Supplementary Material for

Infinite Nature: Perpetual View Generation of Natural Scenes from a Single Image

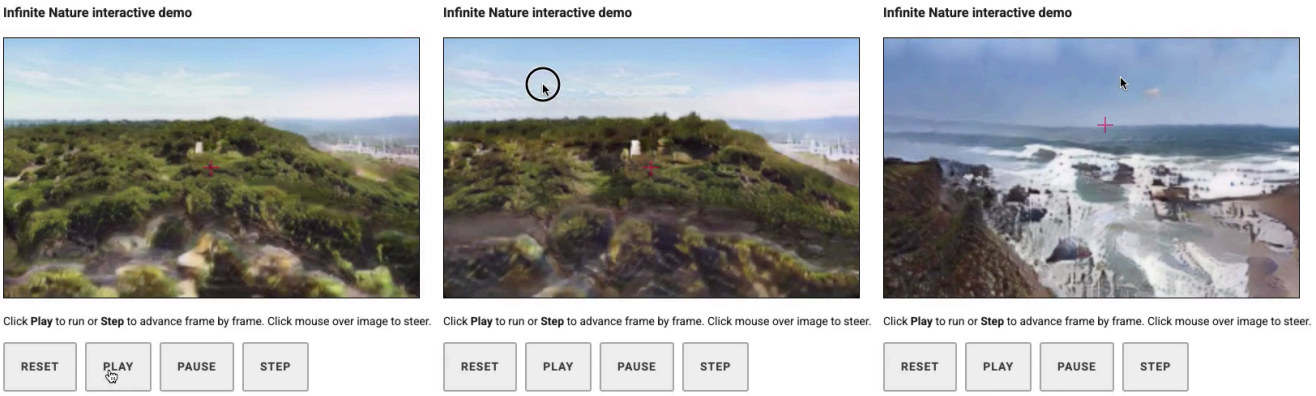

Figure 1. **Infinite Nature Demo.** We built a lightweight demo interface so a user can run Infinite Nature and control the camera trajectory. In addition, the demo can take any uploaded image, and the system will automatically run MiDaS to generate an initial depth map, then allow the user hit "play" to navigate through the generated world and click to turn the camera towards the cursor. The demo runs at several frames per second using a free Google Colab GPU-enabled backend. Please see our video for the full effect of generating an interactive scene flythrough.

1. Implementation Details

This section contains additional implementation details for our system, including data generation, network architecture, and inference procedure.

1.1. ACID Collection and Processing

To create the ACID dataset, we began by identifying over 150 proper nouns related to coastline and island locations such as *Big Sur*, *Half Moon Bay*, *Moloka'i*, *Shi Shi Beach*, *Waimea bay*, etc. We combined each proper noun with a set of keywords ($\{aerial, drone, dji, and mavic\}$) and used these combinations of keywords to perform YouTube video search queries. We combined the top 10 video IDs from each query to form a set of candidate videos for our dataset.

We process all the videos through a SLAM and SfM pipeline as in Zhou *et al*. [9]. For each video, this process yields a set of camera trajectories, each containing camera poses corresponding to individual video frames. The pipeline also produces a set of 3D keypoints. We manually identify and remove videos that feature a static camera or are not aerial, as well as videos that feature a large number of people or man-made structures. In an effort to limit the potential privacy concerns of our work, we also discard frames that feature people. In particular, we run the state of the art object detection network [6] to identify any humans present in the frames. If detected humans occupy more than 10% of a given frame, we discard that frame. The above filtering steps are applied in order to identify high-quality video sequences for training with limited privacy implications, and the remaining videos form our dataset.

Many videos, especially those that feature drone footage, are shot with cinematic horizontal borders, achieving a letterbox effect. We pre-process every frame to remove detected letterboxes and adjust the camera intrinsics accordingly to reflect this crop operation.

For the remaining sequences, we run the MiDaS system [4] on every frame to estimate dense disparity (inverse depth). MiDaS predicts disparity only up to an unknown scale and shift, so for each frame we use the 3D keypoints produced by running SfM to compute scale and shift parameters that best fit the MiDaS disparity values to the 3D keypoints visible in that frame. This results in disparity images that better align with the SfM camera trajectories during training. More specifically, the scale a and shift b are

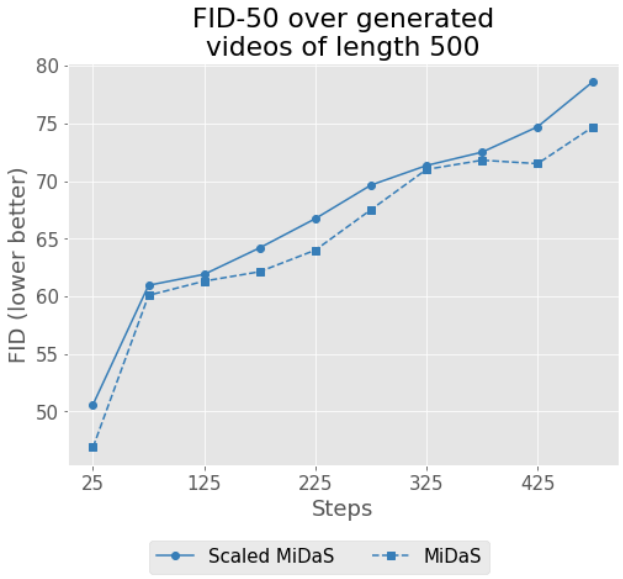

Figure 2. **Scaled MiDaS vs original MiDaS.** We scale the MiDaS disparity maps to be consistent with the camera poses estimated by SfM during training. At test-time our approach only requires a single image with disparity. Here we show results of FID-50 long generation using the original MiDaS output vs the scaled MiDaS. Despite being only trained on scaled disparity, our model still performs competitively with (unscaled) MiDaS as its input.

calculated via least-squares as:

$$\underset{a,b}{\operatorname{argmin}} \sum_{(x,y,z)\in\mathcal{K}} \left(a\hat{D}_{xyz} + b - z^{-1}\right)^2 \qquad (1)$$

where $\mathcal{K}$ is the set of visible 3D keypoints from the local frame's camera viewpoint, $\hat{D}$ is the disparity map predicted by MiDaS for that frame, and $\hat{D}_{xyz}$ is the disparity value sampled from that map at texture coordinates corresponding to the projection of the point (x, y, z) according to the camera intrinsics. The disparity map D we use during training and rendering is then $D = a\hat{D} + b$.

1.2. Inference without Disparity Scaling

Scaling and shifting the disparity as described above requires a sparse point cloud, which is generated from SfM and in turn requires video or multi-view imagery. At test-time, however, we assume only a single view is available. Fortunately, this is not a problem in practice, as scaling and shifting the disparity is only necessary if we seek to compare generated frames at target poses against ground truth. If we just want to generate sequences, we can equally well use the original MiDaS disparity predictions. Fig. 2 compares long generation using scaled and original MiDaS outputs, and shows that using original MiDaS outputs has a negligible effect on the FID scores. Fig. 3 shows an example of a long sequence generated with the unscaled MiDaS prediction from a photo taken on a smartphone, demonstrating that our framework runs well on a single test image using the original MiDaS disparity.

1.3. Aligning Camera Speed

The speed of camera motion varies widely in our collected videos, so we normalize the amount of motion present in training image sequences by computing a proxy for camera speed. We use the translation magnitude of the estimated camera poses between frames after scale-normalizing the video as in Zhou *et al*. [9] to determine a range of rates at which each sequence can be subsampled in order to obtain a camera speed within a desired target range. We randomly select frame rates within this range to subsample videos. We picked a target speed range for training sequences that varies by up to 30% and, on average, leaves 90% of an image's content visible in the next sampled frame.

1.4. Network Architecture

We use Spatially Adaptive Normalization (SPADE) of Park *et al*. [3] as the basis for our refinement network. The generator consists of two parts, a variational image encoder and a SPADE generator. The variational image encoder maps a given image to the parameters of a multivariate Gaussian that represents its feature. We can use this new distribution to sample GAN noise used by the SPADE generator. We use the initial RGBD frame of a sequence as input to the encoder to obtain this distribution before repeatedly sampling from it (or using its mean at test-time) at every step of refinement.

Our SPADE generator is identical to the original SPADE architecture, except that the input has only 5 channels corresponding to RGB texture, disparity, and a mask channel indicating missing regions.

We also considered a U-net [5]–based approach by using the generator implementation of Pix2Pix [2], but found that such an approach struggles to achieve good results, taking longer to converge and in many cases, completely failing when evaluating beyond the initial five steps.

As our discriminator, we use the Pix2PixHD [7] multi-scale discriminator with two scales over generated RGBD frames. To make efficient use of memory, we run the discriminator on random crops of pixels and random generated frames over time.

1.5. Loss Weights

We used a subset of our training set to sweep over checkpoints and hyperparameter configurations. For our loss, we used $\lambda_{\text{reconst}} = 2$, $\lambda_{\text{perceptual}} = 0.01$, $\lambda_{\text{adversarial}} = 1$, $\lambda_{\text{KLD}} = 0.05$, $\lambda_{\text{feat matching}} = 10$.

1.6. Data Source for Qualitative Illustrations

Note that for license reasons, we do not show generated qualitative figures and results on ACID. Instead, we col-

Figure 3. **Generation from smartphone photo.** Our perpetual view generation applied to a photo captured by the authors on a smartphone. We use MiDaS for the initial disparity, and assume a field of view of 90°.

lect input images with open source licenses from Pexels [1] and show the corresponding qualitative results in the paper and the supplementary video. The quantitative results are computed on the ACID test set.

1.7. Auto-pilot View Control

We use an auto-pilot view control algorithm when generating long sequences from a single input RGB-D image. This algorithm must generate the camera trajectory in tandem with the image generation, so that it can avoid crashing into the ground or obstacles in the scene. Our basic approach works as follows: at each step we take the current disparity image and categorize all points with disparity below a certain threshold as *sky* and all points with disparity above a second, higher threshold as *near*. (In our experiments these thresholds are set to 0.05 and 0.5.) Then we apply three simple heuristics for view-control: (1) look up or down so that a given percentage (typically 30%) of the image is *sky*, (2) look left or right, towards whichever side has more *sky*, (3) If more than 20% of the image is *near*, move up (and if less, down), otherwise move towards a horizontally-centered point 30% of the way from the top of the image. These heuristics determine a (camera-relative) *target look direction* and *target movement direction*. To ensure smooth camera movement, we interpolate the actual look and movement directions only a small fraction (0.05) of the way to the target directions at each frame. The next camera pose is then produced by moving a set distance in the move direction while looking in the look direction. To generate a wider variety of camera trajectories (as for example in Section 3.4), or to allow user control, we can add an offset to the target look direction that varies over time: a horizontal sinusoidal variation in the look direction, for example, generates a meandering trajectory.

This approach generates somewhat reasonable trajectories, but an exciting future direction would be to train a model that learns how to choose each successive camera pose, using the camera poses in our training data.

We use this auto-pilot algorithm to seamlessly integrate user control and obstacle avoidance in our demo interface which can be seen in Fig. 1.

1.8. Additional Frame Interpolation

For the purposes of presenting a very smooth and cinematic video with a high frame rate, we can additionally interpolate between frames generated by our model. Since our system produces not just RGB images but also disparity, and since we have camera poses for each frame, we can use this information to aid the interpolation. For each pair of frames (P_t, I_t, D_t) and $(P_{t+1}, I_{t+1}, D_{t+1})$ we proceed as follows:

First, we create additional camera poses (as many as desired) by linearly interpolating position and look-direction between P_t and P_{t+1}. Then, for each new pose P a fraction λ of the way between P_t and P_{t+1}, we use the differentiable renderer $\mathcal{R}$ to rerender I_t and I_{t+1} from that viewpoint, and blend between the two resulting images:

$$\begin{aligned} I'_t &= \mathcal{R}(I_t, D_t, P_t, P), \\ I'_{t+1} &= \mathcal{R}(I_{t+1}, D_{t+1}, P_{t+1}, P), \\ I &= (1-\lambda) I'_t + \lambda I'_{t+1}, \end{aligned} \tag{2}$$

Note: we apply this interpolation to the long trajectory sequences in the supplementary video only, adding four new frames between each pair in the sequence. However, all short-to-mid range comparisons and all figures and metrics in the paper are computed on raw outputs without any interpolation.

1.9. Aerial Coastline Imagery Dataset

Our ACID dataset is available from our project page at https://infinite-nature.github.io, in the same format as RealEstate10K[9]). For each video we identified as aerial footage of nature scenes, we identified multiple frames for which we compute structure-from-motion poses and intrinsics within a globally consistent system. We divide ACID into train and test splits.

To get test sequences used during evaluation, we apply the same motion-based frame subsampling described in Section 1.3 to match the distribution seen during training for all

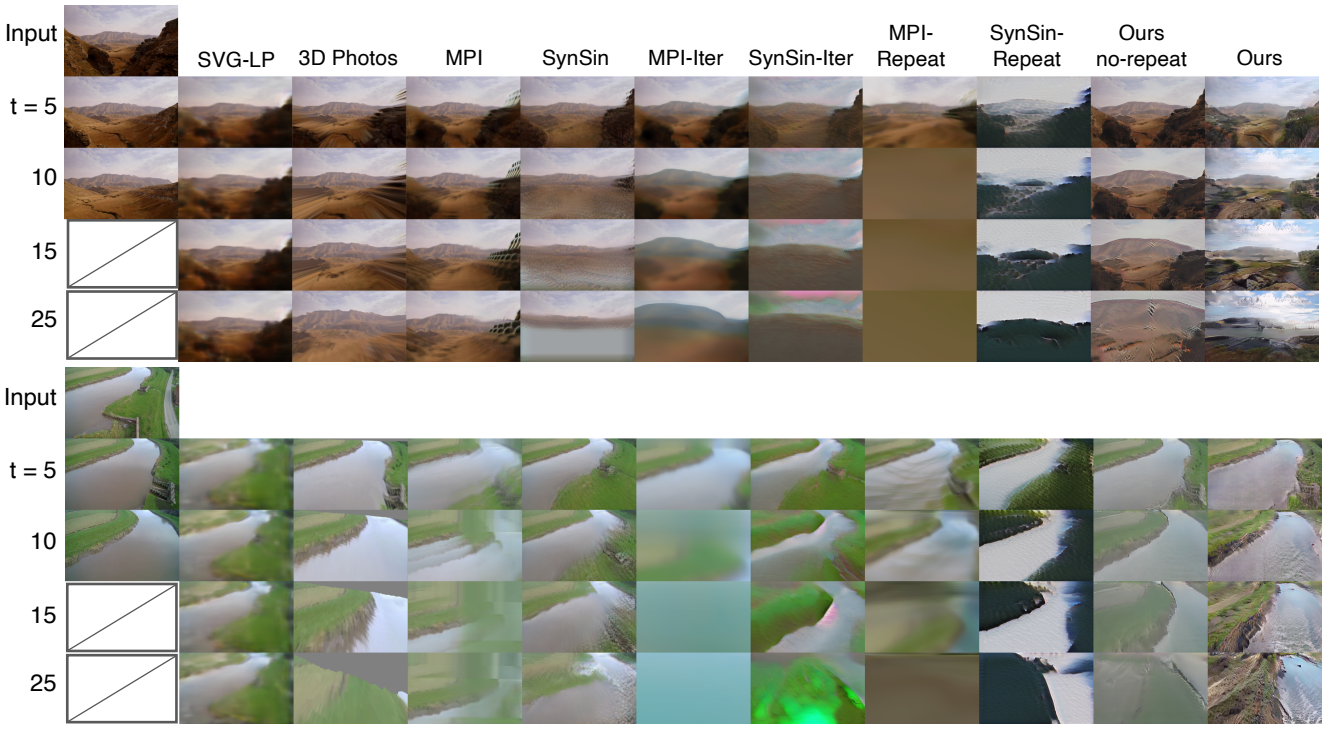

Figure 4. **Additional Qualitative Comparisons.** As in Figure 6 in the main paper, we show more qualitative view synthesis results on various baselines. Notice how other methods produce artifacts like stretched pixels (3D Photos, MPI), or incomplete outpainting (3D Photos, SynSin, Ours no-repeat) or fail to completely move the camera (SVG-LP). Further iter and repeat variants do not improve results. Our approach generates realistic looking images of zoomed in views that involves adding content and super resolving stretched pixels.

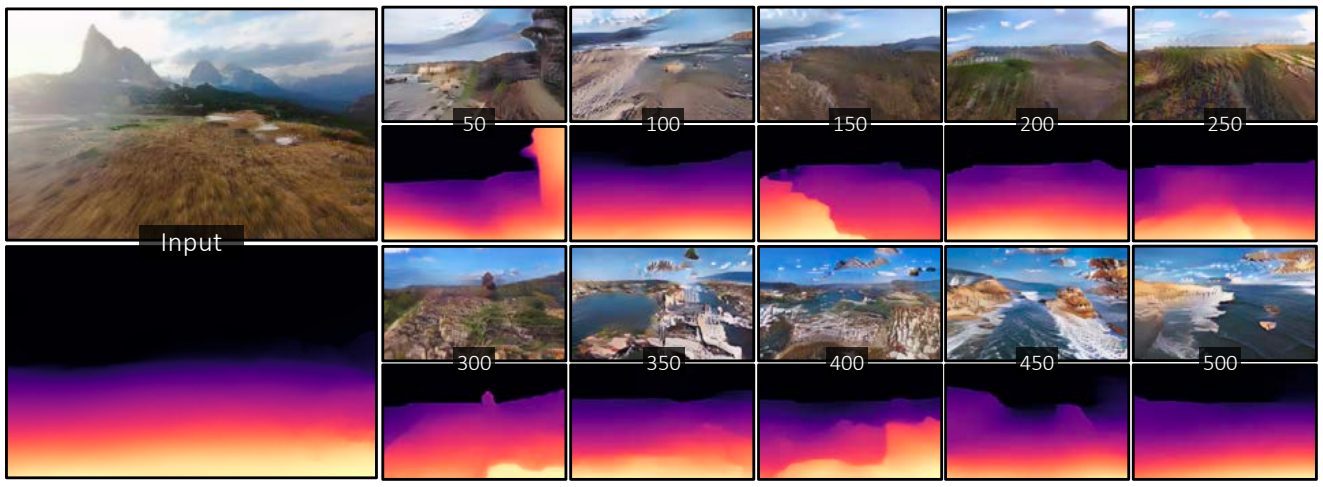

Figure 5. **Long Generation with Disparity.** We show generation of a long sequence with its corresponding disparity output. Our render-refine-repeat approach enables refinement of both geometry and RGB textures.

view synthesis approaches. Further we constrain test items to only include forward motion which is defined as trajectories that stay within a 90° frontal cone of the first frame. This was done to establish a fair setting with existing view synthesis methods which do not incorporate generative aspects. These same test items were used in the 50-frame FID experiments by repeatedly extrapolating the last two known poses to generate new poses. For the 500-generation FID, we compute future poses using the auto-pilot control described in Section 1.7. To get "real" inception statistics to compare with, we use images from ACID.

2. Experimental implementation

2.1. SynSin training

We first trained Synsin [8] on our nature dataset with the default training settings (i.e. the presets used for the KITTI model). ==We then modified the default settings by changing the camera stride in order to train Synsin to perform better for the task of longer-range view synthesis. Specifically, we employ the same motion-based sampling for selecting pairs of images as described in Section 1.3. However, here we increase the upper end of the desired motion range by a factor of 5, which allow the network to train with longer camera strides. This obtains a better performance than the default setting, and we use this model for all Synsin evaluations. We found no improvement going beyond 5× camera motion range. We also implemented an exhaustive search for desirable image pairs within a sequence to maximize the training data.

We also experimented with *SynSin-iter* to synthesize long videos by applying the aforementioned trained SynSin in an auto-regressive fashion at test time. But this performed worse than the direct long-range synthesis.

In addition to this, we also consider the repeat variant. *SynSin-repeat* was implemented using a similar training setup, however instead we also train SynSin to take its own output and produce the next view for $T = 5$ steps. Due to memory and engineering constraints, we are unable to fit *SynSin-repeat* with the original parameters into memory, so we did our best by by reducing the batch size while keeping as faithful to the original implementation. While this does not indicate SynSin fails at perpetual view generation, it does suggest that certain approaches are better suited to solve this problem.

3. Additional Analysis and Results

This section contains additional results and analysis to better understand Infinite Nature's behavior. In Fig. 4, we show additional view synthesis results given an input image across various baselines.

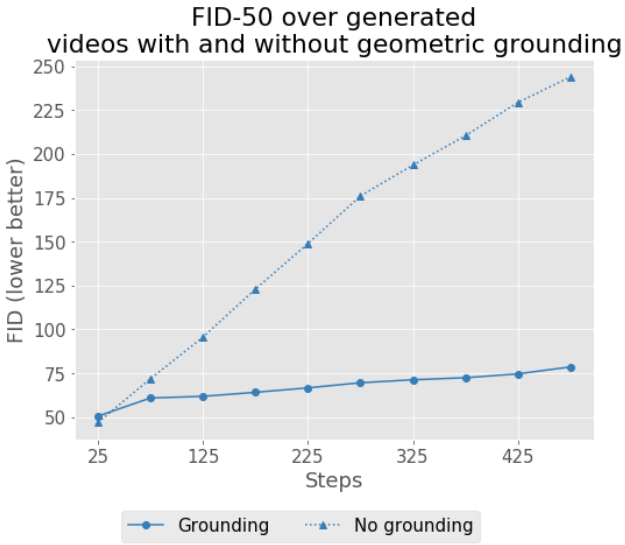

Figure 6. **Geometric Grounding Ablation.** Geometric grounding is used to explicitly ensure disparities produced by the refinement network match the geometry given by its input. We find this important as otherwise subtle drift can cause the generated results to diverge quickly as visible in Fig. 7.

3.1. Limitations

As discussed in the main paper, our approach is essentially a memory-less Markov process that does not guarantee global consistency across multiple iterations. This manifests in two ways: First on the geometry, *i.e*. when you look back, there is no guarantee that the same geometric structure that was observed in the past will be there. Second, there is also no global consistency enforced on the appearance——the appearance of the scene may change in short range, such as sunny coastline turning into a cloudy coastline after several iterations. Similarly, after hundreds of steps, two different input images may end up in a scene that has similar stylistic appearance, although never exactly the same set of frames. Adding global memory to a system like ours and ensuring more control over what will happen in the long range synthesis is an exciting future direction.

3.2. Disparity Map

In addition to showing the RGB texture, we can also visualize the refined disparity to show the geometry. In Fig. 5, we show the long generation as well as its visualized disparity map in an unnormalized color scheme. Note that the disparity maps look plausible as well because we train our discriminator over RGB and disparity concatenated. Please also see our results in the supplementary video.

3.3. Effect of Disabling Geometric Grounding

We use geometric grounding as a technique to avoid drift. In particular we found that without this grounding, over a

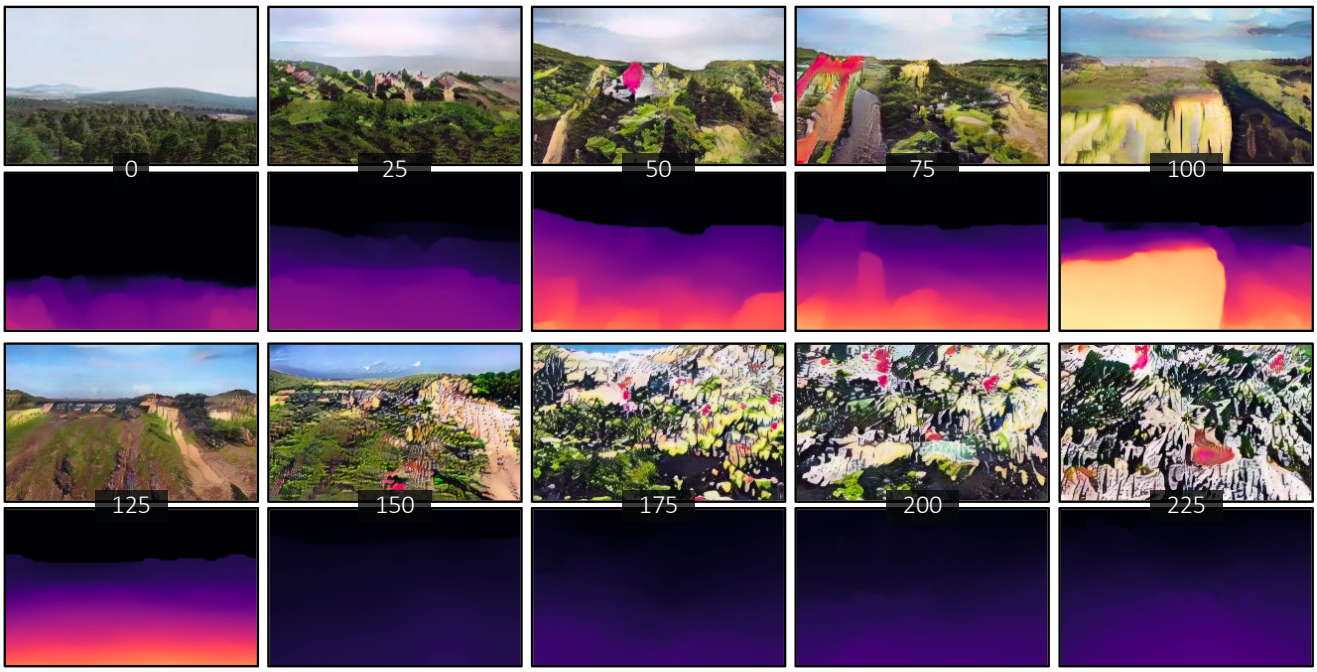

Figure 7. **Geometric Grounding Ablation.** An example of running our pretrained model on the task of long trajectory generation but *without* using geometric grounding. Disparity maps are shown using an *unnormalized* color scale. Athough the output begins plausibly, by the 150th frame the disparity map has drifted very far away, and subsequently the RGB output drifts after the 175th frame.

time period of many frames the render-refine-repeat loop gradually pushes disparity to very small (i.e. distant) values. Fig. 7 shows an example of this drifting disparity: the sequence begins plausibly but before frame 150 is reached, the disparity (here shown unnormalized) has become very small. It is notable that once this happens the RGB images then begin to deteriorate, drifting further away from the space of plausible scenes. Note that this is a test-time difference only: the results in Fig. 7 were generated using the same model checkpoint as our other results, but with geometric grounding disabled at test time. We show FID-50 results to quantitatively measure the impact of drifting in Fig. 6.

3.4. Results under Various Camera Motions

In addition to the demo, we also provide a quantitative experiment to measure how the model's quality changes with different kinds of camera motion over long trajectories. As described in Section 1.7, our auto-pilot algorithm can be steered by adding an offset to the target look direction. We add a horizontal offset which varies sinusoidally, causing the camera to turn alternately left and right every 50 frames. Fig. 8 compares the FID-50 scores of sequences generated where the relative magnitude of this offset is 0.0 (no offset), 0.5 (gentle turns), and 1.0 (stronger turns), and visualizes the resulting camera trajectories, viewed from above. This experiment shows that our method is resilient to different turning camera motions, with FID-50 scores that are comparable on long generation.

3.5. Generating Forward-Backwards Sequences

Because the *Render-Refine-Repeat* framework uses a memory-less representation to generate sequences, the appearance of content is not maintained across iterations. As a consequence, pixel content seen in one view is not guaranteed to be preserved later when seen again, particularly if it goes out of frame. We can observe such inconsistency by synthesizing forward camera motion followed by the same motion backwards (a palindromic camera trajectory), ending at the initial pose. While generating the forward sequence of frames, some of the content in the original input image will leave the field of view. Then, when synthesizing the backward motion, the model must regenerate this forgotten content anew, resulting in pixels that do not match the original input. Fig. 9 shows various input scenes generated for different lengths of forward-backward motion. The further the camera moves before returning to the initial position, the more content will leave the field of view, and so we find that that longer the palindromic sequence, the more the image generated upon returning to the initial pose will differ from the original input image.

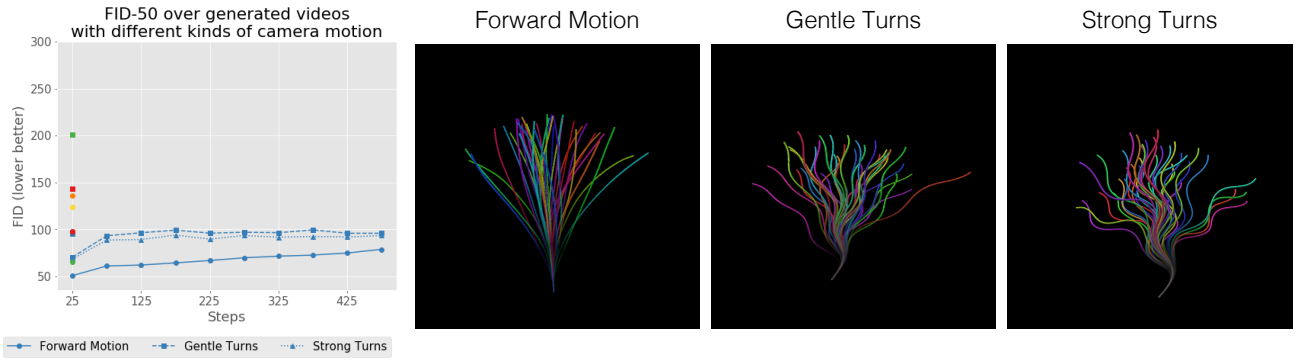

Figure 8. **FID with different camera motion.** We consider different types of camera motion generated by our auto-pilot algorithm with different parameters and its effect on generated quality. **Right:** Top-down view of three variations of camera motion that add different amounts of additional turning to the auto-pilot algorithm. **Left:** Even with strongly turning camera motion, our auto-pilot algorithm is able to generate sequences whose quality is only slightly worse than our full model evaluated only on forward translations. The unlabeled points refer to reported baselines on FID-50 from the main paper. See Section 3.4.

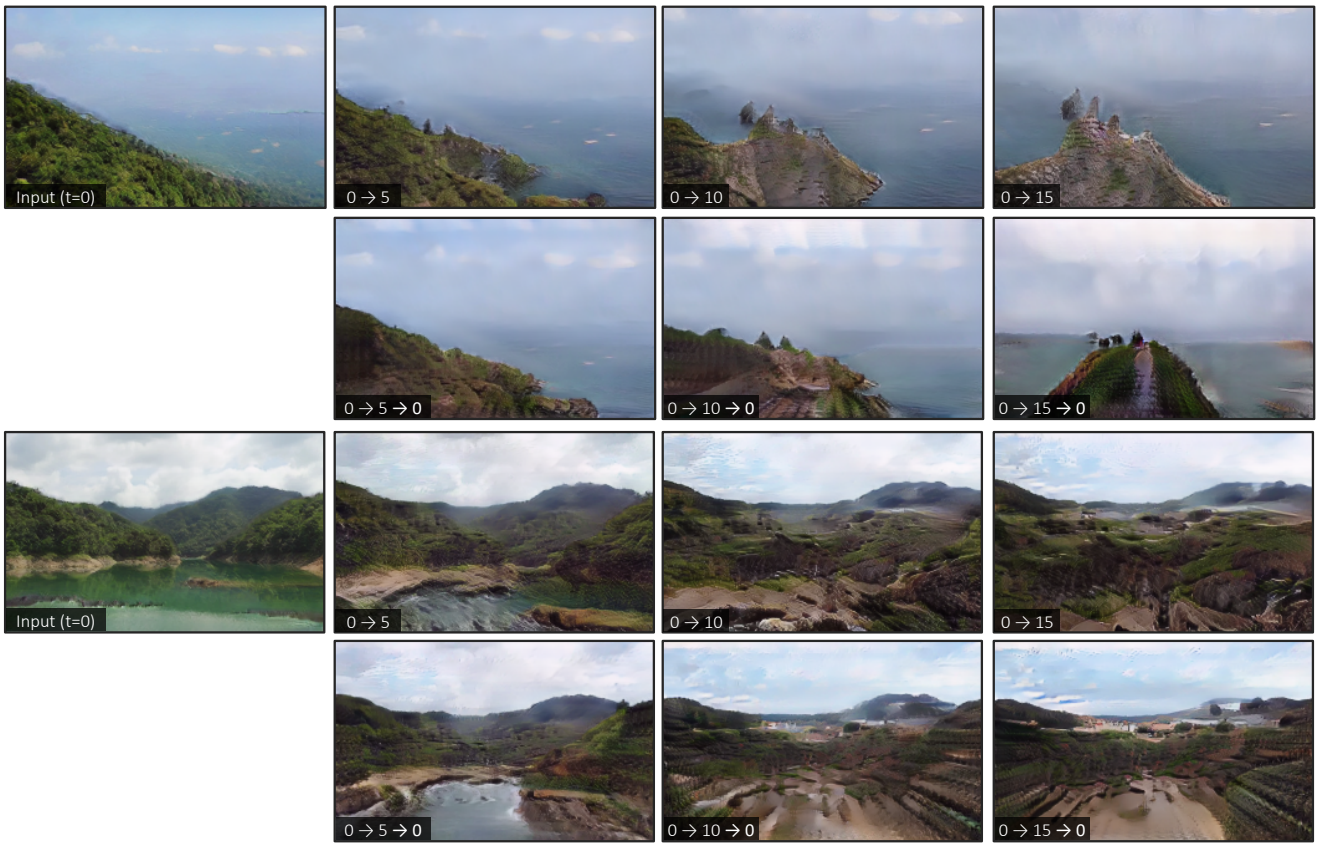

Figure 9. **Palindromic Poses.** Here we show Infinite Nature generated on palindromic sequences of poses of different lengths. Because our model uses a memory-less representation, the forward-backward motion requires the model to hallucinate content it has previously seen but which has gone out frame or been occluded, resulting in a generated image that does not match the original input.

References

[1] *Pexels.* Pexels provides high quality and completely free stock photos licensed under the Creative Commons Zero (CC0) license. All photos are tagged, searchable and easy to discover . 3

[2] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *CVPR*, 2017. 2

[3] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019. 2

[4] 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*, 2020. 1

[5] O. Ronneberger, P.Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015. (available on arXiv:1505.04597 [cs.CV]). 2

[6] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10781–10790, 2020. 1

[7] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018. 2

[8] Olivia Wiles, Georgia Gkioxari, Richard Szeliski, and Justin Johnson. SynSin: End-to-end view synthesis from a single image. In *CVPR*, 2020. 5

[9] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. *ACM Trans. Graph.*, 37(4):65:1–65:12, 2018. 1, 2, 3