Appendix: Mixed Neural Voxels for Fast Multi-view Video Synthesis

Anonymous ICCV submission

Paper ID 5238

The supplementary material presents the following contents: (1) A video named MixVoxels_Video.mp4 including a simple introduction to our method and video comparisons to other representative methods. (2) This appendix which gives more details of our method and some extra experiments.

1. Factorized Voxel-grid Representation

The voxel-grid representations require large GPU memory to store the cubically growing voxel numbers. To implement the voxel-grid representations more memory efficient, we use the tensor factorization technique proposed in TensoRF [1] to reduce the memory footprint. A 3D scene tensor can be factorized into multiple compact vector and matrix (the Vector-Matrix decomposition [1]). Specifically, for a 3D tensor $\mathcal{T} \in \mathbb{R}^{N_x \times N_y \times N_z}$, the VM factorization is as follow [1]:

$$\mathcal{T} = \sum_{r=1}^{R_1} \mathbf{v}_r^1 \circ \mathbf{M}_r^{2,3} + \sum_{r=1}^{R_2} \mathbf{v}_r^2 \circ \mathbf{M}_r^{1,3} + \sum_{r=1}^{R_3} \mathbf{v}_r^3 \circ \mathbf{M}_r^{1,2} \quad (1)$$

where $\mathbf{M}_r^{2,3} \in \mathbb{R}^{N_y \times N_z}$, $\mathbf{M}_r^{1,3} \in \mathbb{R}^{N_x \times N_z}$, $\mathbf{M}_r^{1,2} \in \mathbb{R}^{N_x \times N_y}$ are matrix factors for two of the three axes. $\mathbf{v}_r^1 \in \mathbb{R}^{N_x}$, $\mathbf{v}_r^2 \in \mathbb{R}^{N_y}$, $\mathbf{v}_r^3 \in \mathbb{R}^{N_z}$ are corresponding vectors. $\circ$ is the outer-product operation. In MixVoxels, the static density voxels $\mathcal{S}^\sigma$ and the variation field $\mathcal{V}$ are 3D tensors and are factorized according to the above formulation. In this way, the space complexity is reduced from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$.

For the 4D voxel-grid feature representations $\mathcal{S}^c$, $\mathcal{G}^\sigma$ and $\mathcal{G}^c$, the last feature dimension is represented by a vector, the factorization is [1]:

$$\begin{aligned}\mathcal{T} = &\sum_{r=1}^{R_1} \mathbf{v}_r^1 \circ \mathbf{M}_r^{2,3} \circ \boldsymbol{b}_{3r-2} + \sum_{r=1}^{R_2} \mathbf{v}_r^2 \circ \mathbf{M}_r^{1,3} \circ \boldsymbol{b}_{3r-1} \\ &+ \sum_{r=1}^{R_3} \mathbf{v}_r^3 \circ \mathbf{M}_r^{1,2} \circ \boldsymbol{b}_{3r}\end{aligned} \quad (2)$$

where $\boldsymbol{b}_r$ is the vector representing the last feature dimension. In MixVoxels, the static color voxels $\mathcal{S}^c$, dynamic density and color voxels $\mathcal{G}^\sigma$ and $\mathcal{G}^c$ are 4D tensors and are factorized with Eq 2.

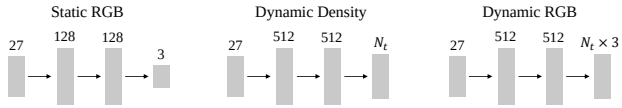

Figure 1. The architectures of all MLPs, including the networks for static color, dynamic density and dynamic color. The static density requires no MLP, because it is a scalar directly queried from the voxel-grid representation.

Figure 2. The ablation of using different hidden dimensions D.

With the help of tensor factorization, the learned model costs about 500MB for a 300-frame multi-view video scene. For the voxel resolutsions, we follow [1] to start from an initial low resolution of 256^3, and upsample the resolution at steps 1500, 2000, 2500, and 2750 with a linear increase in the log space. The final resolution is set to 640^3. The voxel-grid feature dimensions of $\mathcal{S}^c$, $\mathcal{G}^\sigma$ and $\mathcal{S}^c$ are set to 27, and the hidden dimensions of MLPs are set to 128 for static color branch and 512 for dynamic branch. For training, we use Adam [2] optimizer with a learning rate of 0.02 for voxels and $3e-3$ for MLPs. The total variation loss [5] is incorporated as a regularization to encourage the space smoothness.

2. Architectures of MLPs

Fig 1 illustrates the architectures of different MLPs, including the static RGB MLP, the dynamic density MLP and dynamic RGB MLP. All input dimensions are set to 27, which is relatively small to reduce the memory footprint. All MLPs employ a three-layer architectures, including two hidden layers and an output layer. For the static RGB MLP, the hidden dimensions are set to 128. For the dynamic branch, the hidden dimensions are set to 512 for better parsing the temporal information.

For the inner-product queries, we integrate N_T temporal queries into a matrix W. Each row of W represents a corre-

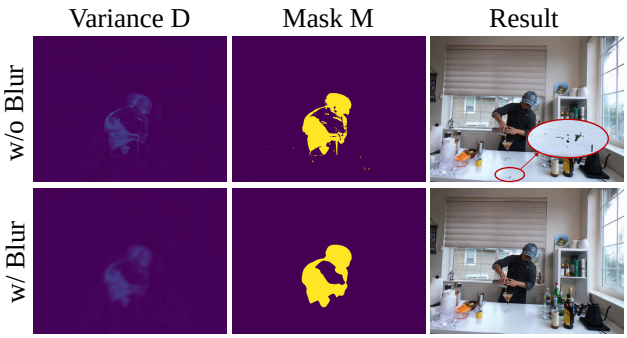

Figure 3. The ablation of using gaussian blur. In the first row, the temporal variance is directly calculated and without any other process. In the second row, the temporal variance is gaussian blurred.

sponding time query. In this way, the inner-product queries are unified to an MLP architecture. For the dynamic densities, the output is a vector of N_T values corresponding to different time steps. The dynamic RGB branch outputs a vector of $N_T \times 3$ values, representing the 3 color channels for each time step.

The choice of hidden dimensions in MLPs in the dynamic branch is also important to recover highly-dynamic components. The two dynamic MLPs for density and color have the same architecture which consists of three layers inserted by the activation layer, the dimension of hidden layer is denoted as D. We study the effect of D by setting different dimensions. Fig. 2 demonstrates the qualitative results. Models with more hidden dimensions recover the moving hand sharper. We intuitively speculate that networks with larger hidden dimensions have greater capacity to better decompress the compact features stored in each voxel.

3. Variation Field

In this section, we supplement the details of the variation field. To make the variation field work better, we apply some techniques: **(1)** Gaussian blur for reducing noisy temporal variances. **(2)** Weighted BCE loss for balancing the static and dynamic voxels. We will introduce the details of the two tricks. Besides, we also visualize the variation field in different views.

3.1. Gaussian Blur

The pixel-level temporal variance provide the supervision for training the variation field. Although the temporal variance reflect the variation of a pixel, some isolated noises are also incorporated, making some isolated dynamic voxels. The phenomenon is illustrated in Fig. 3. In normal case, identifying a static voxel to a dynamic one (false negative) will not affect the rendering quality. However, the isolated false negative will cause some artifacts, which is illustrated in the third column of Fig. 3. The phenomenon is because that the surrounding dynamic voxels of these isolated false negatives are not trained by the model, and the values of

Figure 4. Visualization of the variation field (the dynamic regions).

these isolated voxels are interpolated by the inaccurate surrounding voxels. This will make the isolated false negatives present a sudden change in colors and densities. To address the problem, we simply apply the Gaussian Blur to the temporal variance map. In this way, the isolated dynamic pixels will be smoothed by the surrounding pixels. The learned variation fields are also smoothed. The second row shows the results of using gaussian blur, where the isolated noises are removed. In detail, the gaussian kernel size is set to 31.

3.2. Weighted BCE

The static and dynamic samples are not balanced in a typical dynamic scene. To better classify the static and dynamic points, we use the weighted BCE loss to multiply a large balancing factor p for positive samples, as following:

$$\begin{aligned}\mathcal{L}_v(\boldsymbol{r}) = \frac{1}{|\mathcal{R}|}\sum_{\boldsymbol{r}} & -p \cdot \mathrm{M}(\boldsymbol{r}) \cdot \log(\hat{\mathrm{M}}(\boldsymbol{r})) \\ & -(1-\mathrm{M}(\boldsymbol{r}))\log(1-\hat{\mathrm{M}}(\boldsymbol{r}))\end{aligned} \tag{3}$$

We empirically set $p = 19$ to trade off the recall and precision and found it works well for all datasets.

We visualize the learned variation field in different views, shown in Fig. 4. We can observe that the dynamic regions identified by the variation field tightly surround the moving person. This makes MixVoxels can only process a small number of dynamic querying.

4. Voxel Prunning

To accelerate the training, the RGB prediction caculations for the points with near to zero densities can be pruned [4, 1]. For the dynamic branch, we prune the voxels with two conditions: (1): the mean density is smaller than a predefined threshold δ_1. (2): the difference between the maximum and minimum of densities over all time steps should be smaller than a threshold δ_2. In practice, $\delta_1 = 0.00001$ and $\delta_2 = 0.1$. With the voxels satisfying the above conditions, the calculation of RGB is ignored to reduce the amount of computations.

5. More Results

We provide the comprehensive quantitative results for all scenes in Tab. 1. For qualitative results for each scene, we provide them in the supplemental video.

6. Dataset Collection

We provide two multi-view synchronized videos shown in Fig. 5. The two dynamic scenes are solving-rubik where

Figure 5. The above figure showcases our collected dynamic scenes: moving-cars (top) and solving-rubik (bottom).

Table 1. Quantitative results on six different dynamic scenes [3]. We report the average performances in the last row.

Dataset	PSNR↑	DSSIM↓	LPIPS↓	FLIP↓	JOD↑
MixVoxels-S					
Spinach	31.8437	0.0193	0.1254	0.1228	8.19
Coffee	28.5421	0.0335	0.1610	0.1238	7.79
Salmon	29.2122	0.0324	0.1620	0.1144	7.96
Beef	32.8220	0.0172	0.1088	0.1107	8.24
Sear	31.5998	0.0147	0.1047	0.1273	8.39
Flame	32.1841	0.0153	0.1088	0.1239	8.35
Mean	31.0339	0.0221	0.1285	0.1204	8.15
MixVoxels-M					
Spinach	32.3087	0.0162	0.0991	0.1220	8.32
Coffee	29.4727	0.0262	0.1167	0.1223	7.99
Salmon	29.4414	0.0273	0.1280	0.1172	8.07
Beef	32.4231	0.0157	0.0920	0.1169	8.26
Sear	32.0474	0.0128	0.0848	0.1322	8.45
Flame	31.6447	0.0140	0.0896	0.1305	8.33
Mean	31.2230	0.0187	0.1017	0.1235	8.24
MixVoxels-L					
Spinach	32.2538	0.0162	0.0988	0.1220	8.28
Coffee	29.6312	0.0244	0.1057	0.1253	8.05
Salmon	29.8134	0.0255	0.1161	0.1141	8.04
Beef	32.3960	0.0157	0.0882	0.1312	8.20
Sear	32.0950	0.0122	0.0800	0.1361	8.47
Flame	31.8344	0.0144	0.0875	0.1294	8.46
Mean	31.3372	0.0180	0.0961	0.1263	8.25
MixVoxels-X					
Spinach	32.3091	0.0160	0.0622	0.1492	8.29
Coffee	30.3896	0.0232	0.0811	0.1069	8.20
Salmon	30.6021	0.0233	0.0777	0.1004	8.16
Beef	32.6293	0.0146	0.0572	0.1199	8.36
Sear	32.3310	0.0121	0.0526	0.1373	8.49
Flame	32.1037	0.0137	0.0508	0.1389	8.42
Mean	31.7274	0.0172	0.0636	0.1254	8.32

a man is solving a rubik cube with a relative fast speed, and moving-cars where some cars are moving in an outdoor scene. We collect the two scenes to test the ability of our method to deal with **(1)** scene with high-dynamic motions with fast moving speed. **(2)** scene with large areas of moving regions. To collect the synchronized video, we build a capture system similar to DyNeRF [3], which consists of 24 GoPro 9 cameras. We record videos using the linear camera mode at a resolution of 2704×1520 and frame rate of 30 FPS. We synchronized the frames by the timecode system. The intrinsic and extrinsic parameters are estimated by COLMAP.

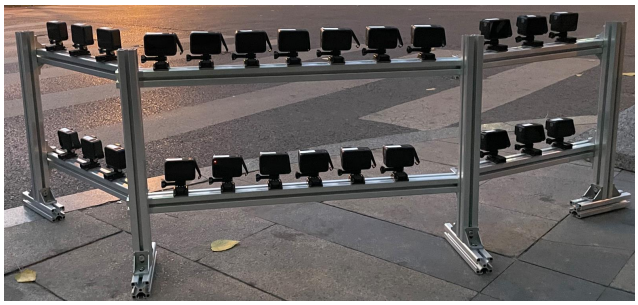

Figure 6. The multi-view synchronized GoPro 9 cameras.

References

[1] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. *arXiv preprint arXiv:2203.09517*, 2022. 1, 2

[2] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 1

[3] Tianye Li, Mira Slavcheva, Michael Zollhoefer, Simon Green, Christoph Lassner, Changil Kim, Tanner Schmidt, Steven Lovegrove, Michael Goesele, Richard Newcombe, et al. Neural 3d video synthesis from multi-view video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5521–5531, 2022. 3

[4] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *Advances in Neural Information Processing Systems*, 33:15651–15663, 2020. 2

[5] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. *arXiv preprint arXiv:2112.05131*, 2021. 1