HOLODIFFUSION: Training a 3D Diffusion Model using 2D Images

Supplementary material

1. Views2Voxel-grid Unprojection Mechanism

Given a training video s containing frames I_j, we generate a grid $\bar{V} \in \mathbb{R}^{d^V \times S \times S \times S}$ of auxiliary features $\bar{V}_{:mno} \in [-1, 1]^{d_V}$ by using the following procedure. We first project the 3D coordinate $\mathbf{x}^V_{mno}$ of each grid element (m, n, o) to every video frame I_j and sample corresponding 2D image features. The 2D image features f^j_{mno} are obtained using a ResNet32 [3] encoder $E(I_j)$. We use bilinear interpolation for sampling continuous values and use zero-features for projected points that lie outside the Image. Thus, we obtain N_{frames} features (corresponding to each frame in the video) for each grid element of the voxel-grid. We accumulate these features using the Accumulator MLP $\mathcal{A}_{acc}$. The accumulator $\mathcal{A}_{acc}$ takes as input $[f^j_{mno}; v^j]$, where $[;]$ denotes concatenation and v^j corresponds to the viewing direction corresponding to the camera center of j^{th} frame, and outputs $[\sigma^j_{mno}; {f'}^j_{mno}]$. Finally, we compute the feature at each of the voxel grid centers as a weighted sum of the newly mapped features:

$$F_{mno} = \sum_j \sigma^j_{mno} {f'}^j_{mno}. \tag{1}$$

2. Implementation Details

In this section, we provide more details related to implementing our proposed method.

2.1. Network Architectures

Our proposed pipeline (Fig 2. of main paper) contains three neural components: The Encoder, Diffusion UNet and Renderer. The Encoder network is a ResNet32 model [3]. For the main diffusion network, we use a 3D variant of the UNet used by Dhariwal and Nichol [1]. The model comprises residual blocks containing downsampling, upsampling, and self-attention blocks (with additive residual connections).

2.2. Renderer

In order to decode the generated voxel-grid of features into density and radiance fields, we use a NeRF-like [7] MLP (Multi-layer perceptron). The MLP contains 4 layers of 256 hidden units with a skip-connection on the 3rd hidden layer. The skip connection concatenates the input features with the intermediate hidden layer features. Similar to NeRF, and for the reasons described in Zhang et al. [9], we also input the view-directions at a latter layer in the MLP. The input features are not encoded, but we apply sinusoidal encodings [7, 8] to the input viewing directions with max frequency level $L = 4$. The activation functions used are: `LeakyReLU` for the hidden layers, `Softplus` for the density output head, and the `Sigmoid` for the radiance output head. All trainable weights are initialized using the Xavier uniform initialization [2]. Figure I shows the detailed architecture of the `RenderMLP`.

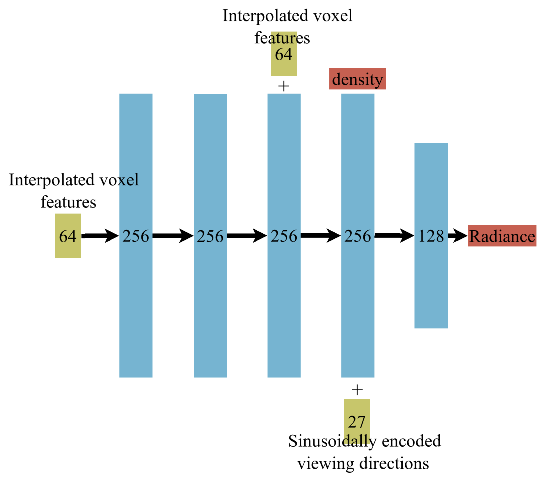
Interpolated voxel features
64
+
density
Interpolated voxel features
64
256
256
256
256
128
Radiance
+
27
Sinusoidally encoded viewing directions

Figure I. Architecture of the `RenderMLP` used for decoding the features of the generated voxel grids into density and radiance fields.

2.3. Training Details

We train the full HOLODIFFUSION pipeline for 1000 epochs over the dataset containing the object-centric videos. During training, we randomly sample 11 source views for unprojecting into the initial voxel-grid, and 1 target (reserved) novel view for computing loss. The latter enforces 3D structure in the generated samples. We use $L2$ distance between the rendered views and the G.T. views as the photometric-consistency loss. In terms of hardware, we train all our models on 4-8 32GB-V100 GPUs, with a `batch-size` equal to the number of GPUs in use, i.e., each GPU processes one voxel-grid during training. We use *Adam* [6] optimizer with a learning rate (α) of 0.00005 and default values of β_1, β_2, and ϵ for all the trainable networks during training.

2.4. Diffusion Details

We use the DDPM [4] diffusion-formulation for our bootstrap-latent-diffusion module as described in section 4.2 of the main paper. We use the default $t = 1000$ time-steps and the default β_t schedule in our experiments: wherein we set $\beta_0 = 0.0001; \beta_{999} = 0.02$. Rest of the β_t values are obtained by linearly interpolating between the β_0 and β_{999}. Finally, to improve the input conditioning of our diffusion module, we apply $tanh$ to the voxel features to constrain their values in the range of [-1, 1], as proposed in Karras et al. [5]. This allows us to apply [-1, 1] clipping during sampling.

References

[1] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794, 2021. 1

[2] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010. 1

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015. 1

[4] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020. 2

[5] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *arXiv preprint arXiv:2206.00364*, 2022. 2

[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Proc. ICLR*, 2015. 1

[7] 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. ECCV*, 2020. 1

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017. 1

[9] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492*, 2020. 1