PackUV: Packed Gaussian UV Maps for 4D Volumetric Video

Supplementary Material

1. PackUV-2B Dataset

We captured PackUV-2B, a novel real-world dataset featuring long-horizon, multi-view video sequences. PackUV-2B comprises 100 such sequences, with over 2B (billion) frames in total, and encompasses a diverse array of scenarios, including human-human interaction, human-object interaction, robot-object interaction, among others. The sequences in PackUV-2B average approximately 10 minutes in length, with some extending up to 30 minutes. Notably, to establish PackUV-2B as a comprehensive benchmark for evaluating dynamic reconstruction approaches, we have curated sequences of varying difficulty levels, aiming to cover a broad spectrum of real-world settings. For instance, in terms of motion speed, PackUV-2B includes movements ranging from slow actions like "inching along" to fast-paced activities such as playing basketball, thereby posing challenges for handling motion blur. Regarding motion scale, PackUV-2B captures both small-scale interactions, like table-top object manipulation, and large-scale activities, such as dance, pickle ball, volleyball, etc. Furthermore, PackUV-2B features diverse object categories, including rigid and articulated objects, as well as some reflective and transparent items. To the best of our knowledge, PackUV-2B significantly surpasses existing datasets in its domain concerning sequence length, number of camera views, motion complexity, dynamic diversity, and overall data volume. More details about the captured sequences are presented in Table 3. The Capture system is discussed in Supp. Sec. 7.

Table 1. PSNR comparison with the baselines on N3DV (*flame_salmon*), DeskGames, and Technicolor datasets.

Dataset	Motion L.	4DGS	Ex4DGS	3DGStream	GIFStream	Ours
N3DV	32.55	32.01	32.11	31.67	28.42	33.06
DeskGames	30.89	30.11	29.48	30.51	29.94	32.74
TechniColor	31.77	28.91	31.33	25.48	27.42	31.87

2. Additional Qualitative and Quantitative Results

More qualitative results are shown in figures 5, 7, 6, 8 at the end of supplementary. From the results, it can be seen that our method, PackUV, significantly outperforms other methods in reconstructing large motions and disocclusions while maintaining the overall quality.

We also evaluate on three additional datasets, including one more sequence from N3DV, Technicolor, and the DeskGames dataset from Motion Layering. A summarized comparison is shown in Tab. 1. Our method performs better across datasets. Metrics are collected from Motion Layering except ours and GIFStream for the same experimental setup.

3. Lossy Post-Optimization UV Mapping for Scenes

Fig. 2 show lossy reconstruction via UVGS [5] mapping for a real-world scene. Even when using 48 layers and 1K resolution of UV maps, there are missing Gaussians resulting in clearly visible artifacts in the renderings. This clearly illustrates the importance of UV mapping during optimizing the Gaussians as done in PackUV.

Table 2. Storage comparison with the baselines (30 frames).

Method	LG	Grid4D	Ex4DGS	3DGStream	GIFStream	Ours
Storage (MB)	950	76	108	204	16	10

4. Video Coding and PackUV Storage

PackUV allow easy and **lossless** encoding of Volumetric videos using standard 2D video codecs while preserving quality of the 4D scene. Because PackUV-GS maintains temporally consistent UV layouts and only updates per-pixel attributes over time for dynamic regions, the atlas sequence exhibits *strong spatial locality* and *temporal coherence*, allowing direct reuse of mature video coding pipelines. This solved one of the major drawbacks of streaming based methods like GIFStream, 3DGStream and AT-GS - high storage requirements of the trained models for every timestamp for lossless conversion. Using the FFV1 codec for lossless video conversion is giving us an average storage rate of under **10 MBPS**. Tab. 2 compares storage with SOTA 4DGS methods. Our method outperforms all volumetric video streaming baselines and significantly surpasses specialized per-frame compression methods such as LG [1].

We analyze the compression of PackUV atlases via different techniques including quantization of individual 3DGS attributes, using different video and image compression techniques, and mix of both. The results are presented in Fig. 1(left). FFV1 lossless compression with 8-bit LPO achieves the highest compression ratio with only a **0.11 dB** PSNR drop.

Another advantage of PackUV is it's structural arrangement, which can be used for mapping the scene to a latent space to achieve neural compression [5, 7]. Unlike

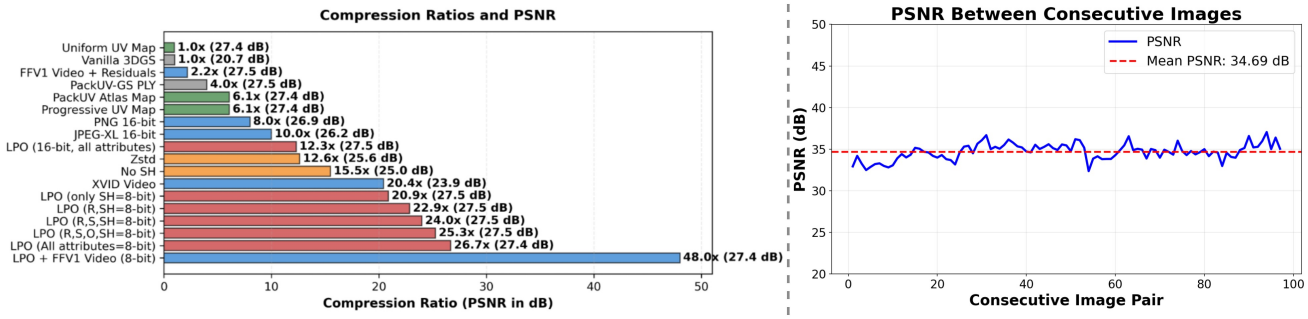

Figure 1. (Left) Compression evaluation via different methods. (Right) PSNR consistency over time.

Figure 2. Post-optimization lossy UV mapping fails to capture details of a real-world scene even with 48 layers and 1K resolution.

UVGS [5], GaussianAtlas [7], PackUV allows us to represent the entire scene into a single multiscale atlas. This further reduces the complexity of neural networks required to compress it to a latent space. The objective is to compress a single-layer UV atlas representation—which encodes full PackUV atlas to a compact representation that is both easy to store and invertible back to a 4D scene representation. Since the UV format structurally organizes scene attributes in an image-like format, we directly adopt image-based compression pipelines.

4.1. Neural Compression

Neural Compression is a widely used technique for compressing various modalities [3, 5]. We consider a neural network architecture for compressing PackUV inspired from UVGS [5]. We follow the designing by using three lightweight encoder-decoder networks, each responsible for compressing a specific modality:

- **Position Encoder-Decoder:** maps spatial information $\sigma_i \in \mathbb{R}^{3\times H_0\times W_0}$
- **Appearance Encoder-Decoder:** maps RGB/SH color and opacity $\{c_i, o_i\} \in \mathbb{R}^{4-45\times H_0\times W_0}$
- **Covariance Encoder-Decoder:** maps scale and rotation $\{s_i, r_i\} \in \mathbb{R}^{7\times H_0\times W_0}$

Each modality is encoded into a 3-channel latent image per frame. Let $L^t \in \mathbb{R}^{M\times N\times 3}$ be the per-frame latent representation at time t. These latent images are stored along with the shared decoder weights for the entire scene. This

approach offers a consistent compression ratio ranging from **[8:1]** to **[128:1]** at the cost of minimal degradation (**PSNR drop: [1.5] dB** to **PSNR drop: [4.5] dB**). And the decoding process is very efficient due to the light-weight MLP based decoder.

5. PackUV-GS Optimization Details

5.1. Gaussian Labeling:

We use RAFT [6] is used to estimate the optical flow for each set of images. Flow masks are used to label the Gaussians as dynamic or static and only dynamic Gaussians are optimized during the training. We implemented a custom CUDA kernel to expedite this. The process is Explained in Algorithm 1. As a fallback mechanism, we additionally support a simple point-projection strategy that relies solely on the Gaussian centers, without accounting for their full covariance. In the setting, even when the flow-based masking module is disabled, we observe that training the sequence using a keyframing strategy remains stable and temporally coherent. Specifically, the 3D Gaussians for each new keyframe are initialized using the optimized Gaussian parameters from the preceding keyframe. The same is done for the transition frames. This warm-start initialization propagates geometry and appearance across time, effectively enforcing temporal continuity and preserving structural consistency between consecutive segments, despite the absence of explicit motion-aware supervision. We believe, this is possible by fact that PackUV focus learning on surface-relevant regions and most of them remain static during training of the subsequent timestamps.

5.2. Gradient thresholding:

A common challenge in the existing dynamic 3D reconstruction methods is the uncontrolled growth of the number of Gaussian primitives over time. This progressive accumulation leads to excessive memory consumption and frequent out-of-memory (OOM) errors, especially when fitting long video sequences. To mitigate this issue, we introduce a *gradient thresholding* mechanism that constrains the total number of Gaussians allowed during optimization. The key idea is to retain only those Gaussians that meaningfully contribute to the reconstruction objective while pruning away redundant or inactive ones.

Let the loss function be denoted by $\mathcal{L}$, and consider a Gaussian primitive g_i. We compute the gradient norm of its contribution to the loss:

$$\|\nabla_{g_i}\mathcal{L}\|_2$$

If this norm falls below a predefined threshold τ, the Gaussian is considered non-contributory and is removed from the scene representation. Formally:

$$\text{If } \|\nabla_{g_i}\mathcal{L}\|_2 > \tau, \quad \text{then } g_i \text{ is pruned}$$

This strategy ensures that only actively contributing Gaussians are retained during optimization, effectively capping the total number of primitives and maintaining computational efficiency. By doing so, we avoid unnecessary memory overhead and preserve high-quality reconstruction over long temporal horizons.

5.3. Video Keyframing

To efficiently fit given synchronized multi-view videos with T frames, represented as a set $\{V_n\}_{n=1}^{N}$, where N is the total number of views. We divide each video into s set of m temporal segments. To do so, for each frame t, we compute the optical-flow magnitude $M(t)$ on one video, select the top $(m-1)$ magnitude peaks with a minimum separation θ, and use the first frame of every segment as a keyframe. These keyframes define the segment boundaries. For each keyframe F_i^K, the PackUV Gaussians are initialized from the previous keyframe which preserves temporal and spatial consistency. The frames between keyframes are treated as transition frames. Each transition frame F^t is initialized from the preceding frame and refined with a few training iterations compared to F^K:

$$\mathcal{G}(K) \leftarrow \text{Update}(\mathcal{G}(K-1)), \mathcal{G}(t) \leftarrow \text{Update}(\mathcal{G}(t-1)).$$

This staged, stream-based strategy enables efficient reconstruction of high-fidelity dynamic scenes while allowing us to parallelize the fitting process. Frames exhibiting high drift, occlusions/disocclusions, or appearance breaks are promoted to keyframes. This keyframing technique helps us handle arbitrary length sequences, large motions, and disocclusions without quality degradation over time.

Through our experiments we observe that keeping keyframes farther apart can cause the quality to degrade over time, thus we keep the keyframe threshold to 30 timestamps.

5.4. Low-Precision Training

Unlike most prior works that employ lossy quantization after optimization for storage efficiency, PackUV-GS utilizes low precision optimization (LPO) for learning the Gaussian attributes $\theta = \{\mu, \mathbf{s}, \mathbf{r}, \alpha, \mathbf{c}\}$. Our experiments show that LPO effectively compensates for the quantization loss, maintaining both PSNR and training efficiency. At each iteration, the renderer operates on a quantized proxy $\tilde{\theta}$ obtained by a uniform K-bit fake quantizer with scale Δ. We use a straight-through estimator (STE) to preserve gradient flow while keeping master weights in FP32. The training objective remains the same, combining an L1+SSIM photometric term with a scheduled regularizers. Backpropagation updates FP32 master parameters with Sparse Adam optimizer [2, 4].

We use 8-bit reduced precision for $\{\mathbf{s}, \mathbf{r}, \alpha, \mathbf{c}\}$ and 16-bit for $\{\mathbf{x}\}$. During storage, the 16-bit $\{\mathbf{x}\}$ values are split

Table 3. **PackUV-2B Dataset Layout.** We present our newly captured dataset PackUV-2B sequences . We report number of sequences captured, total timestamps (T), total cameras used to capture the sequence, FPS, setting type, and view range. We also report special tags for each dataset describing the type of activity in the captured sequence. PackUV-2B contains 100 diverse sequences totaling over 2B (billion) high-quality frames, recorded with more than 50 cameras at 1920×1200 resolution. The capture system supports up to 90 FPS, providing high temporal fidelity. Tags: RI - Robot Interaction, HI - Human-human Interaction, OI - Object Interaction, SP - Sports, LM - Large Motion, DO - Disocclusion, TR - Transparent/Reflective Objects, EN - Entertainment .

Sequence	Num Sequences	Num Cameras	FPS	Setting	Tags	View Range
Baby Dance	3	88	30	Studio	EN, HI, DO	360
Spot	3	88	30, 90	Studio	RI, HI, OI, DO	360
Volleyball	3	88	30, 90	Studio	SP, HI, LM, DO	360
Kitchen	3	55	30	Non-studio	OI	320
Woodwork	4	55	30	Non-studio	OI	320
Pickleball	30	55	30	Non-studio	SP, HI, LM	300
Meat Shop	11	84	30	Non-studio	OI, DO	320
Kuka Robot	2	84	30	Non-studio	RI, OI	300
Panda Robot	3	84	30	Non-studio	RI, OI	300
Articulation	2	84	30	Studio	HI, OI, DO	360
Chair Play	1	86	30	Studio	HI, OI, DO, LM	360
Object Placement	1	88	30	Studio	HI, OI, DO, LM	360
Dance 01	3	80	30	Studio	HI, DO, LM, EN	360
Dance 02	7	84	30, 60, 90	Studio	HI, DO, LM, EN	360
Dance 03	3	82	30, 60, 90	Studio	DO, LM, EN	360
Dance 04	3	85	30	Studio	DO, LM, EN	360
Yoga	3	78	30	Studio	LM, SP	360
Tools Play	2	82	30	Studio	LM, DO, OI	360
Board Games	4	86	30	Studio	HI, OI, LM, DO	360
Photography	4	88	30	Studio	HI, OI, LM, DO	360
Conversation	5	84	30	Studio	HI, LM, DO	360
PackUV-2B (TOTAL)	**100**	**55–88**	30-90	-	-	**∼360**

into two 8-bit parts. This 8-bit-per-channel design makes PackUV readily compatible existing video coding infrastructures, enabling the direct application of both lossless or lossy compression methods (*e.g.*, HEVC, AVC, FFV1). We also conduct several experiments to evaluate the loss incurred from reduced-precision training using different levels of bit quantization. From these experiments, we observe that that quantizing the position parameters ($\{\mathbf{x}\}$) cause the largest PSNR drop during training. Consequently, we retain them in FP16, which introduces negligible error during optimization. In contrast, the other attributes ($\{\mathbf{s}, \mathbf{r}, \alpha, \mathbf{c}\}$) show no noticeable PSNR drop in our experiments, even when quantized to 8-bit integers after appropriate scaling.

6. Viewer

The viewer renders a dynamic 4D Gaussian scene as a time-ordered sequence of Gaussian frames. At time step t_k, the scene is represented as

$$\mathcal{G}(t_k) = \{g_i^{(k)}\}_{i=1}^{N_k},$$

where each Gaussian $g_i^{(k)}$ stores a 3D mean $\boldsymbol{\mu}_i$, covariance $\boldsymbol{\Sigma}_i$, opacity α_i, and appearance parameters (e.g., SH coefficients). A stream manager loads PackUV-encoded frame data sequentially, maintains a small look-ahead cache, and exposes both CPU-side Gaussian data and GPU-ready packed metadata for the current frame.

During playback, the active frame index is advanced ac-

Algorithm 1 Covariance-Aware Flow Masking with CUDA Acceleration

Require: Gaussian parameters $\{\boldsymbol{\mu}_i, \mathbf{s}_i, \mathbf{q}_i\}_{i=1}^{N}$, camera views $\mathcal{C}$, flow masks $\{M^c\}_{c\in\mathcal{C}}$
Ensure: Dynamic mask $D_i \in \{0, 1\}$ for each Gaussian i

Initialize $D_i \leftarrow 0$ for all $i = 1, \ldots, N$
for each camera $c \in \mathcal{C}$ **do**
  CUDA Kernel (parallel over all Gaussians):
  for each Gaussian i **in parallel do**
    // 1. Compute 3D covariance and transform to camera space
    $\Sigma_i^{3D} \leftarrow \mathbf{R}(\mathbf{q}_i) \cdot \mathrm{diag}(\mathbf{s}_i^2) \cdot \mathbf{R}(\mathbf{q}_i)^\top$
    $\Sigma_{i,\text{cam}}^{3D} \leftarrow \mathbf{T}_c \cdot \Sigma_i^{3D} \cdot \mathbf{T}_c^\top$
    $\boldsymbol{\mu}_{i,\text{cam}} \leftarrow \mathbf{T}_c \cdot \boldsymbol{\mu}_i$

    // 2. Project to 2D using EWA splatting
    Compute Jacobian: $\mathbf{J} = \begin{bmatrix} f_x/z & 0 & -f_x x/z^2 \\ 0 & f_y/z & -f_y y/z^2 \end{bmatrix}$ at $\boldsymbol{\mu}_{i,\text{cam}}$
    $\Sigma_{i,c}^{2D} \leftarrow \mathbf{J} \cdot \Sigma_{i,\text{cam}}^{3D} \cdot \mathbf{J}^\top$
    $\Sigma_{i,c}^{2D}[0,0] \leftarrow \Sigma_{i,c}^{2D}[0,0] + 0.3$ ▷ Low-pass filter
    $\Sigma_{i,c}^{2D}[1,1] \leftarrow \Sigma_{i,c}^{2D}[1,1] + 0.3$
    $\mathbf{m}_{i,c} \leftarrow \text{NDC2Pixel}(\mathbf{P}_c \cdot \boldsymbol{\mu}_i)$ ▷ Project mean to pixels

    // 3. Check covariance validity
    $\det \leftarrow \Sigma_{i,c}^{2D}[0,0] \cdot \Sigma_{i,c}^{2D}[1,1] - \Sigma_{i,c}^{2D}[0,1]^2$
    if $\det \le 10^{-7}$ **or** $\det < 0$ **or** $z \le 0$ **then**
      $D_{i,c} \leftarrow 0$ ▷ Invalid, mark static
      continue
    end if

    // 4. Compute sampling radius from eigenvalues
    $\mathrm{tr} \leftarrow \Sigma_{i,c}^{2D}[0,0] + \Sigma_{i,c}^{2D}[1,1]$
    $\lambda_{\max} \leftarrow \frac{\mathrm{tr} + \sqrt{\max(0, \mathrm{tr}^2 - 4\cdot\det)}}{2}$
    $r \leftarrow \min\left(r_{\max}, \left\lceil 3\sqrt{\lambda_{\max}} \right\rceil\right)$ ▷ 3-sigma, clamped

    // 5. Test ellipse overlap with flow mask
    $D_{i,c} \leftarrow 0$
    for $\mathbf{p} \in \{\mathbf{m}_{i,c} + \boldsymbol{\delta} \mid \|\boldsymbol{\delta}\|_\infty \le r\}$ **do**
      if $\mathbf{p}$ within image bounds **then**
        $d^2 \leftarrow (\mathbf{p} - \mathbf{m}_{i,c})^\top (\Sigma_{i,c}^{2D})^{-1} (\mathbf{p} - \mathbf{m}_{i,c})$
        if $d^2 \le 9$ **and** $M^c(\mathbf{p}) > 0.5$ **then**
          $D_{i,c} \leftarrow 1$
          break
        end if
      end if
    end for
  end for

  // Aggregate on host (OR across cameras)
  for each Gaussian i **do**
    $D_i \leftarrow D_i \vee D_{i,c}$
  end for
end for
return $\{D_i\}_{i=1}^{N}$

cording to the target frame rate,

$$k(t) = \lfloor ft \rfloor \bmod N,$$

where f is the playback FPS and N is the number of frames. When a new frame is selected, its packed representation is uploaded to the renderer and the Gaussian set is updated before drawing.

Rendering follows the standard 3D Gaussian splatting pipeline. A Gaussian in world coordinates is projected into screen space using the current camera pose and intrinsics. For a 3D point $\boldsymbol{X}$, projection is given by

$$\boldsymbol{X}_c = \boldsymbol{R}\boldsymbol{X} + \boldsymbol{t}, \qquad \tilde{\boldsymbol{x}} = \boldsymbol{K}\boldsymbol{X}_c,$$

with image coordinates obtained by perspective division. The projected 2D covariance is approximated as

$$\boldsymbol{\Sigma}_{2D} = \boldsymbol{J}\boldsymbol{W}\boldsymbol{\Sigma}_{3D}\boldsymbol{W}^\top\boldsymbol{J}^\top,$$

where $\boldsymbol{W}$ is the local world-to-camera transform and $\boldsymbol{J}$ is the Jacobian of the projection function. The resulting ellipse defines the screen-space footprint of each splat.

To support correct transparency, Gaussians are depth-sorted with respect to the current camera and composited back-to-front:

$$\boldsymbol{C} = \sum_{i=1}^{M} T_i \alpha_i \boldsymbol{c}_i, \qquad T_i = \prod_{j<i}(1-\alpha_j),$$

where $\boldsymbol{c}_i$ is the color contribution of the i-th splat. The viewer additionally supports multiple shading modes, including depth visualization and spherical harmonics shading, making it suitable for interactive inspection of dynamic PackUV-based 4DGS sequences.

Excluding I/O, our viewer achieves >200 FPS; including I/O, it maintains ≥ 30 FPS on a NVIDIA 4090 GPU with a set number of GS in the range of 300K-500K for every scene. The viewer's rendering performance independent of sequence length due to buffering.

7. CAPTURE System and Camera Synchronization

For capturing the sequences in PackUV-2B, we constructed a dedicated capture studio equipped with 88 synchronized static cameras, as shown in Figure 3. For the non-studio captures, we also built a wireless version of the similar capture setup. The ultra-large scale of the PackUV-2B dataset raises great challenges to the corresponding data processing steps. To this end, we develop an automatic pipeline to handle camera calibration, color and lighting correction, and automatic synchronization. Notably, the automatic synchronization is efficiently achieved by a carefully designed data structure in Section 7. These cameras are uniformly distributed across the four walls of a rectangular room, enabling the capture of fine-grained details with minimal occlusion. PackUV-2B provides high-resolution frames (1920×1200) at frame rates of 30, 60, or 90 FPS, selected based on the level of dynamics in each sequence. We plan to make PackUV-2B publicly available, with the goal of establishing it as a new benchmark for evaluating general-purpose, long-horizon dynamic reconstruction.

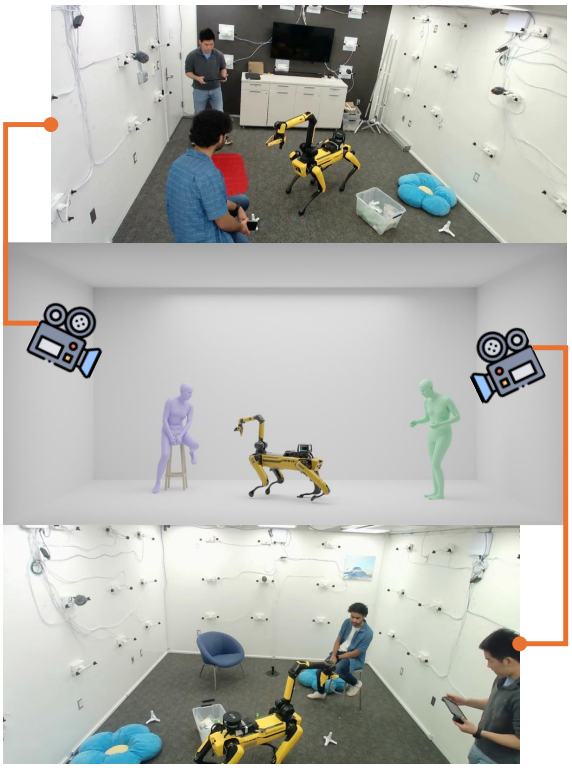

Figure 3. **CAPTURE Studio Layout.**

CAPTURE is a designated multi-view capture system with 88 cameras, suitable for various kinds of capture settings. This section provides an overview of an AVL tree implementation designed specifically for managing data captured by this system. Based on the timecode of each frame data, this implementation achieves automatic synchronization and efficient search. In practice, it is non-trivial and time-consuming to achieve high-accuracy synchronization efficiently when working with such a large number of cameras. We introduces an AVL tree implementation designed specifically for automatically synchronizing, managing and querying frame data based on timecodes. AVL trees are self-balancing binary search trees, ensuring that operations like insertion, deletion, and search can be performed efficiently, typically in $O(logn)$ time, where n is the number of nodes in the tree.

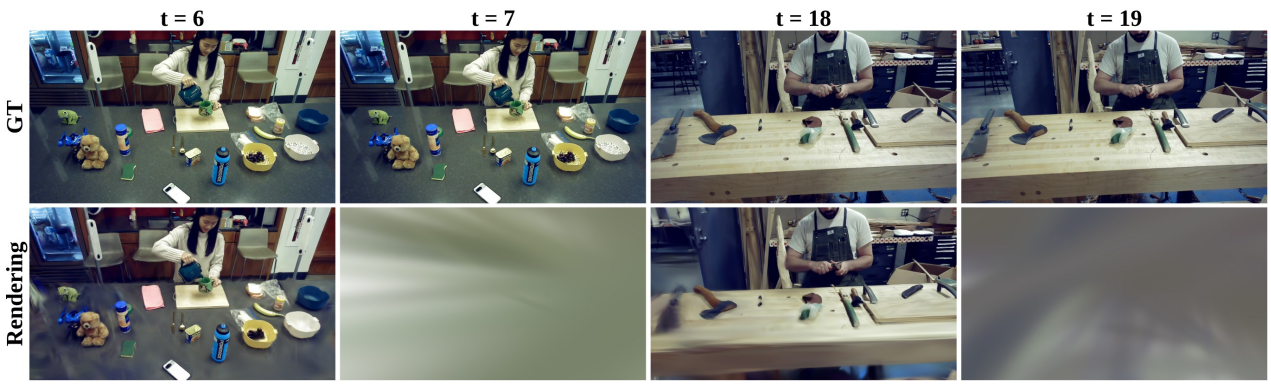

Figure 4. Shows gradient explosion in ATGS training.

The primary goal of this AVL tree is to transform unstructured raw frame data into an efficient data structure with synchronized frames. This structure allows for:

1. Quick Lookups: Rapidly searching all the frames at some timecode within a specified tolerance (threshold).
2. Data Persistence: Once an AVL tree is built, it can be saved as a binary file to avoid the need to rebuild it from the raw file in the future.

7.1. Implementation

The overview of the implementation can be divided into two steps:

1. Build an AVL tree for each camera;
2. After randomly selecting a reference camera, iterate through all the frames in the reference camera, and search the closest frames from all the other AVL trees as the synchronized frames.

7.2. Build an AVL tree

Following 2, We build an AVL tree for each camera based on the camera information file which stores the correspondence of the frame index idx_i and the timecode t_i. It takes as input each pair of $\{idx_i, t_i\}$, and then inserts it as a node into the AVL tree. The AVL tree's ensures an insertion logic that the tree remains balanced after each new node is added.

With a built AVL tree, we can achieve efficient and fast lookups. Since all the data is stored in an AVL tree, which is a type of Binary Search Tree (BST), nodes in a BST are organized such that all nodes in the left subtree of a node have timecodes less than the node's timecode, and all nodes in the right subtree have timecodes greater. This structure allows for efficient searching. Given a timecode to search, the code iterates through the tree, keeping track of the node it has encountered so far whose timecode is closest to the target timecode. It also considers a threshold to ensure that the "closest" frame found is "close enough". If the difference of timecodes between the searched frame and reference frame is larger than a pre-defined threshold, that searched frame will be dropped and viewed as "no synchronized frame".

Algorithm 2 Build an AVL Tree from a Camera Information File

function BUILDAVLTREE(filename)
  root node $r \leftarrow$ NULL
  Open a camera information file as $\mathcal{F}$
  for all f in $\mathcal{F}$ **do**
    Parse f to get a {timecode string t_i, frame index idx_i
    $r \leftarrow$ INSERTINTOAVL(r, t_i, idx_i) ▷ InsertIntoAVL handles node creation and tree balancing
  end for
  Close f
  return r
end function

7.3. Iteratively Synchronize All the Cameras

After building AVL trees for all the cameras, we prepare an automatic workflow (Algorithm 4) to synchronize all of them. Firstly, we sample a reference camera, either randomly or intentionally. Then, we go through all the frames in the reference camera and look up the closest frame of every AVL tree as the synchronized frame. Finally, we save the synchronization information as a json file for future use. Moreover, based on torchcodec, we achieve efficiently extracting any specific frames directly from raw MP4 video file without the need to extracting all the frames first.

8. Limitations and Future Work

While our method achieves high-quality 4D neural video renderings, some limitations remain that requires further exploration. A primary challenge arises from the inherently unstructured nature of 3D Gaussian representations. Although our approach introduces structure through UV projection, it still requires enforcing a large spatial arrangement to capture the fine details of real-world scenes.

Algorithm 3 Find Closest Frame by Timecode

function FINDCLOSEST(root_node r, target_timecode t, threshold τ)
  closest_node $\hat{r} \leftarrow$ NULL
  min_difference $d_{min} \leftarrow \infty$
  current_node $r' \leftarrow$ root_node r'
  while r' IS NOT NULL **do**
    difference $d \leftarrow |\text{current\_node.timecode} - \text{target\_timecode}|$
    if $d < d_{min}$ **then**
      $d_{min} \leftarrow d$
      $\hat{r} \leftarrow r'$
    end if
    if $d = 0$ **then**
      break ▷ Exact match found, cannot be closer
    end if
    if $t <$ the timecode of r' **then**
      $r' \leftarrow$ the left child node of r'
    else
      $r' \leftarrow$ the right child node of r'
    end if
  end while
  if $\hat{r}$ IS NOT NULL **and** $d_{min} > \tau$ **then**
    return NULL ▷ No node found within threshold
  else
    return $\hat{r}$
  end if
end function

Algorithm 4 Synchronization Workflow

procedure SYNCHRONIZEANDEXTRACTFRAMES
  Define `SyncMetaFile` path.
  if file at `SyncMetaFile` exists **then**
    `SyncInfo` $\leftarrow$ LOADSYNCINFO(`SyncMetaFile`) ▷ Loaded existing synchronization info.
  else
    `AVLTrees` $\leftarrow$ GENERATEAVLTREES ▷ Leverage Algorithm 2
    `SyncInfo` $\leftarrow$ SEARCHFRAMESUSINGAVLTREES ▷ Leverage Algorithm 3
    SAVESYNCINFO(`SyncInfo`, `SyncMetaFile`)
  end if
  EXTRACTANDSAVEIMAGEFRAMESFROMVIDEOS(`SyncInfo`) ▷ Extract specific frames directly from videos based on torchcodec.
end procedure

PackUV can sometimes produce dragged Gaussian artifacts when the optical flow is estimated inaccurately, which becomes a bottleneck in the pipeline. Although setting more keyframes and reducing optical flow threshold could fix the problem, improving the flow prediction module or adopting a more robust mapping strategy could lead to better temporal consistency.

Furthermore, while PackUV enables the use of video coding infrastructure by mapping to a single frame, discovering more optimal mappings could further improve storage efficiency. Another promising direction is to make this representation compatible with AR/VR devices, thereby enabling direct 4D streaming for immersive applications.

References

[1] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, Zhangyang Wang, et al. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps. *Advances in neural information processing systems*, 37:140138–140158, 2024. 1

[2] 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 2024 Conference Papers*, New York, NY, USA, 2024. Association for Computing Machinery. 3

[3] Aashish Rai and Srinath Sridhar. Egosonics: Generating synchronized audio for silent egocentric videos. In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 4935–4946. IEEE, 2025. 2

[4] Aashish Rai, Hiresh Gupta, Ayush Pandey, Francisco Vicente Carrasco, Shingo Jason Takagi, Amaury Aubel, Daeil Kim, Aayush Prakash, and Fernando De la Torre. Towards realistic generative 3d face models. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3738–3748, 2024. 3

[5] Aashish Rai, Dilin Wang, Mihir Jain, Nikolaos Sarafianos, Kefan Chen, Srinath Sridhar, and Aayush Prakash. Uvgs: Reimagining unstructured 3d gaussian splatting using uv mapping. *arXiv preprint arXiv:2502.01846*, 2025. 1, 2

[6] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *European conference on computer vision*, pages 402–419. Springer, 2020. 3

[7] Tiange Xiang, Kai Li, Chengjiang Long, Christian Häne, Peihong Guo, Scott Delp, Ehsan Adeli, and Li Fei-Fei. Repurposing 2d diffusion models with gaussian atlas for 3d generation. *arXiv preprint arXiv:2503.15877*, 2025. 1, 2

[8] Zhen Xu, Yinghao Xu, Zhiyuan Yu, Sida Peng, Jiaming Sun, Hujun Bao, and Xiaowei Zhou. Representing long volumetric video with temporal gaussian hierarchy. *ACM Transactions on Graphics*, 43(6), 2024. 11

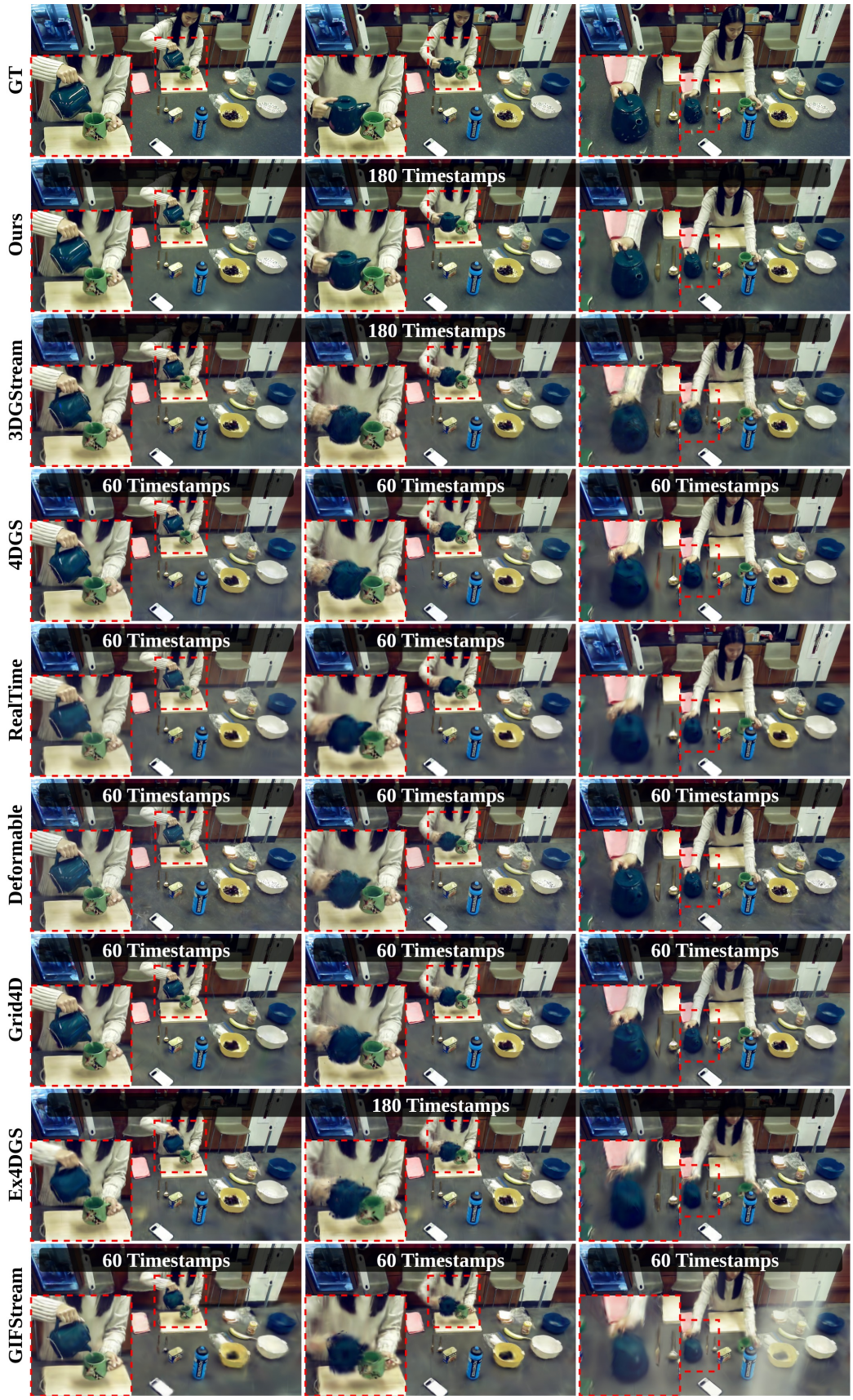

Figure 5. Baseline comparison on PackUV-2B's *Kitchen* sequence.

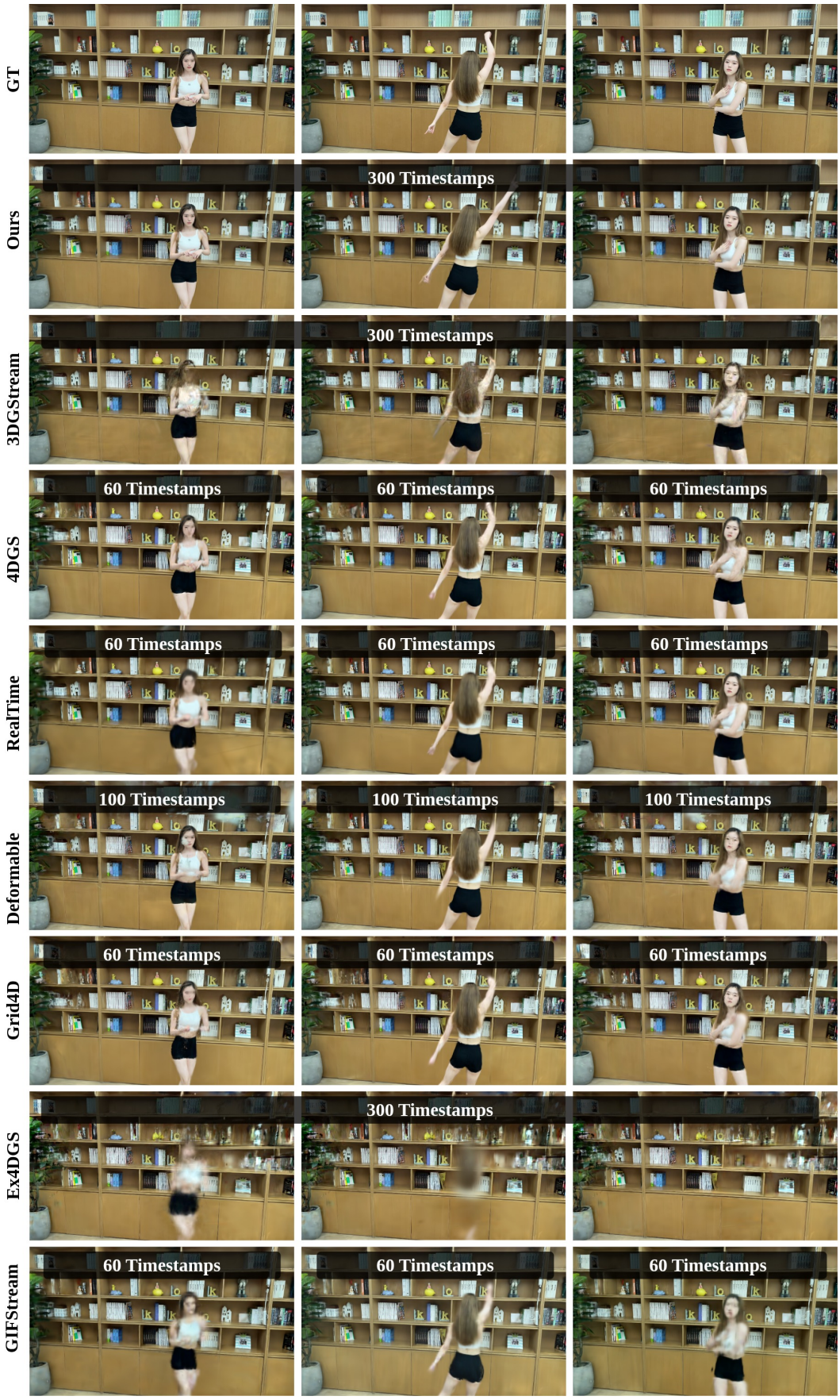

Figure 6. Shows baseline comparison on SelfCap [8] dataset.

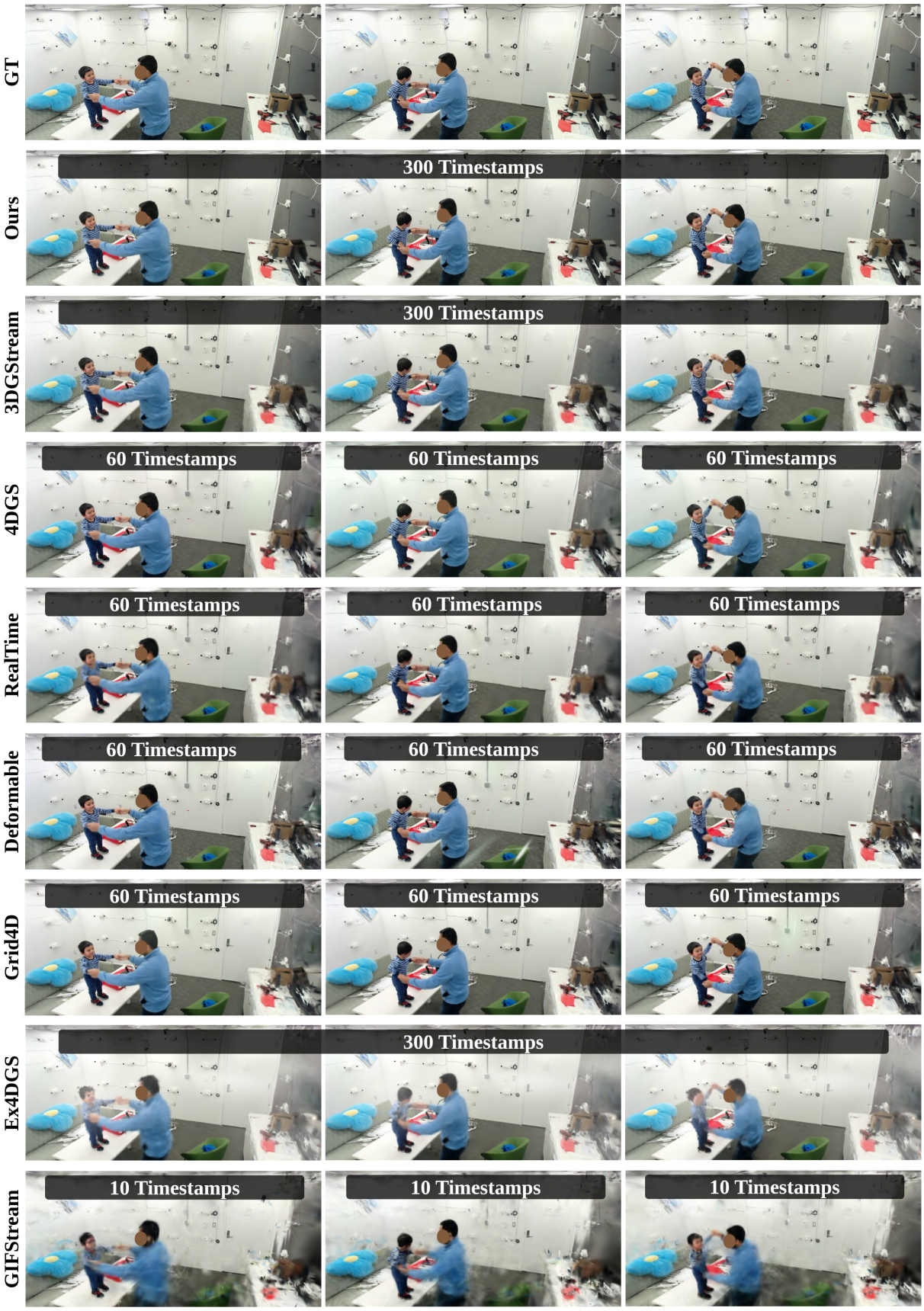

Figure 7. Shows baselines comparison on PackUV-2B's *Baby Dance* sequence.

Figure 8. Baselines comparison on PackUV-2B's *SPOT* sequence.