# Tree-NeRV: Efficient Non-Uniform Sampling for Neural Video Representation via Tree-Structured Feature Grids

Supplementary Material

## A. Overview

In this supplementary document, we provide additional details to complement our main paper. First, we describe the lower and upper bound query mechanism in Tree-NeRV in Appendix B. Next, we present an in-depth discussion and visualization of the balancing mechanism adopted in Tree-NeRV in Appendix C. We then provide a preliminary review of NeRV in Appendix D and detail the implementation of our experiments in Appendix E. Furthermore, we include additional experimental results to analyze our method in Appendix F. Finally, we provide qualitative comparisons in Appendix G.

## B. Query Mechanism in Tree-NeRV

Tree-NeRV introduces a novel feature representation paradigm by organizing and storing features within a Binary Search Tree (BST), enabling efficient non-uniform sampling and retrieval. In a balanced Tree-NeRV, the query process follows a divide-and-conquer strategy, reducing the search space by half at each step. At each node, the query key is compared with the current node's temporal key, and based on the result, the search proceeds to either the left or right subtree. Since an exact match is rare, the query typically continues until reaching a leaf node, which is the most common case in Tree-NeRV. Consequently, the query time complexity is determined by the height of the tree, yielding $\mathcal{O}(\log n)$ complexity. In contrast, LINKED-LIST-based representations, where features are sequentially stored, require a linear scan to locate the nearest temporal keys. This results in a worst-case query complexity of $\mathcal{O}(n)$, making retrieval significantly slower for large video sequences. To further optimize the search process, we introduce a set of intermediate variables that dynamically store and update the lower and upper bounds during traversal. This ensures that a single query efficiently retrieves both bounds. The query process summarized in Algorithm 1.

**Algorithm 1** Temporal Embedding Query in Tree-NeRV

```
1:  Input: Query time $t_i$, subtree $T$
2:  Output: Time embedding $v_i$
3:  Initialize traversal from the root node $(k, v)$ of $T$
4:  Set $(k_i^l, v_i^l) \leftarrow$ None,   $(k_i^u, v_i^u) \leftarrow$ None   ▷ Initialize bounds
5:  while $T \neq \emptyset$ do
6:      if $t_i = k$ then
7:          return $v_i = v$                       ▷ Exact match found
8:      else if $t_i < k$ then
9:          $(k_i^u, v_i^u) \leftarrow (k, v)$      ▷ Update upper bound
10:         $T \leftarrow T_L$                      ▷ Traverse left subtree
11:     else
12:         $(k_i^l, v_i^l) \leftarrow (k, v)$      ▷ Update lower bound
13:         $T \leftarrow T_R$                      ▷ Traverse right subtree
14:     end if
15: end while
16: $v_i \leftarrow$ Interpolate$(v_i^l, v_i^u, k_i^l, k_i^u, t_i)$   ▷ Linear interpolation
17: return $v_i$
```

## C. Tree-NeRV's balance mechanism

To ensure Tree-NeRV remains balanced and supports efficient query operations, we implement an automatic rebalancing algorithm inspired by the self-balancing mechanism of AVL trees. To formally define balance, we first define the height of each subtree $T$ denoted as $h(T)$, and the height of a subtree is the longest path from its root node to a leaf node, where a leaf node is defined as a node with both left and right subtrees empty, having a height of 0. The height of a subtree $T$ is computed recursively as:

$$h(T) = 1 + \max\{h(T_L), h(T_R)\} \tag{10}$$

Beyond height, we define the balance factor $\beta$ for each subtree $T$ as the difference between the height of its left and right subtrees:

$$\beta = h(T_L) - h(T_R) \tag{11}$$

To maintain Tree-NeRV's balance, we enforce the AVL tree constraint, which requires each $\beta$ to satisfy:

$$-1 \leq \beta \leq 1 \tag{12}$$

any subtree $T$ with $\beta$ exceeds this range, is considered unbalanced, necessitating a rebalancing operation to restore its efficiency.

During training, Tree-NeRV encounters different types of imbalanced states. We categorize these scenarios and apply appropriate rebalancing strategies to restore balance efficiently, as summarized in Tab. 8.

To further illustrate these imbalance conditions, Fig. 8 provides a visual depiction of Tree-NeRV's rotation-based rebalancing operations. These scenarios are categorized based on the balance factor $\beta$, of the affected node and its child node. The corresponding rebalancing operations are

Table 8. Classification of imbalance cases in Tree-NeRV and their corresponding rebalancing operations. Here, $ib$ denotes the imbalanced node, and $ibc$ denotes its child node.

| **Imbalance Node** | **Child Node** | **Rebalancing Operation** |
|---|---|---|
| $\beta_{ib} > 1$ | $\beta_{ibc} \geq 0$ | Right Rotation |
| $\beta_{ib} > 1$ | $\beta_{ibc} < 0$ | Left Rotation $\rightarrow$ Right Rotation |
| $\beta_{ib} < -1$ | $\beta_{ibc} \leq 0$ | Left Rotation |
| $\beta_{ib} < -1$ | $\beta_{ibc} > 0$ | Right Rotation $\rightarrow$ Left Rotation |

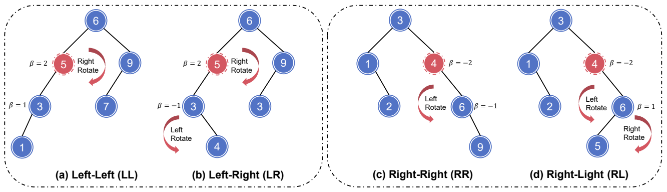


Figure 8. **Right & Left rotation adopted in Tree-NeRV.**

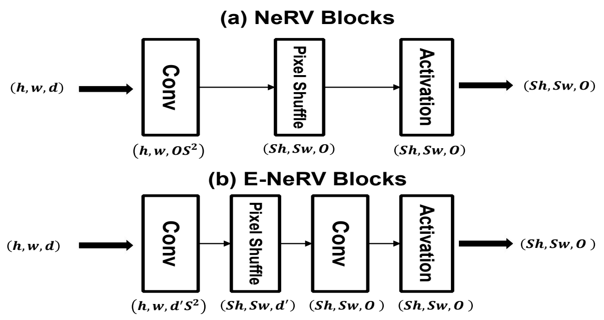


Figure 9. Comparision of NeRV and E-NeRV block.

applied as follows: a) **Left-Left (LL)** Imbalance: The balance factor of node 5 is $\beta_5 = 2$, indicating an imbalance. Its child node 3 has $\beta_3 = 1$, leading to an LL imbalance. A single right rotation at node 5 restores balance. b) **Left-Right (LR)** Imbalance: Similar to case (a), $\beta_5 = 2$, causing an imbalance. However, its child node 3 now has $\beta_3 = -1$, forming an LR imbalance. To restore balance, we first apply a left rotation at node 3, followed by a right rotation at node 5. c) **Right-Right (RR)** Imbalance: The balance factor of node 4 is $\beta_4 = -2$, marking it as unbalanced. Its child node 6 has $\beta_6 = -1$, leading to an RR imbalance. A single left rotation at node 4 restores balance. (d) **Right-Left (RL)** Imbalance: Similar to case (c), $\beta_4 = -2$, causing an imbalance. However, its child node 6 now has $\beta_6 = 1$, forming an RL imbalance. To restore balance, we first apply a right rotation at node 6, followed by a left rotation at node 4.

## D. NeRV

Neural Representations for Videos (NeRV) [11] is an Implicit Neural Representation (INR) framework designed for efficient video modeling. Unlike conventional INR methods that map spatiotemporal coordinates to pixel values, NeRV directly learns a frame-index-to-frame mapping, enabling fast and compact video representations. Given an RGB video sequence $\mathbb{V} = \{x_t\}_{t=0}^{T-1}$, where each frame $x_t \in \mathbb{R}^{3\times H\times W}$, NeRV formulates its mapping as:

$$x_t = f(\gamma(t)), \tag{13}$$

where $f : \mathbb{R}^t \rightarrow \mathbb{R}^{3\times H\times W}$ is the learnable function, and $\gamma(t)$ is an embedding function that encodes the frame index $t$ into a high-dimensional space. The function $f$ is typically parameterized as a cascade of convolution-based NeRV blocks, which progressively upsample and refine feature representations.

As illustrated in Fig. 9, each NeRV block consists of: 1) A convolutional layer to extract and transform features. 2) A pixel-shuffle upsampling operation [34] to increase spatial resolution. 3) An activation function (e.g., ReLU [16], GELU [17]) to introduce non-linearity. Given an input feature of shape $h \times w \times d$, NeRV aims to upsample it by a factor of $S$. The standard NeRV pipeline follows:

$$conv_{3\times3}(d, OS^2) \rightarrow \text{pixel-shuffle}(S). \tag{14}$$

The number of trainable parameters in a single NeRV block is $3 \times 3 \times O \times d$.

To reduce parameter redundancy while maintaining performance, E-NeRV introduces an intermediate projection dimension $d'S^2$, modifying the block structure as:

$$conv_{3\times3}(d, d'S^2) \rightarrow \text{pixel-shuffle}(S) \rightarrow conv_{3\times3}(d', O). \tag{15}$$

The total parameter count in an E-NeRV block is given by: $3\times3\times d'S^2\times(d\times S^2+O)$. By selecting a smaller intermediate channel dimension $d'$, E-NeRV significantly reduces parameters while preserving spatial reconstruction quality.

While E-NeRV achieves substantial parameter savings, we observe that the expressiveness of NeRV blocks remains positively correlated with their parameter count. Excessive parameter reduction can degrade video reconstruction quality. To balance efficiency and performance, we adopt a hybrid design: 1) E-NeRV blocks are applied only in the first NeRV block, which requires the largest upsampling factor (e.g., $5\times$ scaling) and has the highest intermediate channel dimension. 2) tandard NeRV blocks are retained for all subsequent layers, preserving feature expressiveness while maintaining computational efficiency.

## E. Experimental Setup

### E.1. Implementation Details

**Baseline Implementation:** For HNeRV [12], and FFNeRV [21], we conducted experiments using their publicly available implementations. FFNeRV adopts a multi-resolution feature grid, which we implemented following the original paper, using resolutions of [64, 128, 256, 512]. We controlled the parameter budget by adjusting the feature dimensions accordingly. For DS-NeRV [43], we developed our own implementation based on the open-source code of FFNeRV. Following the original work, we utilized varying numbers of static codes ($\sim$ 30–100) and dynamic codes ($\sim$ 150–400) to match their settings. Notably, both HNeRV and DS-NeRV downscale video resolution to a fixed aspect ratio of 1:2 (i.e., height:width). However, in our experiments, we maintain the original 9:16 resolution for all video frames. To ensure a fair comparison, we adjusted their feature sizes accordingly, aligning with other methods in our evaluation.

**Tree-NeRV Configuration:** In our implementation, we adjusted the number of channels in the latent features and NeRV blocks to control the model size, while keeping other hyperparameters consistent with the settings reported in the original papers.

For example, when processing a $1080 \times 1920 \times 3 \times 600$ UVG [28] video sequence, we adopted a uniform sampling rate of $0.1$, resulting in 60 initial features. These features were iteratively grown in four stages, with the top 10 GOPs being inserted with new nodes during each stage. Each feature code was represented as a $9 \times 16 \times 100$ three-dimensional vector. The NeRV blocks used stride steps of $5, 3, 2, 2, 2$, and the minimum number of channels was set to 72. Under this configuration, the total number of parameters amounted to approximately 3M. For the DAVIS [37] dataset, which contains videos with fewer frames and higher dynamic variations, we adjusted only the number of initial features to 10 and increased the feature dimension to $9 \times 16 \times 120$. All other settings were kept consistent with those used for the UVG dataset.

| Video | size | resolution | $h_s \times w_s \times dim_s$ | init feature | topk | $Ch_{min}$ | strides |
|---|---|---|---|---|---|---|---|
| Beauty | 3 | $1080 \times 1920$ | $9 \times 16 \times 100$ | 60 | 10 | 64 | (5,3,2,2,2) |
| Bosph | 3 | $1080 \times 1920$ | $9 \times 16 \times 100$ | 60 | 10 | 64 | (5,3,2,2,2) |
| Honey | 3 | $1080 \times 1920$ | $9 \times 16 \times 100$ | 60 | 10 | 64 | (5,3,2,2,2) |
| Yacht | 3 | $1080 \times 1920$ | $9 \times 16 \times 100$ | 60 | 10 | 64 | (5,3,2,2,2) |
| Ready | 3 | $1080 \times 1920$ | $9 \times 16 \times 100$ | 60 | 10 | 64 | (5,3,2,2,2) |
| Jockey | 3 | $1080 \times 1920$ | $9 \times 16 \times 100$ | 60 | 10 | 64 | (5,3,2,2,2) |
| Shake | 3 | $1080 \times 1920$ | $9 \times 16 \times 100$ | 30 | 20 | 64 | (5,3,2,2,2) |

Table 9. Architecture details of Tree-NeRV on UVG.

| Video | size | resolution | $h_s \times w_s \times dim_s$ | init feature | topk | $Ch_{min}$ | strides |
|---|---|---|---|---|---|---|---|
| Blackswan | 3 | $1080 \times 1920$ | $9 \times 16 \times 120$ | 10 | 10 | 72 | (5,3,2,2,2) |
| Bmx-trees | 3 | $1080 \times 1920$ | $9 \times 16 \times 120$ | 10 | 10 | 72 | (5,3,2,2,2) |
| Boat | 3 | $1080 \times 1920$ | $9 \times 16 \times 120$ | 10 | 10 | 72 | (5,3,2,2,2) |
| Breakdance | 3 | $1080 \times 1920$ | $9 \times 16 \times 120$ | 10 | 10 | 72 | (5,3,2,2,2) |
| Camel | 3 | $1080 \times 1920$ | $9 \times 16 \times 120$ | 10 | 10 | 72 | (5,3,2,2,2) |
| Car-roundabout | 3 | $1080 \times 1920$ | $9 \times 16 \times 120$ | 10 | 10 | 72 | (5,3,2,2,2) |
| Car-shadow | 3 | $1080 \times 1920$ | $9 \times 16 \times 120$ | 10 | 10 | 72 | (5,3,2,2,2) |
| Cows | 3 | $1080 \times 1920$ | $9 \times 16 \times 120$ | 10 | 10 | 72 | (5,3,2,2,2) |
| Dance | 3 | $1080 \times 1920$ | $9 \times 16 \times 120$ | 10 | 10 | 72 | (5,3,2,2,2) |
| Dog | 3 | $1080 \times 1920$ | $9 \times 16 \times 120$ | 10 | 10 | 72 | (5,3,2,2,2) |

Table 10. Architecture details of Tree-NeRV on Davis.

| Topk | 5 | 15 | 20 | 10 |
|---|---|---|---|---|
| PSNR | 33.21 | 33.30 | 33.32 | **33.36** |

Table 11. Ablation study for feature length on UVG. Ours sampled 160 feature points after training.

## F. Additional Experiments

### F.1. Topk Selection

To further investigate Tree-NeRV, we conducted an additional ablation study. First, we evaluated the effect of different Top-$k$ values on Tree-NeRV's sampling behavior and compression performance. Specifically, we tested $k$ values of 5, 10, 15, and 20, analyzing Tree-NeRV's reconstruction results on the UVG dataset. As shown in Tab. 11, similar compression performance was achieved across the different $k$ values. Additionally, in Fig. 10, we visualized the actual sampling outcomes of Tree-NeRV for each $k$ setting, observing a consistent sampling trend across the different configurations.

### F.2. Video Compression

Our compression pipeline follows a standard three-step process: global parameter pruning, quantization, and entropy-based encoding. Table 12 presents the impact of these compression techniques on the final results. Moving forward, we aim to integrate more advanced compression strategies into NeRV-like approaches to further optimize efficiency. Additionally, we plan to explore node pruning as a mechanism for reducing video stream redundancy.

### F.3. Perceptional Quality Comparison

In the field of compression, a widely recognized trade-off exists between 'rate-distortion-realism' [7]. Given that Tree-NeRV is fully trained using the Mean Squared Error (MSE) loss, we aim to evaluate its performance not only in terms of distortion but also in perceptual realism. To this end, we adopt the Learned Perceptual Image Patch Similarity (LPIPS) [47] metric to assess the perceptual quality of Tree-NeRV on the UVG and Davis dataset. The results are shown in Tab. 13.

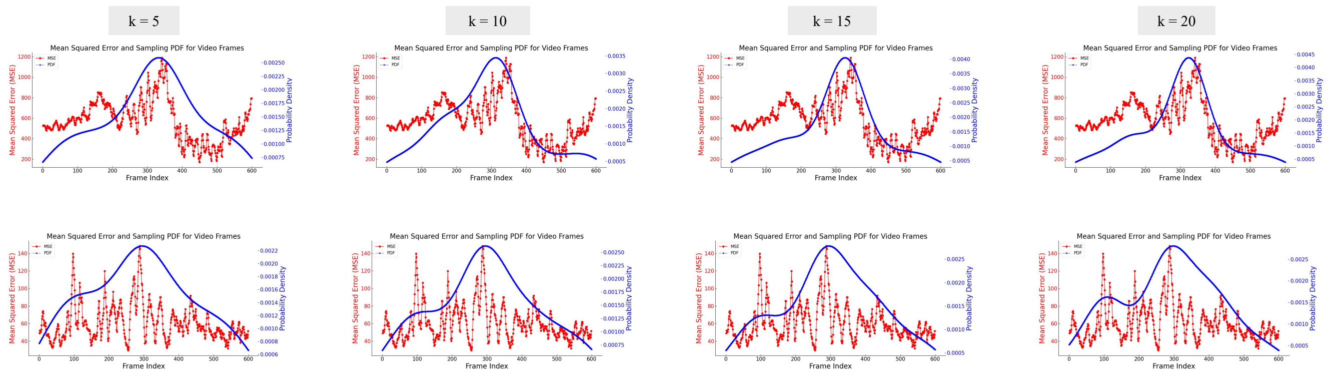


Figure 10. **Tree-NeRV sampling results under different setting of topk, on Jockey (top), Beauty (bottom).**

| UVG | Beauty | Bospho | Honey | Jockey | Ready | Shake | Yacht |
|---|---|---|---|---|---|---|---|
| N/A | 33.54/0.920 | 35.63/0.965 | 39.88/0.990 | 32.74/0.912 | 26.86/0.861 | 35.28/0.953 | 29.74/0.908 |
| 8-bit Quant | 33.48/0.919 | 35.55/0.965 | 39.86/0.989 | 32.71/0.912 | 26.79/0.860 | 35.27/0.953 | 29.68/0.908 |
| 8-bit Quant + Pruning (10%) | 33.13/0.916 | 35.27/0.964 | 39.15/0.989 | 32.08/0.911 | 26.42/0.859 | 35.04/0.953 | 29.44/0.908 |

Table 12. Compression ablations on UVG in PSNR/SSIM.

| Video | Bosph | Honey | Shake | b-dance | b-swan | c-shadow | Avg. |
|---|---|---|---|---|---|---|---|
| HNerv [12] | 0.335±0.009 | 0.199±0.004 | 0.242±0.018 | 0.228±0.007 | 0.367±0.006 | 0.334±0.012 | 0.284±0.009 |
| **Ours** | **0.283±0.012** | **0.194±0.005** | **0.241±0.018** | **0.168±0.006** | **0.291±0.008** | **0.263±0.012** | **0.240±0.01** |

Table 13. Additional LPIPS (↓) results with both method set to 3M parameters.

## G. Additional Qualitative Results

### G.1. Visualization of Video Representation

We present additional qualitative comparisons of video representation on the UVG and DAVIS datasets. Tree-NeRV consistently demonstrates superior reconstruction quality. For example, in Fig. 11, the first row highlights the circular rings on the boat, while the second row shows detailed high-frequency variations in the background. The third row captures splashing water, and the fourth row restores the numbers on the scoreboard. In Fig. 13, Tree-NeRV outperforms other methods in reconstructing graffiti on the wall (first row), background architecture (second row), and the detailed textures on the camel (third row). As shown in Fig. 12, these improvements are attributed to the tree-structured feature representation and our adaptive sampling strategy, which effectively captures the temporal redundancy in video streams.

### G.2. Visualization of Video Interpolation

Additional visual comparisons of video interpolation results are available in Fig. 14. Tree-NeRV successfully preserves intricate details in previously unseen frames, demonstrating its superior interpolation capabilities.

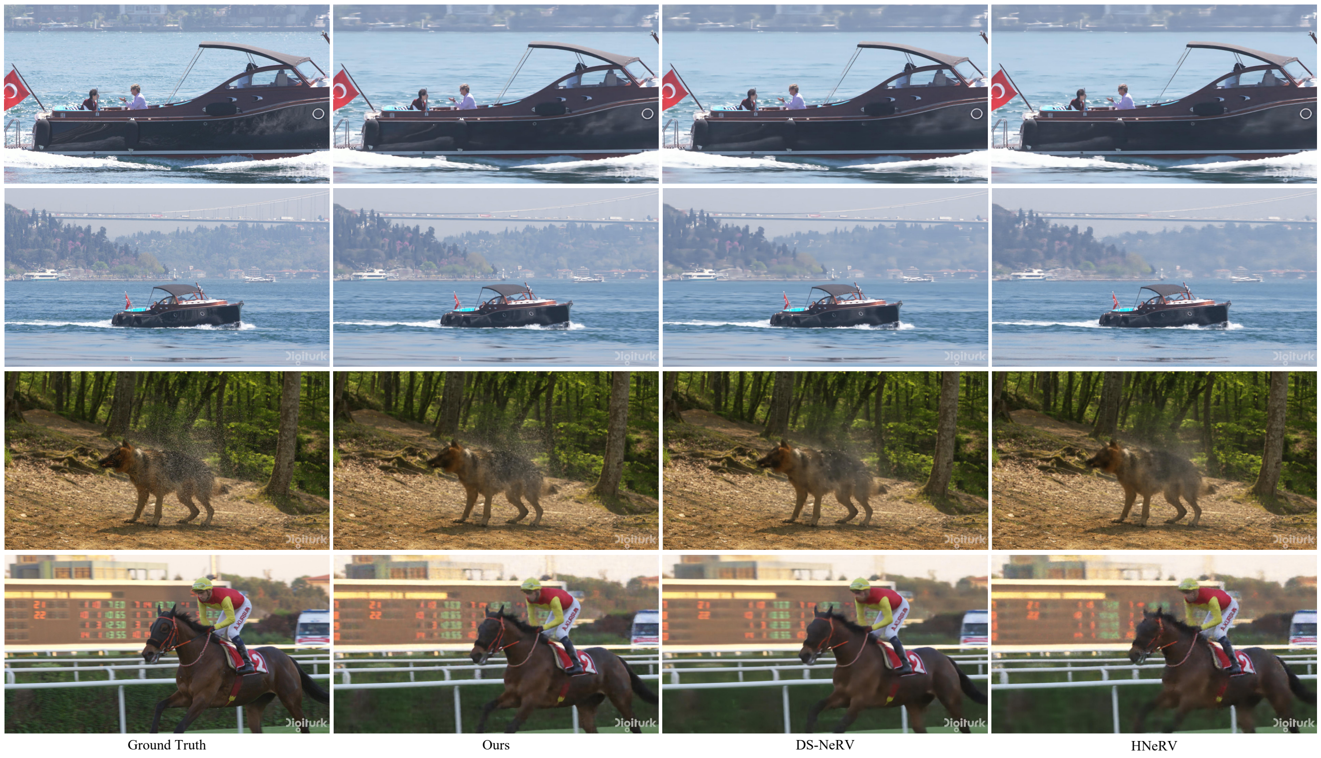
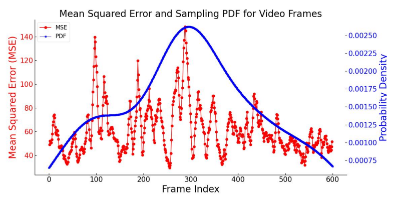
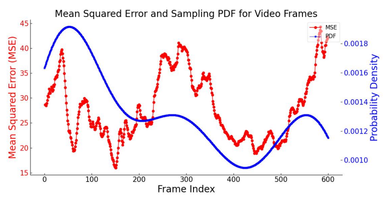


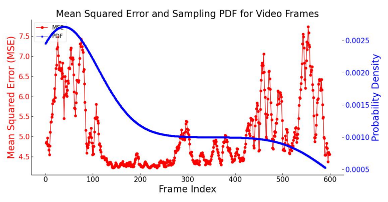
Figure 11. **Additional video reconstruction results on UVG.**



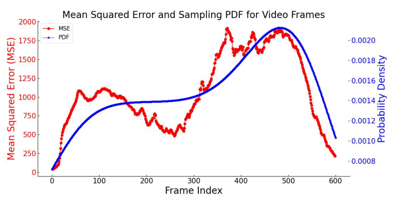
Beauty



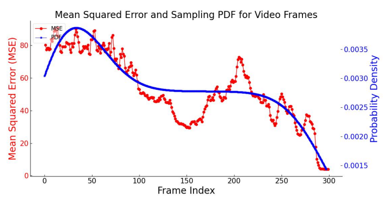
Bosphorus



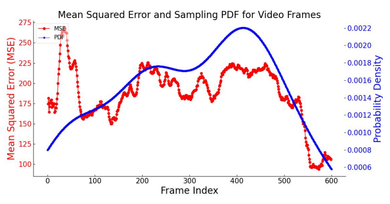
Honeybee



Ready



Shake



Yacht

Figure 12. **Additional Tree-NeRV sampling results on UVG.**

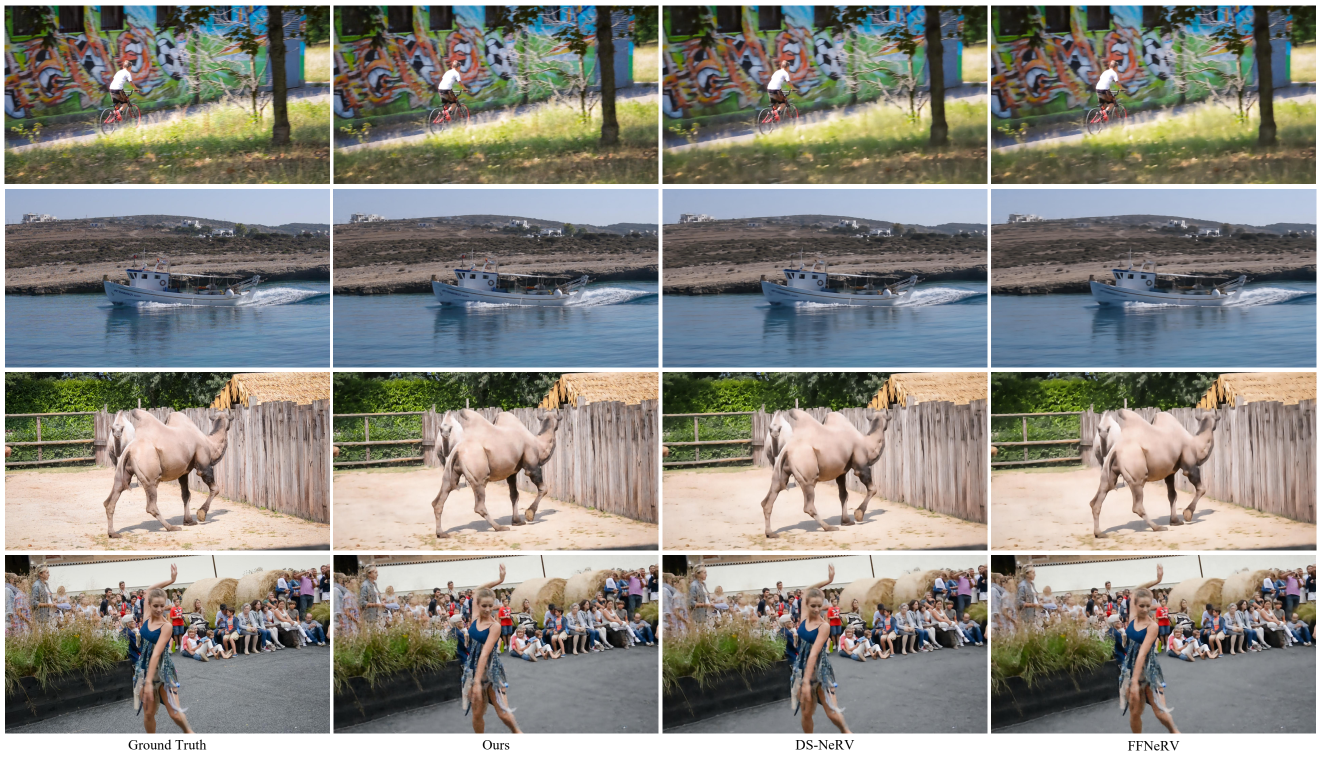


Figure 13. **Additional video reconstruction results on DAVIS.**

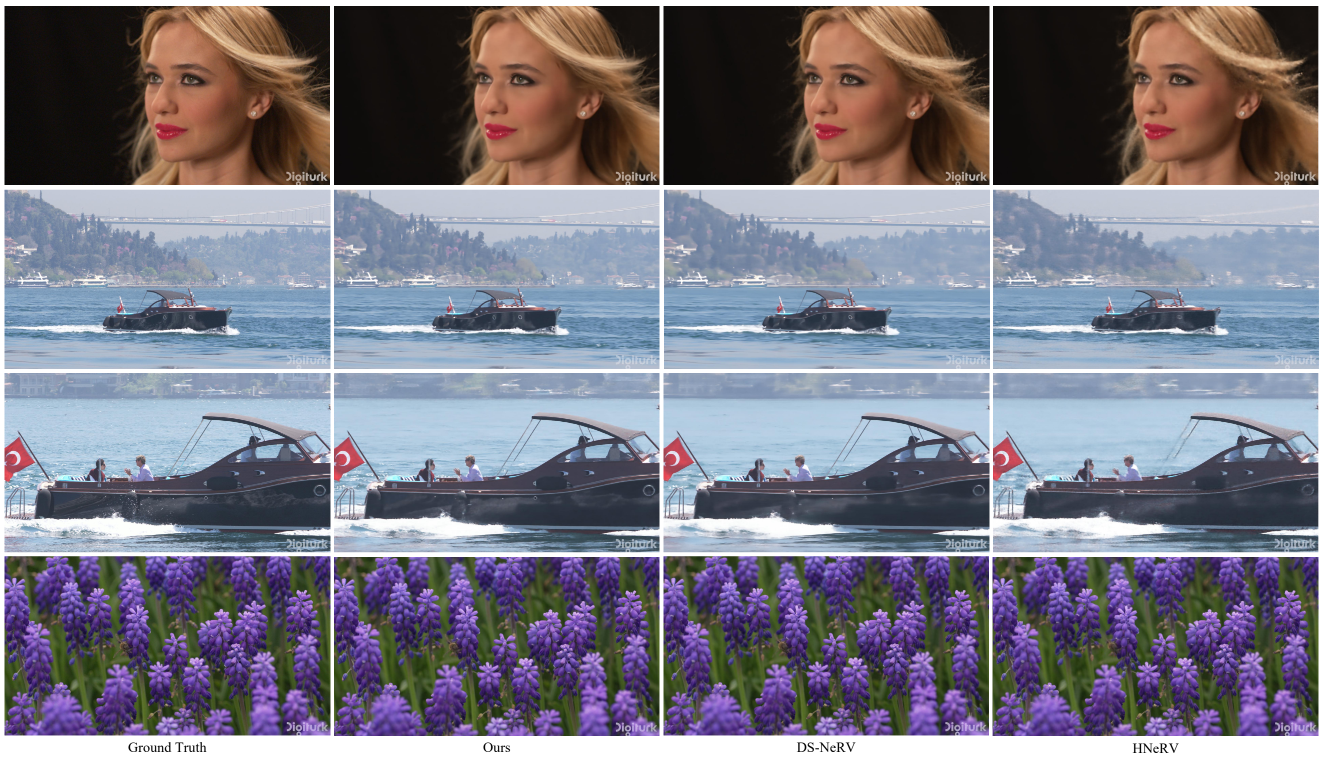


Figure 14. **Additional video interpolation results on UVG.**