Supplementary Mateiral
DynIBaR: Neural Dynamic Image-Based Rendering

Zhengqi Li[1], Qianqian Wang[1,2], Forrester Cole[1], Richard Tucker[1], Noah Snavely[1]

[1]Google Research [2]Cornell Tech

1. Disocclusion weights

Recall in Equation 2 of our main manuscript, we define the following weighted RGB reconstruction loss:

$$\mathcal{L}_{\text{pho}} = \sum_{\mathbf{r}} \sum_{j\in\mathcal{N}(i)} \hat{\mathbf{W}}_{j\to i}(\mathbf{r})\rho(\mathbf{C}_i(\mathbf{r}), \hat{\mathbf{C}}_{j\to i}(\mathbf{r})). \qquad (1)$$

We define the rendered disocculusion weights as:

$$\hat{\mathbf{W}}_{j\to i}(\mathbf{r}) = 1 - \int_{t_n}^{t_f} w_{i\to j}\mathbf{r}(t)\, dt$$
$$w_{i\to j}(\mathbf{r}(t)) = T_i(\mathbf{r}(t))\alpha(\sigma_i(\mathbf{r}(t))) - T_j(\mathbf{r}(t))\alpha(\sigma_j(\mathbf{r}(t))) \qquad (2)$$

where T_i is the accumulated transmittance along the ray at time i, $\alpha(\sigma) = 1 - \exp(\sigma)$ is a function converting volumetric density to alpha value, and we call the expression $T_i(\mathbf{r}(t))\alpha(\sigma_i(\mathbf{r}(t)))$ the *accumulated weight* at each sample location $\mathbf{r}(t)$. When computing the loss during optimization, we will assign a small weight to pixels where the accumulated weight borrowed from time j is different from the accumulated weight at i, addressing the motion disocclusion ambiguity described in the NSFF paper [8].

2. Static-dynamic scene factorization

Visualization. In Figs. 1 and 2, we show additional results from our motion segmentation module on all eight scenes from the Nvidia Dynamic Scene Dataset [19]. Our motion segmentation module can effectively segment moving elements from stationary elements.

Superivsion with segmentation masks. Recall that in Equation 7 of our main manuscript, we use a masked reconstruction loss to supervise our model, but in practice we observe that motion segmentation masks obtained by this module do not perfectly align with the pixel boundaries of moving objects. Therefore, we perform morphological erosion and dilation to each mover mask M_i and each static mask $(1 - M_i)$ respectively to turn off the loss near mask boundaries. For in-the-wild videos, we combine estimated masks with semantic segmentation [6] through mask union since we found that our Bayesian learning strategy can sometimes miss objects with subtle or colinear motions.

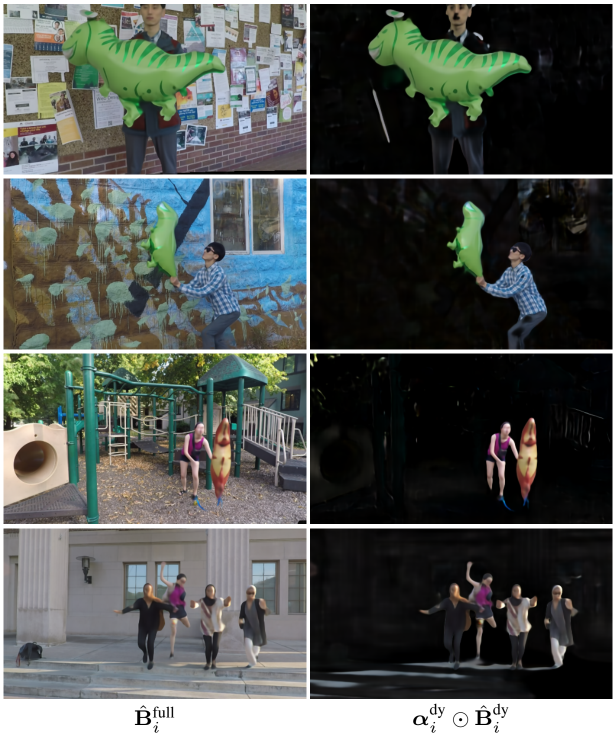

$\hat{\mathbf{B}}_i^{\text{full}}$
$\boldsymbol{\alpha}_i^{\text{dy}} \odot \hat{\mathbf{B}}_i^{\text{dy}}$

Figure 1. **Additional motion segmentation results.** We show full renderings $\hat{\mathbf{B}}_i^{\text{full}}$ (left) as well as motion segmentations overlaid with rendered dynamic content $\boldsymbol{\alpha}_i^{\text{dy}} \odot \hat{\mathbf{B}}_i^{\text{dy}}$ (right).

3. Regularization

Data-driven priors $\mathcal{L}_{\text{data}}$. Following NSFF [8], our data driven priors consist of monocular depth and geometric consistency terms $\mathcal{L}_{\text{data}} = \lambda_z\mathcal{L}_z + \lambda_{\text{geo}}\mathcal{L}_{\text{geo}}$.

The mono-depth term $\mathcal{L}_z$ minimizes the ℓ_1 difference between expected disparity rendered from our representation and the disparity estimated from Zhang *et al*. [20]. We observe that the estimated depths of moving objects from Zhang *et al*. tend to degenerate when the method is applied to in-the-wild videos where camera and object motion are largely colinear. Therefore, we modify the framework of Zhang *et al*. to improve its robustness to in-the-wild videos.

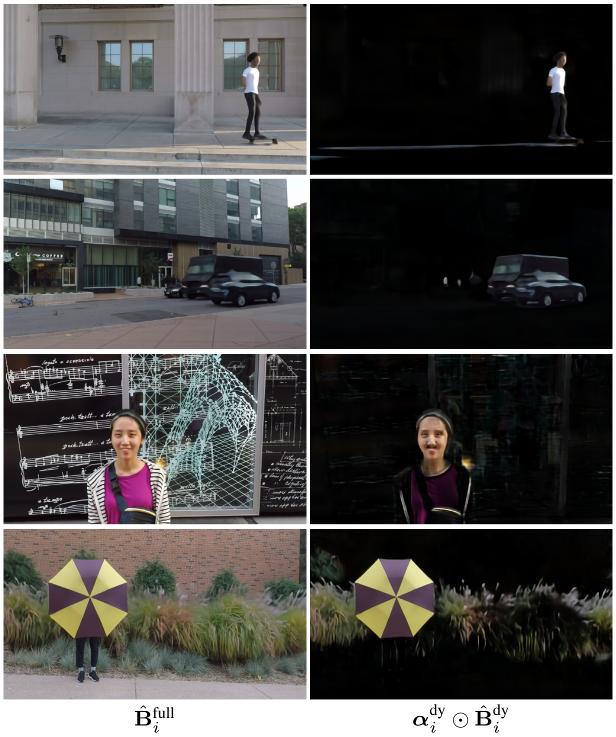

$\hat{\mathbf{B}}_i^{\text{full}}$
$\boldsymbol{\alpha}_i^{\text{dy}} \odot \hat{\mathbf{B}}_i^{\text{dy}}$

Figure 2. **Additional motion segmentation results.** We show full renderings $\hat{\mathbf{B}}_i^{\text{full}}$ (left) as well as motion segmentations overlaid with rendered dynamic content $\boldsymbol{\alpha}_i^{\text{dy}} \odot \hat{\mathbf{B}}_i^{\text{dy}}$ (right).

In particular, we make two major modifications in our implementation:

- We estimate both scale and shift in monocular disparity space by aligning monocular depths with the structure from motion (SfM) point cloud. We first compute the median scale and shift for each input frame, and then average these over the entire video to obtain a final estimated scale and shift. We clip negatives disparity to 0.01, since negative depth will cause the optimization to diverge.
- We add an extra depth regularization loss with weight 10 that encourages the current estimated depth to be close to the initial depth estimated from DPT [13], where we apply a scale-invariant depth loss [7].

The geometric consistency term $\mathcal{L}_{\text{geo}}$ minimizes the ℓ_1 reprojection error between the observed 2D optical flow and the expected 2D flow rendered from the motion trajectory fields within time range $[i-r, i+r]$. We refer readers to NSFF [8] for full derivations. Since estimated depth and flow can be inaccurate, we decay the weights of the two corresponding terms by a factor of 10 every $40K$ optimization iterations.

Motion trajectory priors $\mathcal{L}_{\text{MT}}$. Our motion trajectory priors consist of cycle consistency and spatio-temporal smoothness terms $\mathcal{L}_{\text{MT}} = \lambda_{\text{cycle}}\mathcal{L}_{\text{cycle}} + \lambda_{\text{sm}}\mathcal{L}_{\text{sm}}$.

The cycle consistency term $\mathcal{L}_{\text{cycle}}$ encourages that for a given sampled 3D point $\mathbf{x}$ at time i and its correspondence at time j, $\mathbf{x}_{i\to j}$, their associated motion trajectory should be consistent:

$$\mathcal{L}_{\text{cycle}} = \sum_{\mathbf{x}} \sum_{j \in \mathcal{N}(i)} w_{i\to j}(\mathbf{x}) ||\Delta_{\mathbf{x},i}(j) + \Delta_{\mathbf{x}_{i\to j},j}(i)||_1$$

$$w_{i\to j}(\mathbf{x}) = 1 - |T_i(\mathbf{x})\alpha(\sigma_i(\mathbf{x})) - T_j(\mathbf{x})\alpha(\sigma_j(\mathbf{x}))| \quad (3)$$

where $w_{i\to j}$ is a disocculusion weight meant to address motion disocculusion ambiguity per sample point.

The spatio-temporal smoothness term $\mathcal{L}_{\text{sm}}$ encourages the estimated motion trajectory to be smooth in both space and time, by enforcing 1) an ℓ_1 loss in relative displacement between spatially neighboring points along a ray, 2) an ℓ_1 loss in relative displacement between temporally neighboring points along a motion trajectory, and 3) estimated scene flows is potentially small:

$$\mathcal{L}_{\text{sm}} = \sum_{j \in \mathcal{N}(i)} \sum_{t \in [t_n, t_f]} ||\Delta_{\mathbf{r}(t),i}(j) - \Delta_{\mathbf{r}(t+1),i}(j)||_1 \quad (4)$$

$$+ ||\Delta_{\mathbf{r}(t),j}(j+1) - \Delta_{\mathbf{r}(t),j+1}(j+2)||_1 \quad (5)$$

$$+ \sum_{j \in \mathcal{N}(i)} \sum_{t \in [t_n, t_f]} ||\Delta_{\mathbf{r}(t),i}(j)||_1 \quad (6)$$

Compactness priors $\mathcal{L}_{\text{cpt}}$. Our compactness prior $\mathcal{L}_{\text{cpt}}$ consists of entropy and distortion terms $\mathcal{L}_{\text{cpt}} = \lambda_{\text{etp}}\mathcal{L}_{\text{etp}} + \lambda_{\text{dist}}\mathcal{L}_{\text{dist}}$.

The entropy loss encourages the static-dynamic scene factorization to be close to binary by minimizing the entropy of ratio of accumulated weights rendered from the time-varying dynamic model:

$$\mathcal{L}_{\text{etp}} = \sum_{\mathbf{r}} -R(\mathbf{r})\log(R(\mathbf{r})) - (1-R(\mathbf{r}))\log(1-R(\mathbf{r}))$$

$$R(\mathbf{r}) = \frac{\hat{W}^{\text{dy}}(\mathbf{r})}{\hat{W}^{\text{dy}}(\mathbf{r}) + \hat{W}^{\text{st}}(\mathbf{r})} \quad (7)$$

where $\hat{W}^{\text{dy}}$ and $\hat{W}^{\text{st}}$ are the accumulated weights rendered from the time-varying and time-invariant models respectively. Note that we don't adopt the skewness parameter from D^2-NeRF [15] since we observe that it does not improve decomposition quality in our framework. In addition, we add the distortion loss $\mathcal{L}_{\text{dist}}$ proposed in Mip-NeRF 360 [1], which encourages estimated geometry to be concentrated by consolidating accumulated weights along the ray into a small region. We refer readers to Mip-NeRF 360 [1] for full derivations of this loss. We observed that these two terms improve visual decomposition and rendering quality, but do not yield noticeable improvements in our numerical evaluation.

4. Network architectures

Encoder-decoder network. In Table 1, we show the architecture of the encoder-decoder network used in our motion

Input (**id**: dimension)	Layer	Output (**id**: dimension)
0: $W \times H \times 4$	7×7 Conv, 64, stride 2	**1**: $\frac{W}{2} \times \frac{H}{2} \times 64$
1: $\frac{W}{2} \times \frac{H}{2} \times 64$	3×3 MaxPool, stride 2	**2**: $\frac{W}{2} \times \frac{H}{2} \times 64$
2: $\frac{W}{2} \times \frac{H}{2} \times 64$	Residual Block 1	**3**: $\frac{W}{2} \times \frac{H}{2} \times 64$
3: $\frac{W}{2} \times \frac{H}{2} \times 64$	Residual Block 2	**4**: $\frac{H}{8} \times \frac{H}{8} \times 128$
4: $\frac{H}{8} \times \frac{H}{8} \times 128$	Residual Block 3	**5**: $\frac{W}{16} \times \frac{H}{16} \times 256$
5: $\frac{W}{16} \times \frac{H}{16} \times 256$	3×3 Upconv, 128, factor 2	**6**: $\frac{H}{8} \times \frac{H}{8} \times 128$
6: $\frac{H}{8} \times \frac{H}{8} \times 128$	3×3 Conv, 128	**7**: $\frac{H}{8} \times \frac{H}{8} \times 128$
7: $\frac{H}{8} \times \frac{H}{8} \times 128$	3×3 Upconv, 128, factor 2	**8**: $\frac{W}{4} \times \frac{H}{4} \times 128$
8: $\frac{W}{4} \times \frac{H}{4} \times 128$	3×3 Conv, 128	**9**: $\frac{W}{4} \times \frac{H}{4} \times 64$
9: $\frac{H}{4} \times \frac{H}{4} \times 128$	3×3 Upconv, 128, factor 2	**10**: $\frac{W}{2} \times \frac{H}{2} \times 128$
10: $\frac{W}{2} \times \frac{H}{2} \times 128$	3×3 Conv, 64	**11**: $\frac{W}{2} \times \frac{H}{2} \times 64$
11: $\frac{H}{2} \times \frac{H}{2} \times 64$	3×3 Upconv, 64, factor 2	**12**: $W \times H \times 64$
12: $W \times H \times 64$	1×1 Conv, 5	**Out**: $W \times H \times 5$

Table 1. **Encoder-decoder network for motion segmentation**. ‘Conv” is a sequence of operations, including 2D convolution and ReLU followed by Instance Normalization [18]. “Upconv” is a 2x bilinear upsampling operator followed by a “Conv” operation.

Input (**id**: dimension)	Layer	Output (**id**: dimension)
0: $W \times H \times 3$	7×7 Conv, 64, stride 2	**1**: $\frac{W}{2} \times \frac{H}{2} \times 64$
1: $\frac{W}{2} \times \frac{H}{2} \times 64$	3×3 MaxPool, stride 2	**2**: $\frac{W}{4} \times \frac{H}{4} \times 64$
2: $\frac{W}{4} \times \frac{H}{4} \times 64$	Residual Block 1	**3**: $\frac{W}{4} \times \frac{H}{4} \times 64$
3: $\frac{W}{4} \times \frac{H}{4} \times 64$	1×1 Conv, 64	**Out**: $\frac{W}{4} \times \frac{H}{4} \times 64$

Table 2. **2D CNN feature extractor**. ‘Conv” represents a sequence of operations, including 2D convolution and ReLU followed by Instance Normalization [18]. Residual Block 1 represents the first residual block (as used in ResNet34 [4]).

segmentation module. Note that we do not adopt the skip connections often used in U-Net, since such connections would easily lead to a trivial decomposition solution where the encoder-decoder network predicts entire scene content by copying pixel information from the input frame.

2D CNN feature extractor. The network architecture of our 2D CNN feature extractor is shown in Table 2; we use a separate feature extractor for the coarse and fine models. We predict a 64-channel feature map from an input RGB frame, where the first 32 channels are used for the time-varying model, and the last 32 channels are used for time-invariant model.

Time-varying dynamic model. We show the architecture of our time-varying dynamic model in Fig. 3. The input to the model for each sample location is the time embedding $\gamma(i)$, where we use Fourier position encoding with ten frequencies. The concatenation of image color and features extracted from nearby source views are average-pooled to obtain mean and variance vectors. The processed features from the time embedding and image features are then element-wise added and fed to a MLP to produce an intermediate feature $\mathbf{f}'_j$ and pooling weight w'_j. w'_j is used to perform weight pooling to the intermediate feature $\mathbf{f}'_j$ to obtain aggregated features $\mathbf{f}''$. The aggregated features of each sample along the ray are then fed to the ray transformer to construct features that are cross-attended. The cross-attended features from the ray transformer are combined with global spatial coordinate embedding $\gamma(\mathbf{x})$ and view direction embedding $\gamma(\mathbf{d})$ to output per-sample density σ_i and color $\mathbf{c}_i$ at time i respectively. Note that we concatenate the global spatial coordinate embedding $\gamma(\mathbf{x})$ after the ray transformer instead of before the ray transformer. The reason for this choice is that if we append $\gamma(\mathbf{x})$ before the ray transformer, the global spatial coordinate embedding would leak the information that the input ray is curved due to cross-time rendering. However, during novel view synthesis, the input target ray is always straight. The resulting inconsistency in the ray space between optimization and inference stages would hurt model performance in view interpolation.

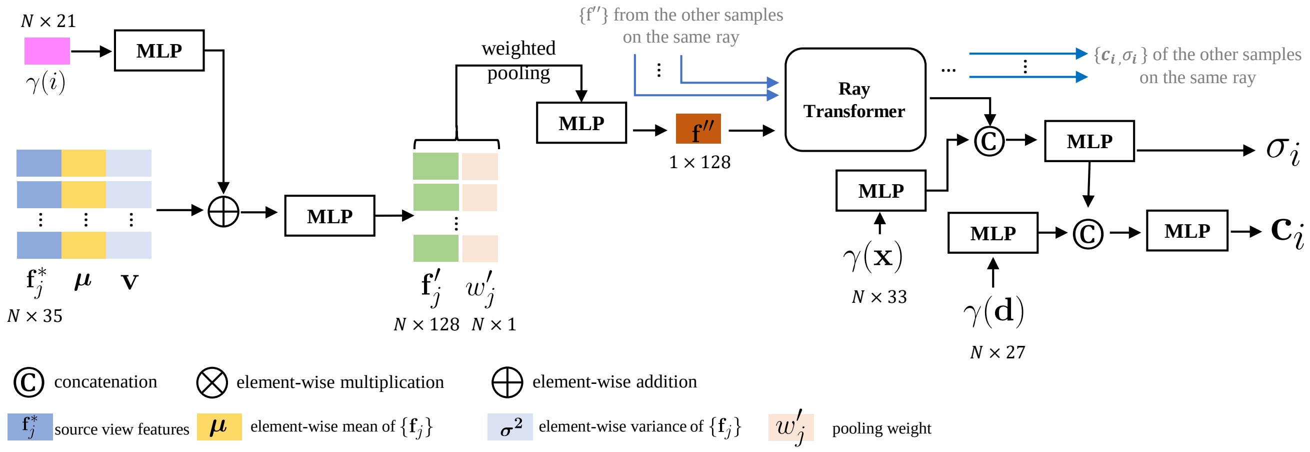
Time-invariant static model. We show the architecture of our time-invariant static model in Fig. 4. The input to the model for each sample location includes 1) the embedding of the ray coordinate with respect to the target view, $\gamma(\mathbf{r}_i)$; 2) input features from each source view, which are the concatenation of source view color C_j and CNN feature $\mathbf{f}_j$; 3) the embedding of ray coordinates with respect to the source view, $\gamma(\mathbf{r}_j)$; 4) the relative viewing direction $\Delta\mathbf{d}_j$; and 5) the global spatial coordinate embedding $\gamma(\mathbf{x})$:

$$\mathbf{f}^*_j = [C_j|\mathbf{f}_j|\gamma(\mathbf{r}_j)|\Delta\mathbf{d}_j|\gamma(\mathbf{x})] \tag{8}$$

where $|$ is a concatenation operator. We use Fourier position encoding with four frequencies for all coordinate embeddings. We then compute the dot product between the features from source views and those from target rays. The multiplied features are then fed into an MLP, yielding intermediate feature $\mathbf{f}'_j$ and pooling weights w'_j. An aggregated feature $\mathbf{f}''$ is then obtained by weight pooling $\mathbf{f}'_j$ through w'_j. Next, the aggregated feature $\mathbf{f}''$ is fed into the ray transformer to produce a per-sample density σ, while the output features of the ray transformer are concatenated with the relative viewing direction $\Delta\mathbf{d}_j$ and intermediate image feature $\mathbf{f}'_j$ from each source view, and then these are fed as input to another MLP to produce color blending weights w^c_j corresponding to each source view. The source view color C_j is then linearly blended through w^c_j to obtain the final color $\mathbf{c}$ for each sampled location.

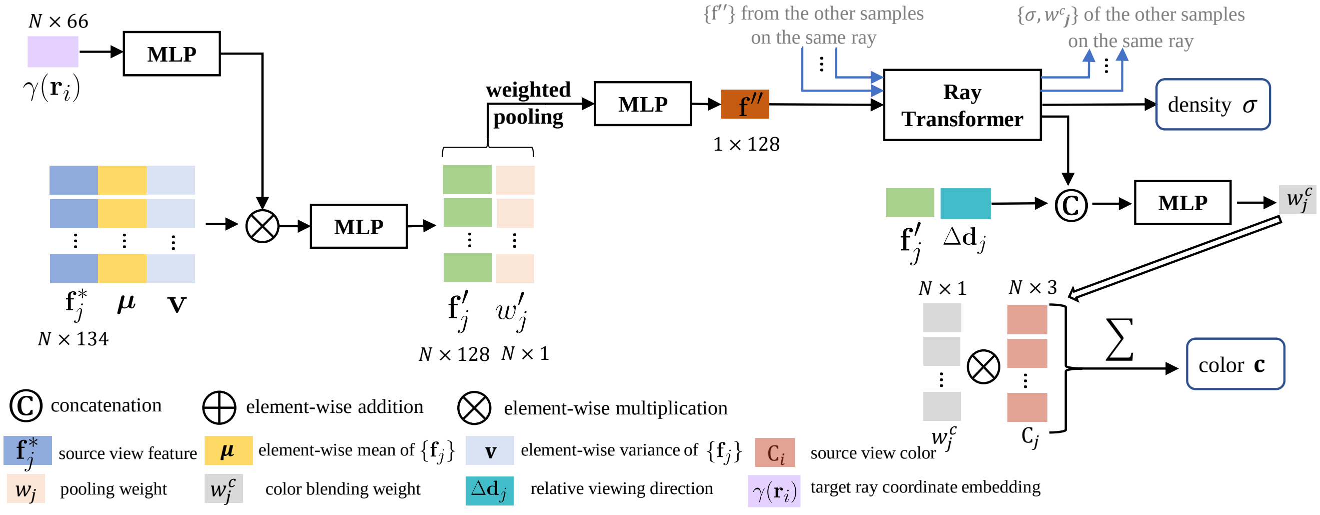
5. Additional implementation details

Global spatial coordinate embedding With local image feature aggregation alone, it is hard to determine density accurately on non-surface or occluded surface points due to inconsistent features from different source views, as described in NeuRay [9]. Therefore, to improve global reasoning for density prediction, we append a global spatial coordinate embedding as an input to the ray transformer, in addition to the time embedding, similar to the ideas from [16].



$N \times 21$
$\gamma(i)$
MLP
$\mathbf{f}_j^*$ $\boldsymbol{\mu}$ $\mathbf{v}$
$N \times 35$
MLP
$\mathbf{f}_j'$ w_j'
$N \times 128$ $N \times 1$
weighted pooling
MLP
$\mathbf{f}''$
1×128
{f''} from the other samples on the same ray
Ray Transformer
$\{c_i, \sigma_i\}$ of the other samples on the same ray
MLP
$\gamma(\mathbf{x})$
$N \times 33$
MLP
σ_i
MLP
$\gamma(\mathbf{d})$
$N \times 27$
MLP
$\mathbf{c}_i$
Ⓒ concatenation
⊗ element-wise multiplication
⊕ element-wise addition
$\mathbf{f}_j^*$ source view features
$\boldsymbol{\mu}$ element-wise mean of $\{\mathbf{f}_j\}$
σ^2 element-wise variance of $\{\mathbf{f}_j\}$
w_j' pooling weight

Figure 3. **Network architecture of our time-varying dynamic representation.**



$N \times 66$
$\gamma(\mathbf{r}_i)$
MLP
$\mathbf{f}_j^*$ $\boldsymbol{\mu}$ $\mathbf{v}$
$N \times 134$
MLP
$\mathbf{f}_j'$ w_j'
$N \times 128$ $N \times 1$
weighted pooling
MLP
$\mathbf{f}''$
1×128
{f''} from the other samples on the same ray
Ray Transformer
$\{\sigma, w^c_j\}$ of the other samples on the same ray
density σ
$\mathbf{f}_j'$ $\Delta\mathbf{d}_j$
MLP
w_j^c
$N \times 1$
$N \times 3$
w_j^c
C_j
$\sum$
color $\mathbf{c}$
Ⓒ concatenation
⊕ element-wise addition
⊗ element-wise multiplication
$\mathbf{f}_j^*$ source view feature
$\boldsymbol{\mu}$ element-wise mean of $\{\mathbf{f}_j\}$
$\mathbf{v}$ element-wise variance of $\{\mathbf{f}_j\}$
C_i source view color
w_j pooling weight
w_j^c color blending weight
$\Delta\mathbf{d}_j$ relative viewing direction
$\gamma(\mathbf{r}_i)$ target ray coordinate embedding

Figure 4. **Network architecture of our time-invariant static representation.**

For the time-invariant model, we concatenate each extracted local image feature at each sampled point with its corresponding (1) ray coordinates with respect to the target view, (2) ray coordinates with respect to the source view, and (3) xyz coordinates in the global reference frame. We use Plücker coordinates [5] to represent ray coordinates since these allow us to model arbitrary scene and camera geometry without ambiguities or singularities.

For the time-varying model, recall that local image features are aggregated along curved rays during the optimization stage, but once optimization is complete, novel views are rendered by aggregating features along straight rays cast directly from the desired target view. As a result, we observe that concatenating the 3D coordinates of $\mathbf{x}$ or $\mathbf{x}_{i\to j}$ with the corresponding 2D image features as inputs to the ray transformer impairs view interpolation. Therefore, we concatenate the encoded global 3D coordinates of $\mathbf{x}$ with the features output from the attention module in the ray transformer before predicting density and color (please see the supplement for more details of this design choice). For all spatial coordinates, we apply a Fourier features–based positional encoding [10, 17] with four frequencies. but note that our embedding is not mainly used for recovering high-frequency details as in most of prior work and thus does not require special scene parameterization.

Optimization/inference details. We reconstruct the entire scene in a standard Euclidean space where we scale each scene so that depth of the near plane is set to $\frac{4}{3}$. We set the maximum value of the far plane depth to be 75. For our coarse to fine sampling strategy, we observe that simultaneously optimizing coarse and fine models together tends to be less stable in dynamic scene reconstruction, and we

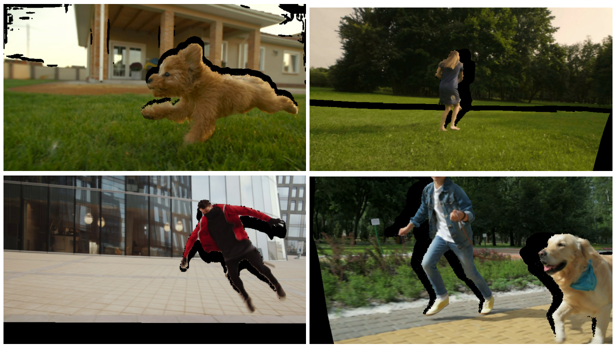


Figure 5. **Virtual views.** We show views rendered at nearby virtual viewpoints. The virtual views help model recover residual details of fast moving objects that are hard to be tracked, and avoid degenerated reconstruction in real world monocular videos.

therefore train the two models sequentially. Specifically, we first optimize the coarse model for 300K iterations. We then initialize the fine model with the weights of the pre-trained coarse model and optimize it for another 150K iterations.

In image-based motion segmentation module, we set the learning rate of the encoder-decoder network D to 4×10^{-4}. We set the learning rate of the ray transformer and the 2D CNN feature extractor to 4×10^{-4} and 8×10^{-4} respectively.

For our two main representations, we set the initial learning rate to 5×10^{-4} for both ray transformers (representing static and dynamic scene contents), and to 1×10^{-3} for the 2D CNN feature extractors. We set the learning rate to 5×10^{-4} for the motion trajectory MLP, and to 2×10^{-4} for the time-varying motion basis. We decay the learning rate by a factor of 2 every 50K iterations for the coarse model, and every 25K iterations for the fine model. At each optimization step, we randomly choose a target image and sample a batch of random rays with size 4096.

In the Nvidia Dynamic Scene and the UCSD Dynamic Scene datasets, the training images are resized to 512×288 and 533×300 respectively. We set the weights of the data-driven priors $\lambda_{\text{z}} = 5 \times 10^{-2}, \lambda_{\text{geo}} = 5 \times 10^{-3}$. We set the weights of the motion trajectory priors $\lambda_{\text{cycle}} = 0.1, \lambda_{\text{sm}} = 0.05$, where we increase the cycle consistency weight λ_{cycle} by 0.1 every 40K iterations until it reaches 0.5. We set the weights of compactness priors $\lambda_{\text{etp}} = 5 \times 10^{-4}, \lambda_{\text{dist}} = 1 \times 10^{-3}$. For in-the-wild videos, we resize video frames to 768×432, and we set $\lambda_{\text{z}} = 0.3, \lambda_{\text{geo}} = 2 \times 10^{-2}$ and keep other hyper-parameters the same.

6DoF Stabilization. For all 6DoF stabilization results produced by our approach and prior methods, we apply a Gaussian filter with kernel size 50 time steps to the first two column vectors in the camera rotation matrix and camera translation vector, and render novel views along this Gaussian-filtered camera path.

Handling degeneracy of monocular videos. Prior work [8] observed that optimization might converge to bad local minimal if camera and object motions are close to colinear, or object motions are too fast to track with flow algorithms. To handling first case, during both training and inference, we mask out input features for static time-invariant model with estimated motion masks before performing feature aggregation and attention. To handle the second case, we propose to synthesize virtual views via depths estimated from [20]. These views are rendered at randomly sampled nearby viewpoints. We randomly sample an image and use it as an additional source view for multi-view feature aggregation during rendering. Furthermore, to recover details of fast moving objects, we fine-tune the models by supervising full renderings from the models against randomly sampled virtual views masked by estimated motion segmentation. Fig. **??** show several examples of rendered virtual views. We found it significantly improves optimization stability and rendering quality for in-the-wild videos, while having no noticeable impact on the two benchmarks due to uncorrelated camera-object motion patterns [3].

6. Limitations

Our approach inherits similar limitations from prior methods [2, 8]. First, our method has a long optimization and rendering time. Integrating recent voxel representations [11, 14] into our IBR framework is an interesting research direction for accelerating training and rendering. Second, although our network architecture is based on that of IBRNet, our system is a per-scene optimization approach, and thus does not generalize across scenes and is unable to inpaint or outpaint unseen regions. Third, the performance of our method degrades if the target viewpoint is from an oblique angle with respect to the input views, because our approach only use source views within a small temporal window for reconstructing moving content. Combining our approach with canonical space–based methods using more advanced data-driven priors [12] to overcome this limitation can be an exciting research direction. Last, our representation renders novel view by using nearby source views instead of constructing a global scene representation like NeRFs. Therefore, rendered contents can be unrealistic or blank if insufficient (typical less than 5) correspondences from nearby source views are visible. Developing a better view selection algorithm within a volumetric IBR framework is an interesting direction to explore for addressing this problem.

References

[1] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded anti-aliased neural radiance fields. In *Proc. Computer Vision and Pattern Recognition (CVPR)*, 2022. 2

[2] Chen Gao, Ayush Saraf, Johannes Kopf, and Jia-Bin Huang. Dynamic view synthesis from dynamic monocular video. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5712–5721, 2021. 5

[3] Hang Gao, Ruilong Li, Shubham Tulsiani, Bryan Russell, and Angjoo Kanazawa. Monocular dynamic view synthesis: A reality check. *arXiv preprint arXiv:2210.13445*, 2022. 5

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 3

[5] Yan-Bin Jia. Plücker coordinates for lines in the space. *Problem Solving Techniques for Applied Computer Science, Com-S-477/577 Course Handout. Iowa State University. http://web.cs.iastate.edu/∼cs577/handouts/plucker-coordinates.pdf*, 2020. 4

[6] Lei Ke, Martin Danelljan, Xia Li, Yu-Wing Tai, Chi-Keung Tang, and Fisher Yu. Mask transfiner for high-quality instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4412–4421, 2022. 1

[7] Zhengqi Li, Tali Dekel, Forrester Cole, Richard Tucker, Noah Snavely, Ce Liu, and William T Freeman. Learning the depths of moving people by watching frozen people. In *Proc. Computer Vision and Pattern Recognition (CVPR)*, pages 4521–4530, 2019. 2

[8] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural scene flow fields for space-time view synthesis of dynamic scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR)*, 2021. 1, 2, 5

[9] Yuan Liu, Sida Peng, Lingjie Liu, Qianqian Wang, Peng Wang, Christian Theobalt, Xiaowei Zhou, and Wenping Wang. Neural rays for occlusion-aware image-based rendering. In *CVPR*, 2022. 3

[10] 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. *Proc. European Conf. on Computer Vision (ECCV)*, 2020. 4

[11] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *arXiv preprint arXiv:2201.05989*, 2022. 5

[12] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint arXiv:2209.14988*, 2022. 5

[13] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(3), 2022. 2

[14] Sara Fridovich-Keil and Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, 2022. 5

[15] Liangchen Song, Anpei Chen, Zhong Li, Zhang Chen, Lele Chen, Junsong Yuan, Yi Xu, and Andreas Geiger. NeRFPlayer: A streamable dynamic scene representation with decomposed neural radiance fields. *arXiv preprint arXiv:2210.15947*, 2022. 2

[16] Mohammed Suhail, Carlos Esteves, Leonid Sigal, and Ameesh Makadia. Light field neural rendering. In *Proc. Computer Vision and Pattern Recognition (CVPR)*, 2022. 3

[17] Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33:7537–7547, 2020. 4

[18] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016. 3

[19] Jae Shin Yoon, Kihwan Kim, Orazio Gallo, Hyun Soo Park, and Jan Kautz. Novel view synthesis of dynamic scenes with globally coherent depths from a monocular camera. In *Proc. Computer Vision and Pattern Recognition (CVPR)*, pages 5336–5345, 2020. 1

[20] Zhoutong Zhang, Forrester Cole, Richard Tucker, William T Freeman, and Tali Dekel. Consistent depth of moving objects in video. *ACM Transactions on Graphics (TOG)*, 40(4):1–12, 2021. 1, 5