T\}$. In each encoding iteration, we randomly sample a single frame to consider. The 3D Gaussian volume is sliced at its corresponding time step, then rasterized to a decoded frame. We use the loss function $\mathcal{L}_2$ between the original $\mathcal{I}_t$ and decoded frame $\hat{\mathcal{I}}_t$. Since our slice & rasterize mechanism is differentiable, we can backpropagate the loss directly to the 3D Gaussian attributes. This process is repeated until convergence.

4. Experimental Results and Applications

In this section, we provide details about the dataset, implementation and compare GaRV with video INR and Gaussian methods. Additionally, we demonstrate 5 applications of GaRV in Frame Decoding, Video Compression, Spatially Variable Resolution, Frame Interpolation, and Denoising.

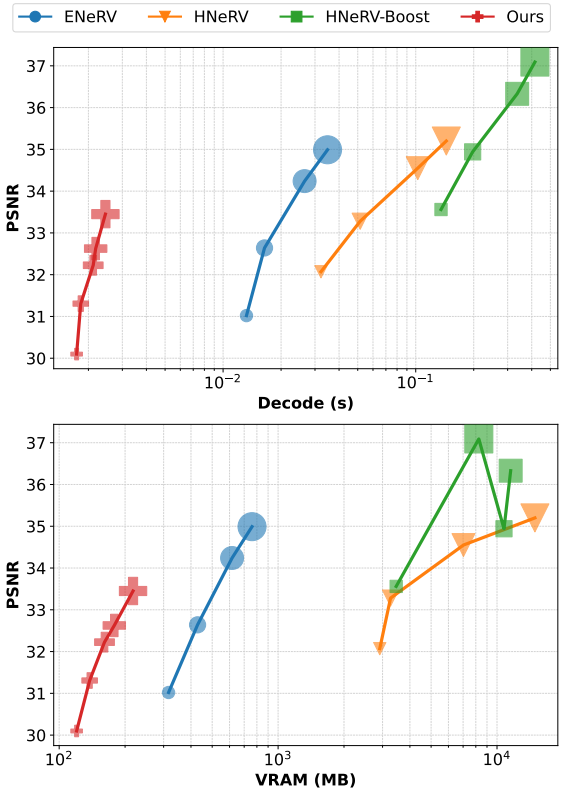

Figure 3. **Video decoding efficiency.** Dot size represents disk space. Observe how GaRV is substantially faster (as denoted by lower Decode values in the top plot) and more memory efficient (as denoted by lower VRAM values in the bottom plot) than its neural counterparts. Since GaRV's largest model is substantially faster than the next fastest neural model, we can significantly scale up the number of Gaussians.

Implementation Details. We use the Adam optimizer [24] with a learning rate of 5×10^{-3}, and the ReduceOnPlateau learning rate scheduler with a patience set to 25. We use a batch size of 16, 2000 training epochs, and 20 warmup epochs. We implement our slice & rasterize method in CUDA [35] and the training loop in PyTorch [37]. All experiments are run on NVIDIA RTX A4000s. Encoding time is 2-6 hours for GaRV, and 4-10 hours for neural variants.

Comparisons. We fit our method on the UVG dataset [31] at 1920×1080 resolution at 120 fps to compare against recent neural video representations, ENeRV [29], HNeRV [5], and HNeRV-Boost [59]. For each task, we use PSNR to evaluate frame quality and standardize bits-per-pixel for fair comparison. We measure peak GPU memory usage (VRAM) while frame decoding and average frame decode time to evaluate efficiency. We also compare with recent Gaussian video representations, Splatter-a-Video [48], VeGaS [46], and GaussianVideo [3], on the DAVIS dataset at 480p [38].

(a) Ground Truth (b) HNeRV [5] (c) Ours

Figure 4. **Visual comparison of frames from UVG dataset** at 0.06 bpp. Top: 'Bosphorus' sequence, Bottom: 'YachtRide' sequence.

4.1. Frame Decoding

Full video decoding. As described in §3.2, one can easily extract a frame from our representation by calculating Eqn. (3) with our slice & rasterize technique. As shown in Figure 3, our method is significantly more efficient than neural alternatives. Boosting-NeRV [59] is a recent work and although boasts an improvement in frame quality, comes at a significant cost to memory usage and computation time. GaRV can produce similar frame qualities **50x** faster with **15x** less GPU memory by trading off disk space making it more practical for consumer grade hardware. We show example frames of our method in Figure 4.

Arbitrary frame decoding. Similar to neural counterparts, Eqn. (3) can be calculated at any time t with no dependence on previous computations. This can enable efficient frame skipping and parallel decoding unlike hand-crafted codecs which require decoding a previous key-frame. Since GaRV requires significantly less compute than neural counterparts, more frames can be feasibly decoded simultaneously.

Arbitrary pixel decoding. Unique to GaRV is the ability to selectively decode frames not only temporally, but also spatially without any dependence on neighboring pixels. Existing neural representations parameterize solely on time so they fundamentally only extract full-frames at once. And, hand-crafted codecs leverage high temporal-spatial de-duplication to produce disk-compact files disabling arbitrary partial frame decoding. Since GaRV's underlying representation is spatially interpretable, we can easily decode only a partial region. For instance, smart phone users occasionally watch horizontal 16:9 content cropped to their vertical 9:16 aspect ratio screen ($1080 \times 1920 \rightarrow 1080 \times 608$).

Since only a third of pixels are visible by the user, rendering a full frame wastes valuable computation power. GaRV can simply select which Gaussians to consider before rasterization. Performing this process at frame-decode time results in $+250$ FPS and -50 MB VRAM usage for videos at 0.06 bpp. Leveraging a 2ms pre-processing step to 'discard' unnecessary Gaussians before decoding any frames, can double the gains to $+500$ FPS and -120 MB VRAM — **2x** faster and a **third** of memory.

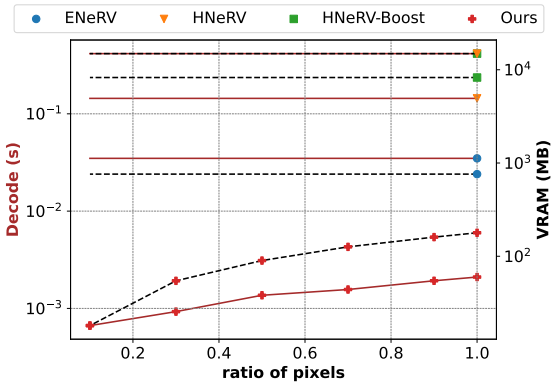

Figure 5. **Decode efficiency as fewer pixels are requested.** Solid-brown line represents frame decode speed on the left, while dashed-black lines represents memory usage on the right. GaRV supports partial-frame decoding for more resource efficient rendering if only select regions will be displayed, unlike neural counterparts which have a fixed cost for rendering an entire frame.

Line scan decoding Another unique capability of GaRV is the ability decode a frame row or column's change over time, i.e., rasterize an image slice of the 3D volume parallel to the xt or yt planes. With existing representations, if one wants to know how a pixel's color varies, one must decode every pixel of every frame. GaRV on the other hand can change the axis sliced over. For instance,

$$f(x, y, t) = \sum_{i}^{N} p_i(x, t \mid y) p_i(y) \boldsymbol{c_i}, \tag{7}$$

produces an xt-image at a y row. This leverages our fast slice & rasterize based renderer but selects a different plane

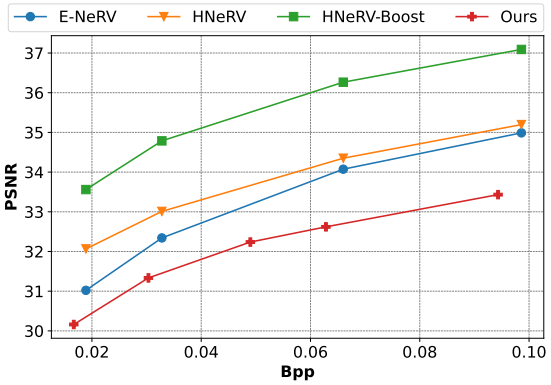

Figure 6. **Frame quality vs BPP** on the UVG Dataset. Our method trades-off some frame-quality at lower bits-per-pixel in favor of substantially faster decoding. Since GaRV is significantly more memory efficient, we can scale the number of Gaussians to match frame quality without consequence.

through the 3D volume. Neural video representations can produce a line scan in 8s with a 3M E-NeRV model, while GaRV takes just 8ms — a **1000x** speed up.

4.2. Compression

Our Gaussian representation is naturally much smaller than the original pixel collection since $N \ll W \times H \times T$. However, we can still reduce the storage requirements of each Gaussian from needing 32-bits for all 12 parameters.

We leverage an additional quantization-aware fine-tuning compression stage. For each attribute, we use b-bit asymmetric quantization [1], with $b = 16$ for position and $b = 8$ for color and Cholesky coefficients, so in total each Gaussian uses $16 \times 3 + 8 \times 3 \times 3 = 120$ bits — **3x smaller**.

Recently, many Gaussian attribute compression techniques have been proposed in the context of 3DGS [6, 10, 16, 34]. Self-Organizing Gaussian Grids [34] is a generalized technique to produce smooth attribute orderings of 3DGS attributes on 2D planes, to leverage entropy encoding mechanisms to compress them. We modify this approach for use in the video setting. To compress N Gaussians, we reshape the parameters into a $\sqrt{N} \times \sqrt{N}$ grid with 12 channels. We perform Parallel Linear Assignment Sorting (PLAS) [34] with the position and color attributes as keys. Then, we use JPEG-XL [39] to compress each attribute grid. On any Gaussian set, this scheme can produce a 20% reduction in space, but through train-time regularization we can improve this result to a 50% reduction. While fitting Gaussians to a video, we regularize the Cholesky parameters to be smoother, thereby improving JPEG-XL's efficiency at grid compression. We compare frame quality as bits-per-pixels (bpp) increase in Figure 6.

4.3. Spatially Variable Resolution

As consumer grade cameras and displays improve, video can be increasingly captured and played at extremely high resolutions. However, it is expensive to stream these higher qualities. Fortunately, in many setups, the user does not actually need a full high-resolution image. For instance, when watching a video in VR, only 4% of pixels are fully in view, the rest are naturally blurred by the human eye [22]. In another context, the 'interesting' part of a video may lie in a specific region, such as in sports games near the ball. Then, sending just the region of interest in full-resolution, and the rest in a lower-resolution would give the user a similar experience. Unfortunately, existing video codec's cannot effectively take advantage of this. Existing codecs potentially can re-encode a video with blurry peripherals, but that is slow to perform on the fly and still sends full-resolution worth of bits, so depending on the codec's efficiency at blurry regions some bits will be wasted.

Our approach is different. In §4.1 we described how Gaussians can be selectively processed to produce cropped frames. By leveraging this selection process, we can mix Gaussians in the compressed space to directly control (a) where sharp regions are and (b) how many bits per pixel to spend on unimportant regions. Figure 7 visualizes this spatial resolution control. At encoding time, fit the video twice: once at a high bitrate to preserve fine detail, second at a low bitrate to produce blurry frames. Then, depending on the region of interest (i.e., where the user is looking), we mix-and-match the two sets of Gaussians. This produces a "foveated" set of Gaussians, where there is high detail inside the selection and blurry details outside the selection.

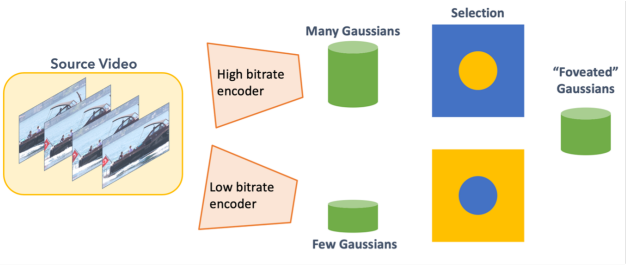

Figure 7. **Spatial Resolution Control** in our representation. Encode twice, at two bitrates, and mix-and-match to form a spatially variable set of Gaussians to stream. The top route captures better detail with many tinier Gaussians, while the bottom route produces blurry frames with fewer larger Gaussians.

Even though GaRV's frame quality at low bit rates is worse than neural counterparts, if we only care about a random 4% region when decoding, we can merge two sets of Gaussians to match frame quality with faster rendering time (Table 2). This process can happen quickly in the compressed space – 1ms – making it ideal for streaming applications where a server holds full-resolution video, and sends

video to consumers on differing network abilities.

Method	PSNR↑	FPS↑	VRAM↓
E-NeRV [29]	29.45	75	315 MB
HNeRV [5]	30.64	31	2.9 GB
HNeRV-Boost [59]	**32.34**	07	3.5 GB
Ours	32.06	**240**	**210** MB

Table 2. **Preservation of a region of interest.** If only 4% of pixels are important, our method can adaptively distribute bits to spend more on that region than the peripheral. Neural counterparts on the other hand have uniform quality control, so a regions quality will decreases as the total available bits also decrease.

4.4. Dynamic Gaussian Representations

Concurrent Gaussian-based video representations primarily target video editing and tracking related tasks, neglecting the impact of their choices on disk space. From a functionality perspective, GaRV can better maximize flexible decoding and spatially variable resolution. Whereas other methods require evaluating expensive projection operations before determining if a Gaussian should be decoded, GaRV requires only a lightweight ellipsoid-plane intersection test.

Table 3 shows a quantitative comparison on the DAVIS dataset [38]. Since concurrent works are not concerned with disk space and utilize an adaptive Gaussian framework, bits per pixel is estimated based on reported Gaussian counts and attribute size with an overly optimistic 8-bit quantization. GaRV's linear motion is more efficient at preserving frame quality than the higher-order motions in other methods. In §2.3 of the supplement, we show adding more Gaussians is more efficient than introducing higher-order terms in terms of representation size.

Method	BPP↓	PSNR↑	FPS↑	VRAM↓
Splatter-a-Video [48]	$3-30$	28.6	150	$1-10$ GB
VeGaS [46]	3	33.3	200	200 MB
GaussianVideo [3]	2	33.8	100	—
GaRV (Ours)	**1**	**34.0**	**350**	**70** MB

Table 3. **DAVIS Dataset frame quality.** GaRV preserves frame quality more efficiently than concurrent works.

4.5. Frame Interpolation

Since our representation shares Gaussians over time, time steps between two supervised frames still intersect with some Gaussians, forming a 'video interpolation' effect. We evaluate performance of our representation on this task by holding out half of the frames while fitting, and evaluate decoded quality on the remaining frames. Specifically, for a 120 FPS video, we fit to a downsampled 60 FPS video starting at frame '0', then evaluate on a 60 FPS video starting at frame '1'. The quantitative details are in Table 4. GaRV produces similar frame quality to neural counterparts, while not having an explicit mechanism for interpolating frames, but at a significantly faster FPS and as mentioned earlier, at significantly lower memory requirement.

Method	PSNR↑	FPS↑
E-NeRV [29]	28.73	75
HNeRV [5]	29.65	31
HNeRV-Boost [59]	**29.99**	07
Ours	28.86	**470**

Table 4. **Frame interpolation.** Our representation renders unseen frames at 3% worse quality but with over **60x** the frame rate.

4.6. Denoising

Since Gaussians are naturally smoothing, GaRV can automatically remove frame noise. We evaluate our denoising capability by applying additive white Gaussian noise to the source video. The original noisy frames have 23.71 dB PSNR. NeRV [4] showed neural methods were second to median filters [36] at Gaussian noise removal. On our noisy frames, a median filter reaches 30.49 dB. HNeRV [5] surpasses this baseline with 30.85 dB. However, our method beats both methods with **32.36** dB. We visualize sample noisy and denoised frames in the supplement.

5. Conclusion

Static 3D Gaussians in the x, y, t space can efficiently represent video content. GaRV enables orders of magnitude more efficient decoding through our novel slice & rasterize technique, which also unlocks more control over when and which regions to decode compared to all existing representations. GaRV is up to **50**× faster than neural counterparts [59], while using **15**× less memory. Compared to dynamic 3D Gaussian approaches to video, GaRV is **2-30**× smaller on disk, while still being faster and more memory efficient. GaRV unlocks an exciting new direction for practical and efficient video representations.

Limitations. There are some limitations with our proposed method. First, the base Gaussian distribution has little capacity to represent complex shapes, requiring more Gaussians to cover the gap. Leveraging alternative primitives such as the Generalized Normal Distribution [14], Gabor filters [54], Smooth Convexes [17] or Deformable Radial Kernels [21] may add capacity more efficiently than scaling the number of Gaussians. Second, to achieve sufficiently high frame quality, the fitting time of GaRV is much longer than the encoding time of traditional video codecs.

Extensions. Although we focus on RGB-videos, our method can be extended to other domains; trivially, to multi-spectral videos and also, other 2D time varying data such as event-streams. Further from our work, slicing can be used on 3DGS scenes to view interior structures. Our proof of concept spatial resolution control can enable progressive video loading, switching between 'high' and 'low' bit settings as network connectivity changes.

Acknowledgements

S.S. and C.A.M. were supported in part by funds from the Dolby Seed Grant program and ONR award no. N00014-23-1-2752.

References

[1] Yash Bhalgat, Jinwon Lee, Markus Nagel, Tijmen Blankevoort, and Nojun Kwak. Lsq+: Improving low-bit quantization through learnable offsets and better initialization. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2978–2985, 2020. 7

[2] Monsij Biswal, Tong Shao, Kenneth Rose, Peng Yin, and Sean Mccarthy. Steganerv: Video steganography using implicit neural representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 888–898, 2024. 3

[3] Andrew Bond, Jui-Hsien Wang, Long Mai, Erkut Erdem, and Aykut Erdem. Gaussianvideo: Efficient video representation via hierarchical gaussian splatting, 2025. 2, 3, 5, 8

[4] Hao Chen, Bo He, Hanyu Wang, Yixuan Ren, Ser-Nam Lim, and Abhinav Shrivastava. NeRV: Neural representations for videos. In *NeurIPS*, 2021. 1, 2, 3, 8

[5] Hao Chen, Matthew Gwilliam, Ser-Nam Lim, and Abhinav Shrivastava. HNeRV: Neural representations for videos. In *CVPR*, 2023. 1, 3, 5, 6, 8, 2

[6] Yihang Chen, Qianyi Wu, Weiyao Lin, Mehrtash Harandi, and Jianfei Cai. Hac: Hash-grid assisted context for 3d gaussian splatting compression. In *European Conference on Computer Vision*, 2024. 7

[7] Anustup Choudhury, Praneet Singh, and Guan-Ming Su. Nerva: Joint implicit neural representations for videos and audios. In *2024 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2024. 3

[8] Boyang Deng, J. P. Lewis, Timothy Jeruzalski, Gerard Pons-Moll, Geoffrey Hinton, Mohammad Norouzi, and Andrea Tagliasacchi. Nasa neural articulated shape approximation. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VII*, page 612–628, Berlin, Heidelberg, 2020. Springer-Verlag. 3

[9] Emilien Dupont, Adam Goli'nski, Milad Alizadeh, Yee Whye Teh, and A. Doucet. Coin: Compression with implicit neural representations. *ArXiv*, abs/2103.03123, 2021. 3

[10] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps, 2023. 7

[11] Brandon Y. Feng, Haiyun Guo, Mingyang Xie, Vivek Boominathan, Manoj K. Sharma, Ashok Veeraraghavan, and Christopher A. Metzler. Neuws: Neural wavefront shaping for guidestar-free imaging through static and dynamic scattering media. *Science Advances*, 9(26):eadg4671, 2023. 3

[12] Quankai Gao, Qiangeng Xu, Zhe Cao, Ben Mildenhall, Wenchao Ma, Le Chen, Danhang Tang, and Ulrich Neumann. Gaussianflow: Splatting gaussian dynamics for 4d content creation. *Transactions on Machine Learning Research*, 2025. 6

[13] Ahmed Ghorbel, Wassim Hamidouche, and Luce Morin. Nerv++: An enhanced implicit neural video representation. *arXiv preprint arXiv:2402.18305*, 2024. 3

[14] Abdullah Hamdi, Luke Melas-Kyriazi, Jinjie Mai, Guocheng Qian, Ruoshi Liu, Carl Vondrick, Bernard Ghanem, and Andrea Vedaldi. Ges : Generalized exponential splatting for efficient radiance field rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19812–19822, 2024. 8, 7

[15] Jingning Han, Bohan Li, Debargha Mukherjee, Ching-Han Chiang, Adrian Grange, Cheng Chen, Hui Su, Sarah Parker, Sai Deng, Urvang Joshi, Yue Chen, Yunqing Wang, Paul Wilkins, Yaowu Xu, and James Bankoski. A technical overview of av1. *Proceedings of the IEEE*, 109(9):1435–1462, 2021. 1

[16] Alex Hanson, Allen Tu, Vasu Singla, Mayuka Jayawardhana, Matthias Zwicker, and Tom Goldstein. Pup 3d-gs: Principled uncertainty pruning for 3d gaussian splatting. *arXiv*, 2024. 7

[17] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025. 8, 7

[18] Nicholas J. Higham. Cholesky factorization. *WIREs Computational Statistics*, 1(2):251–254, 2009. 3

[19] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. 1

[20] Yi-Hua Huang, Yang-Tian Sun, Ziyi Yang, Xiaoyang Lyu, Yan-Pei Cao, and Xiaojuan Qi. Sc-gs: Sparse-controlled gaussian splatting for editable dynamic scenes. *arXiv preprint arXiv:2312.14937*, 2023. 3, 6

[21] Yi-Hua Huang, Ming-Xian Lin, Yang-Tian Sun, Ziyi Yang, Xiaoyang Lyu, Yan-Pei Cao, and Xiaojuan Qi. Deformable radial kernel splatting. *arXiv preprint arXiv:2412.11752*, 2024. 8, 7

[22] Susmija Jabbireddy, Xuetong Sun, Xiaoxu Meng, and Amitabh Varshney. Foveated rendering: Motivation, taxonomy, and research directions, 2022. 7

[23] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42 (4), 2023. 2, 3, 4

[24] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. 5

[25] Ho Man Kwan, Ge Gao, Fan Zhang, Andy Gower, and David Bull. HineRV: Video compression with hierarchical encoding-based neural representation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. 3

[26] Didier Le Gall. Mpeg: A video compression standard for multimedia applications. *Commun. ACM*, 1991. 2

[27] Jiahao Li, Bin Li, and Yan Lu. Deep contextual video compression. *Advances in Neural Information Processing Systems*, 34, 2021. 2

[28] Jiahao Li, Bin Li, and Yan Lu. Deep contextual video compression. *Advances in Neural Information Processing Systems*, 34, 2021. 2

[29] Zizhang Li, Mengmeng Wang, Huaijin Pi, Kechun Xu, Jianbiao Mei, and Yong Liu. E-nerv: Expedite neural video representation with disentangled spatial-temporal context. In *Computer Vision – ECCV 2022*, pages 267–284, Cham, 2022. Springer Nature Switzerland. 1, 3, 5, 8

[30] Guanxing Lu, Shiyi Zhang, Ziwei Wang, Changliu Liu, Jiwen Lu, and Yansong Tang. Manigaussian: Dynamic gaussian splatting for multi-task robotic manipulation. *arXiv preprint arXiv:2403.08321*, 2024. 6

[31] Alexandre Mercat, Marko Viitanen, and Jarno Vanne. Uvg dataset: 50/120fps 4k sequences for video codec analysis and development. In *Proceedings of the 11th ACM Multimedia Systems Conference*, pages 297–302, 2020. 5

[32] 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. *Communications of the ACM*, 65(1):99–106, 2021. 2, 3

[33] Amirali Molaei, Amirhossein Aminimehr, Armin Tavakoli, Amirhossein Kazerouni, Bobby Azad, Reza Azad, and Dorit Merhof. Implicit neural representation in medical imaging: A comparative survey. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2381–2391, 2023. 3

[34] Wieland Morgenstern, Florian Barthel, Anna Hilsmann, and Peter Eisert. Compact 3d scene representation via self-organizing gaussian grids. *arXiv preprint arXiv:2312.13299*, 2023. 7

[35] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. Cuda, release: 10.2.89, 2020. 5

[36] Makoto Ohki, Michael E. Zervakis, and Anastasios N. Venetsanopoulos. 3-d digital filters. In *Multidimensional Systems: Signal Processing and Modeling Techniques*, pages 49–88. Academic Press, 1995. 8

[37] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 5, 6

[38] Jordi Pont-Tuset, Federico Perazzi, Sergi Caelles, Pablo Arbeláez, Alexander Sorkine-Hornung, and Luc Van Gool. The 2017 davis challenge on video object segmentation. *arXiv:1704.00675*, 2017. 5, 8

[39] Alexander Rhatushnyak, Jan Wassenberg, Jon Sneyers, Jyrki Alakuijala, Lode Vandevenne, Luca Versari, Robert Obryk, Zoltan Szabadka, Evgenii Kliuchnikov, Iulia-Maria Comsa, Krzysztof Potempa, Martin Bruse, Moritz Firsching, Renata Khasanova, Ruud van Asseldonk, Sami Boukortt, Sebastian Gomez, and Thomas Fischbacher. Committee draft of jpeg xl image coding system, 2019. 7

[40] Hochang Rhee, Hae Soo Chung, Jun Ho Jo, Eun Ji Lee, and Nam Ik Cho. Sanerv: Scene-adaptive neural representation for videos. In *2024 IEEE International Conference on Image Processing (ICIP)*, pages 1274–1280. IEEE, 2024. 3

[41] Oren Rippel, Sanjay Nair, Carissa Lew, Steve Branson, Alexander G Anderson, and Lubomir Bourdev. Learned video compression. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3454–3463, 2019. 2

[42] Jens Eirik Saethre, Roberto Azevedo, and Christopher Schroers. Combining frame and gop embeddings for neural video representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9253–9263, 2024. 3

[43] Vishwanath Saragadam, Daniel LeJeune, Jasper Tan, Guha Balakrishnan, Ashok Veeraraghavan, and Richard G Baraniuk. Wire: Wavelet implicit neural representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18507–18516, 2023. 2, 3

[44] Tamar Rott Shaham, Michael Gharbi, Richard Zhang, Eli Shechtman, and Tomer Michaeli. Spatially-adaptive pixelwise networks for fast image translation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14882–14891, 2021. 3

[45] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in neural information processing systems*, 33:7462–7473, 2020. 2, 3

[46] Weronika Smolak-Dyżewska, Dawid Malarz, Kornel Howil, Jan Kaczmarczyk, Marcin Mazur, and Przemysław Spurek. Vegas: Video gaussian splatting, 2024. 2, 3, 5, 8

[47] Gary J Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on circuits and systems for video technology*, 22(12):1649–1668, 2012. 1, 2

[48] Yang-Tian Sun, Yi-Hua Huang, Lin Ma, Xiaoyang Lyu, Yan-Pei Cao, and XIAOJUAN QI. Splatter a video: Video gaussian representation for versatile processing. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. 2, 3, 5, 8

[49] Jinxiang Wang, Yangdong Liu, Shiping Zhu, and Cheng Feng. Temporal enhanced hybrid neural representation for video compression. In *2024 Picture Coding Symposium (PCS)*, pages 1–5. IEEE, 2024. 3

[50] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *NeurIPS*, 2021. 3

[51] T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the h.264/avc video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 2003. 2

[52] Chang Wu, Guancheng Quan, Gang He, Xin-Quan Lai, Yunsong Li, Wenxin Yu, Xianmeng Lin, and Cheng Yang. Qs-nerv: Real-time quality-scalable decoding with neural representation for videos. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 2584–2592, 2024. 3

[53] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20310–20320, 2024. 2, 3, 6, 7

[54] Skylar Wurster, Ran Zhang, and Changxi Zheng. Gabor splatting for high-quality gigapixel image representations. In *ACM SIGGRAPH 2024 Posters*, New York, NY, USA, 2024. Association for Computing Machinery. 8, 7

[55] Yunjie Xu, Xiang Feng, Feiwei Qin, Ruiquan Ge, Yong Peng, and Changmiao Wang. Vq-nerv: A vector quantized neural representation for videos. *arXiv preprint arXiv:2403.12401*, 2024. 3

[56] Hao Yan, Zhihui Ke, Xiaobo Zhou, Tie Qiu, Xidong Shi, and Dadong Jiang. Ds-nerv: Implicit neural video representation with decomposed static and dynamic codes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 23019–23029, 2024. 3

[57] Ziyi Yang, Xinyu Gao, Wen Zhou, Shaohui Jiao, Yuqing Zhang, and Xiaogang Jin. Deformable 3d gaussians for high-fidelity monocular dynamic scene reconstruction. *arXiv preprint arXiv:2309.13101*, 2023. 3, 6

[58] Zeyu Yang, Hongye Yang, Zijie Pan, and Li Zhang. Real-time photorealistic dynamic scene representation and rendering with 4d gaussian splatting. In *International Conference on Learning Representations (ICLR)*, 2024. 2, 3, 4, 7

[59] Xinjie Zhang, Ren Yang, Dailan He, Xingtong Ge, Tongda Xu, Yan Wang, Hongwei Qin, and Jun Zhang. Boosting neural representations for videos with a conditional decoder. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. 3, 5, 6, 8

[60] Xinjie Zhang, Xingtong Ge, Tongda Xu, Dailan He, Yan Wang, Hongwei Qin, Guo Lu, Jing Geng, and Jun Zhang. Gaussianimage: 1000 fps image representation and compression by 2d gaussian splatting. In *European Conference on Computer Vision*, pages 327–345. Springer, 2025. 2, 3, 4, 1

[61] Qi Zhao, M Salman Asif, and Zhan Ma. Pnerv: Enhancing spatial consistency via pyramidal neural representation for videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19103–19112, 2024. 3