

# FlowIBR: Leveraging Pre-Training for Efficient Neural Image-Based Rendering of Dynamic Scenes

## Supplementary Material

### A. Network architecture

The scene flow network is as a combination of the permutohedral lattice from PermutoSDF [38], which provides a latent vector for each 4D input position, and a shallow multilayer perceptron (MLP), which decodes the latent vector into the scene flow. An overview of the network architecture is presented in Fig. 6. The hash map of the permutohedral lattice has  $L = 10$  different levels, containing  $T = 2^{18}$  feature vectors with a dimensionality of  $F = 2$ .

The MLP is four layers deep, with 32 nodes, and GELU activation functions [16] in the hidden layers. For being applicable to general scenes, FlowIBR does not utilize normalized device coordinates (NDC) which are in the range  $[0, 1]$ , but rather unbounded cartesian coordinates, which is the reason we use a linear activation for the final layer. NDCs limit a method to forward facing scenes, but are common in NeRF based methods where the datasets often fulfill this criterion. The network does not use batch normalization, dropout or any other regularization not discussed in this paper.

Images are synthesized by projecting the target rays to eight temporally close source observations, which also means projecting the rays to the eight respective times.

To show the effectiveness behind using a permutohedral lattice instead of a common architecture, we conduct an ablation by using a simple NeRF-like [28] ReLU MLP with depth 5 and width 128 and a sinusoidal encoding of the input. As shown in Tab. 4 the quality of the MLP and permutohedral encoding are about the same, but the rendering speed of the permutohedral encoding is substantially faster.

Table 4. **Ablation of scene flow network** Both methods were trained for 45k steps which took 2:15h for the MLP and 2:05h for the permutohedral encoding. The training parameters of the MLP were found via grid search. Larger MLPs yielded only marginally improvements while smaller ones decreased heavily in quality. Metrics are averaged over the scenes *Balloon1*, *Truck* and *Playground*.

	Render s/img	Full Image			Dynamic Regions		
		PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS
MLP	15.1	<b>26.8</b>	<b>0.805</b>	0.124	<b>21.8</b>	<b>0.643</b>	<b>0.193</b>
Permuto	11.8	26.4	0.799	<b>0.118</b>	21.5	0.631	0.197

### B. Optimization

The network is trained over 60k steps with a batch size of 1024. Higher batch sizes are limited by the imposed restriction of being able to train on a single GPU, lower batch sizes

lead to underfitting of the scene flow. Adam [19] is used as the optimizer for training with  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$  and  $\epsilon = 10^{-8}$ . We experimented with a higher value for  $\beta_1$  to increase the momentum in the running average to compensate for only having one observation per training step, but found results to be worse. The learning rate is  $l_{flow} \in [5 \times 10^{-3}, 1.0 \times 10^{-4}]$  for the flow network and  $l_{GNT} \in [10^{-3}, 10^{-5}]$  for the GNT [44] fine-tuning. Every 20k steps, the learning rate is decreased by 50%. To continue our approach of coarse-to-fine learning of the scene flow, we follow the warm-up procedure of PermutoSDF [38] and first initialize the coarse levels and continuously include finer levels over the duration 12.5k training steps. For pixels marked as dynamic, we reduce  $\alpha_{slow}$  by 50% to encourage a large scene flow in those areas. Additionally, we reduce the weight of the image reconstruction loss by 25% for potentially dynamic pixels, to decrease blurriness in the image due to overfitting of the rendering backbone.

### C. Depth estimation

Using the attention weights of the ray transformer, it is possible to infer a depth value for each pixel in the estimated image [44]. This is possible, since the ray transformer will put the most attention on the points sampled along the ray that are contributing the most to the final pixel color, which most likely corresponds to a solid surface. Therefore, depth can be estimated by weighting the distance of points along the ray by their respective attention weights. An example for this is shown in Fig. 7. In future work, this could allow for additional supervision with monocular depth estimation methods.

### D. Rendering at non-observed times

Rendering at continuous target times  $\tilde{t} \in \mathbb{R}$  outside the intervals  $\Delta t$  in which the scene has been observed, is facilitated by initiating the motion adjustment by first adjusting the continuous times to the two neighbouring observations times, with

$$\tilde{t}_b = \left\lfloor \frac{\tilde{t}}{\Delta t} \right\rfloor \quad (18)$$

as the previous neighbour, and

$$\tilde{t}_f = \left\lceil \frac{\tilde{t}}{\Delta t} \right\rceil \quad (19)$$

as the succeeding neighbour. With  $\lfloor \cdot \rfloor$  we denote the floor operator and with  $\lceil \cdot \rceil$  the ceiling operator.

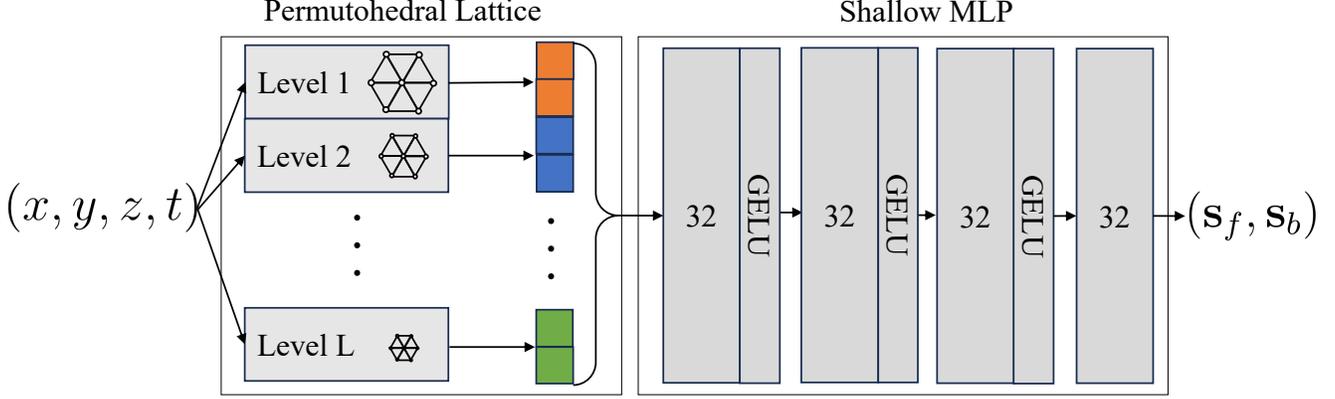


Figure 6. **Scene flow field architecture** For estimating the scene flow for a position and time, the 4D pose is projected into the different levels of the permutohedral lattice. The hashed positions of the surrounding vertices are used to extract latent vectors of length two from the respective hash-map. Interpolating the latent vectors of the vertices yields the final latent vector describing the local properties of the scene flow field at that level. The distinct latent vectors from each layer are then concatenated and inputted into a shallow 4-layer MLP with a width of 32. The MLP functions to decode the latent information to the actual scene flow.



Figure 7. **Depth estimation.** After rendering the target image (b), the distances of points along the camera ray are weighted by the corresponding ray attention weights and then summed to a depth estimate (c).

Using a scaling factor, given by:

$$\delta_b = \frac{\tilde{t}}{\Delta t} - \tilde{t}_b, \quad (20)$$

for the backward time, and respectively:

$$\delta_f = \tilde{t}_f - \frac{\tilde{t}}{\Delta t}, \quad (21)$$

for the forward time to then displace the ray points from the target time  $\tilde{t}$  to the adjacent discrete observation times. With this, forward time adjustments can be represented as:

$$\mathbf{p}_{\tilde{t} \rightarrow \tilde{t}_f} = \mathbf{p}_{\tilde{t}} + \delta_b \mathcal{S}_f(\mathbf{p}_{\tilde{t}}, \tilde{t}_b) - \delta_f \mathcal{S}_b(\mathbf{p}_{\tilde{t}}, \tilde{t}_f) \quad (22)$$

and backward time adjustments as:

$$\mathbf{p}_{\tilde{t} \rightarrow \tilde{t}_b} = \mathbf{p}_{\tilde{t}} + \delta_f \mathcal{S}_b(\mathbf{p}_{\tilde{t}}, \tilde{t}_f) - \delta_b \mathcal{S}_f(\mathbf{p}_{\tilde{t}}, \tilde{t}_b). \quad (23)$$

Following this initial step, the motion adjustment can continue as described for discrete times.

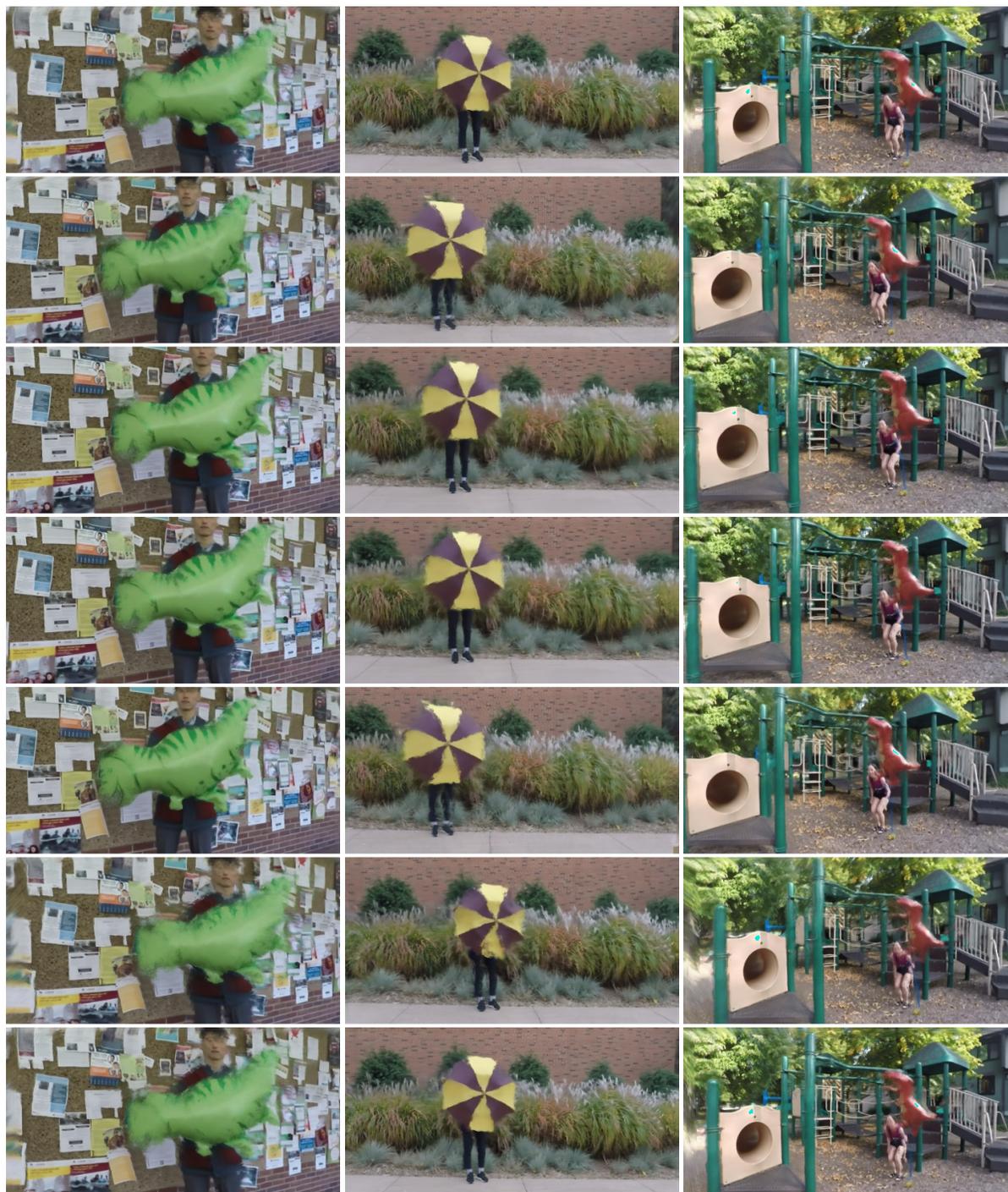
## E. Images sequences

In Fig. 8 we present image sequences of view interpolation for three different scenes. Overall, the method is able to consistently interpolate between the different target view-points without the sudden appearance of artifacts or flickering of the scene content. Nevertheless, the further the target viewpoint gets away from the nearest source observation, the more the dynamic content starts to blur because of the missing information from that perspective.

## F. Societal Impact

We anticipate several potential impacts of our proposed method, and similar methods, on society in the future. Primarily, we present a method to decrease the necessary training time for novel view synthesis methods for dynamic scenes and allow training on a single consumer-grade GPU which has the potential to democratize research capabilities, alleviating the dependency on specialized hardware. While our approach relies on the accumulated knowledge from pre-training, it is imperative to note that any biases

present in the used datasets might propagate through our system. However, as the essence of our research is to synthesize views that are as faithful to the ground truth as possible, we are actively addressing this challenge to ensure accuracy and integrity of our results.



(a) Scene: Balloon 2

(b) Scene: Umbrella

(c) Scene: Jumping

Figure 8. **Image sequences: Bullet time** Three image sequences of bullet time renderings, where the viewpoint is continuously changed while the time stays constant. Viewpoints are on an elliptical trajectory along the center of the camera rig.