A. Slider Window Demo

We provide additional comparison results of baseline methods with TimeFormer. We strongly recommend opening the following website demos in folder *website_demos* and using the "slider window" to see the improvements brought by TimeFormer more clearly.

- Overview: link
- 4DGS+TimeFormer on HyperNeRF Dataset: link
- DeformGS+TimeFormer on HyperNeRF Dataset: link
- DeformGS+TimeFormer on NeRF-DS Dataset: link

B. More Details

Shared Weight, means **a same deformation field** is used in both base branch (Canonical GS $\rightarrow$ Deformation Field) and TimeFormer branch (Canonical GS $\rightarrow$ TimeFormer $\rightarrow$ Deformation Field).

Novelty. The core is that we find a light Transformer that implicitly models **cross-temporal** relationships, can outperform explicit methods just modeling **adjacent timestamps** as in previous methods. This implicit design actually addresses various hard cases in a very simple yet efficient way. We also do deep exploration to make such implicit modeling work, e.g., two-stream and shared weight, addressing issues like quality and speed degradation.

Motion visualization. Following MotionGS [76] to visualize Scene Flow $t \rightarrow t+1$, TimeFormer shows more consistent 3D flow in Fig. 12.

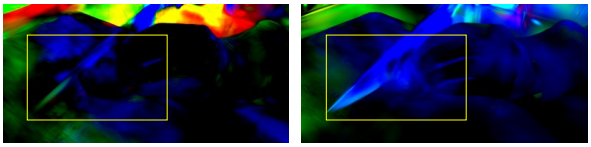

Figure 12. 3D Scene Flow. DeformGS(left) +TimeFormer (right)

Long-term motion. TimeFormer can improve both synthesis quality and motion modeling no matter for short-term or long-term motion changes, as in Fig. 12 (this case has 400+ frames). Besides, TimeFormer can be easily extended to longer videos by partitioning the long video into windows and reconstructing them separately.

LPIPS. We add LPIPS results on N3DV Dataset as in Tab. 6, showing the advantage of TimeFormer.

	4DGS	+ TimeFormer	STGS	+ TimeFormer
LPIPS↓	0.141	**0.135**	0.136	**0.131**

Table 6. LPIPS on N3DV Dataset.

Improvement from background or dynamic parts. TimeFormer is applied on all GS in a scene, so it can improve the reconstruction quality of challenging cases both in dynamic and static parts. Tab. 7 shows TimeFormer's higher quality on foreground and background.

SoTA/+TimeFormer	Broom	Lemon	Hand
Foreground	19.56/**20.89**	27.09/**30.92**	26.78/**29.91**
Background	21.48/**22.01**	28.21/**29.77**	27.89/**29.32**

Table 7. DeformGS with TimeFormer on HyperNeRF Dataset.

Densification/pruning & TimeFormer's effects. We keep the same densification/pruning strategy as in SoTA methods. TimeFormer does have a considerable influence on this process. In Fig. 8, TimeFormer accelerates the optimization and promotes faster gradient descent. Since densification/pruning is performed based on gradient, a faster-decreasing gradient means fewer GS points are needed, leading to higher FPS (Tab. 2).

Why the same deformation fields can support queries of two different Gaussian fields . TimeFormer is designed to capture temporal correlations via a Transformer network, while sharing the canonical Gaussians and subsequent deformation fields. The sharing strategy seamlessly conducts an online distillation of the knowledge (i.e., representations) from TimeFormer branch to the base branch during each optimization step, as a result of which, the two convergence curves share nearly identical trends. This pattern is also similar to the two-stream design in Modern Contrastive Learning, such as SimSiam [7] and SimCLR [6].

Why temporal size will not affect performance. Random sampling along the time dimension yields a subgraph, where each sampled timestamp is treated as a node-analogous to the subgraph sampling strategy in GraphSAGE [20]. By learning temporal dependencies within randomly sampled subgraphs, the model progressively establishes connections among all timestamps over successive training iterations, regardless of the batch size. This facilitates robust and globally consistent optimization over the entire graph structure. This also explains why random sampling is more robust than continuous sampling in Tab. 5a, because it promotes interactivity among different timestamps, leading to robust reconstruction.

C. Implementation

In this section, we provide the Pytorch [44] code of TimeFormer in Alg. 1 and two-stream optimization strategy in Alg. 2, to clarify the framework in Fig. 3.

Alg. 1 is an additional explanation for Fig. 4, including three parts: 1) Position Encoding (PE), 2) Definition of Transformer encoder, 3) Definition of tiny MLP. Note that we use a shared TimeFormer on all Gaussians in the canonical space.

The Transformer Encoder receives input structured as [seq_len, seq_batch, channel], and we input x structured as

Algorithm 1: Implementation of TimeFormer.

```
class TimeFormer (nn.module):
    # d_in:  here is 4, (x, y, z, t)
    # L: frequency of Position Encoding (PE) γ
    def __init__(self, d_in, L, nhead, d_hidden,
     n_layer):
        # PE: PE function, PE_ch:  d_in×2L
        self.PE, PE_ch = get_PE(L=L, d_in=d_in)
        # define Cross-Temporal Encoder
        layer = nn.TransformerEncoderLayer(PE_ch,
         nhead, d_hidden,
         activation=nn.functional.tanh)
        self.encoder =
         nn.TransformerEncoder(layer, n_layer)
        # define Tiny MLP
        self.mlp = nn.Linear(PE_ch, 3)

    # x:  [seq_len, seq_batch, channel]
    def forward(self, x):
        PE_x = self.PE(x)
        h = self.encoder(PE_x)
        h = self.mlp(h)
        return h
```

Algorithm 2: Two-Stream Optimization Strategy.

```
# timeformer:  TimeFormer
# deform:  Shared Deformation Field

# N: number of Gaussians
# GS: Gaussians in the canonical space
# B: size of time batch
B = 4
lambda_t = 0.8
# Vs, Ts:  sampled cameras and timestamps
# images_gt:  sampled GT images
Vs, Ts, images_gt = random.sample(Dataset, B)

# construct input to TimeFormer
# [N, 3] => [B, N, 3]
G_expanded = GS.xyz.unsqueeze(0).expand(B, -1, -1)
# [B] => [B, N, 1]
T_expanded = Ts.unsqueeze(1).expand(-1,
 N).unsqueeze(2)
# src:  [B(seq_len), N(seq_batch), 4(channel)]
src = torch.cat((G_expanded, T_expanded), dim=2)
# offset_t:  [B, N, 3]
offset_t = timeformer(src)

# use iterations to save cuda memory
loss = 0.0
for i in range(B):
    # original branch
    # simplified:  (xyz, t) => d_xyz
    d_xyz = deform(GS.xyz, Ts[i])
    image = splatting(GS, Vs[i], d_xyz=d_xyz)
    loss += L1(image, images_gt[i])

    # TimeFormer branch:  use offset_t[i,:,:]
    d_xyz_t = deform(GS.xyz+offset_t[i,:,:], Ts[i])
    image_t = splatting(GS, Vs[i], d_xyz=d_xyz_t)
    loss += lambda_t * L1(image_t, images_gt[i])

# Optimize shared deformation field, TimeFormer
 and Gaussians in the canonical space
loss.backward()
deform.optimizer.step()
timeformer.optimizer.step()
GS.optimizer.step()
```

[B, N, 4], where B is the size of time batch, N is the number of Gaussians and 4 means 3 position channel and 1 channel for the timestamp. Alg. 2 shows how we construct input to time.

In Alg. 2, we introduce the process in a time batch in detail. We first construct input to TimeFormer by concatenating Gaussian positions and sampled time stamps, as in Sec. 4.2. Besides the original branch where the deformation function is directly performed on the canonical space, we add another TimeFormer branch. TimeFormer calculates prior offsets "offset_t" via cross-time relationships before the deformation field. These two branches are optimized at the same time during training, while the TimeFormer branch can be removed during inference.

D. Ablation Studies

We provide more detailed ablation results on three scenes on NeRF-DS Dataset [68], as in Tab. 9. This further illustrates that TimeFormer is not sensitive to the number of transformer encoder layers M. However, we observe a significant decrease in reconstruction quality on all scenes without shared deformation fields.

TimeFormer does not overfit training frames due to two designs: 1) the shared deformation field between the base and TimeFormer branches promotes consistent feature learning, and 2) random timestamp sampling during optimization introduces variability for better generalization. Ablation studies further show that TimeFormer is not sensitive to hyperparameters like batch size (Tab. 5) or weight λ_t (Tab. 8).

E. Analysis on Canonical Space & FPS

Apart from Fig. 2, we provide more results in Fig. 13 and Fig. 14, as a further illustration on TimeFormer's capability to reduce Gaussians in the canonical space and improve inference speed. TimeFormer promotes more efficient spatial distribution of Gaussians in the canonical space, leading to improvements in reconstruction quality while simultaneously eliminating a substantial number of redundant Gaussians compared to baseline methods.

λ_t	0.8	0.5	1.0	Baseline
PSNR↑	26.05	26.01	25.99	25.44

Table 8. TimeFormer is not sensitive to λ_c (NeRF-DS Dataset)

Setting	Bell			Press			Sieve			Mean		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
Baseline	24.92	0.854	0.126	25.68	0.866	0.141	25.71	0.881	0.108	25.44	0.867	0.125
M=1	25.67	0.872	0.097	26.17	0.867	0.141	26.04	0.884	0.113	25.96	0.874	0.117
M=2	25.46	0.872	0.104	26.01	0.866	0.139	26.22	0.883	0.116	25.90	0.874	0.120
M=3	25.87	0.875	0.095	26.29	0.865	0.138	26.00	0.887	0.104	26.05	0.876	0.112
M=4	25.71	0.870	0.103	26.09	0.865	0.14	26.33	0.885	0.115	26.04	0.873	0.119
w/o Shared	24.12	0.832	0.137	25.01	0.854	0.146	25.29	0.869	0.113	24.81	0.852	0.132

Table 9. Ablation Results of three scenes *press*, *sieve* and *bell* on NeRF-DS Dataset [68].

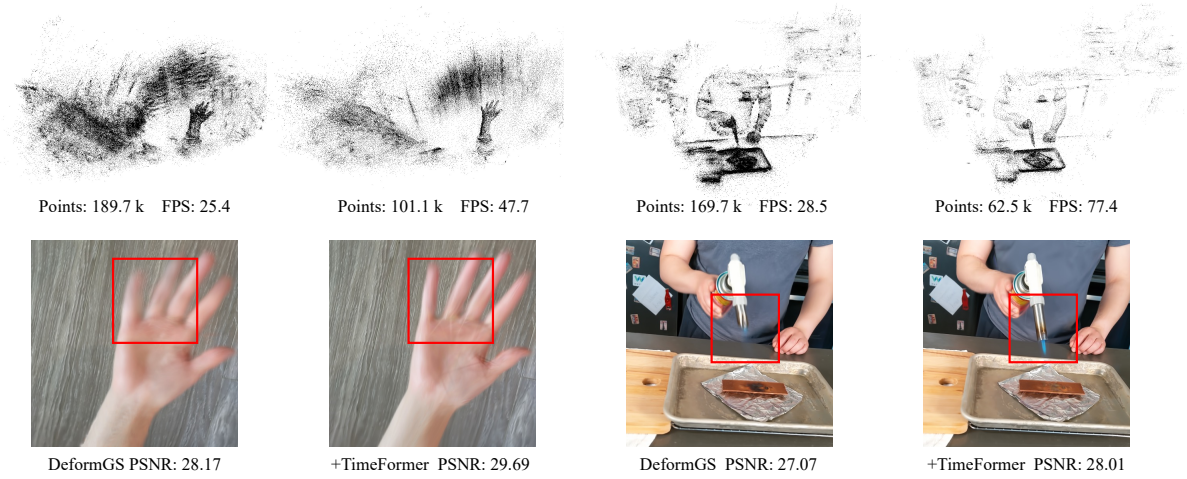

Figure 13. Comparisons of Canonical Space, FPS on Hypernerf Dataset [42].

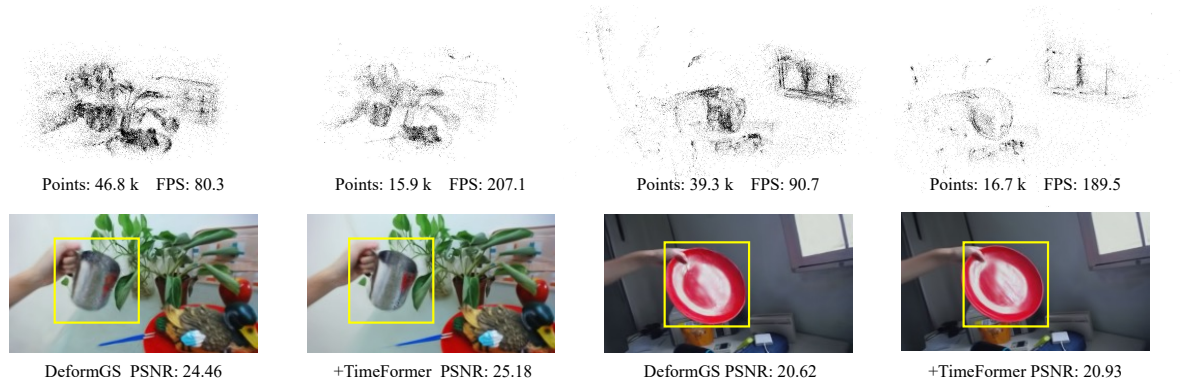

Figure 14. Comparisons of Canonical Space, FPS on NeRF-DS Dataset [68].