PixelSynth: Generating a 3D-Consistent Experience from a Single Image: Supplemental Material

Chris Rockwell David F. Fouhey Justin Johnson
University of Michigan

Video results available on the paper website give a thorough sense of model quality and consistency. As stated in the paper, the proposed method tends to produce high-quality, consistent *scenes*. In contrast, baselines such as SynSin – 6X are unable to create content, and ablations such as No 3D Accumulation are wildly inconsistent. These results are best seen in video; and are available at this URL: https://crockwell.github.io/pixelsynth/#video_results

The pdf portion of the supplemental material shows: detailed descriptions of model architectures (Section 1); implementation details (Section 2); details about the experimental setup (Section 3); additional results (Section 4); and additional and A/B testing details (Section 5).

1. Model Architecture

As stated in the paper, a forward pass of the model takes in a single image and produces a consistent image in a novel view $\mathbf{p}$. This involves multiple components: a depth module D that maps images to depthmaps (producing point clouds); a projector π that projects a point cloud to a novel view; an outpainter O that can outpaint missing pixels by autoregressive modeling on the latent space of a VQ-VAE; and a refinement R module that adds details and corrects mistakes on a full image. We now provide more architectural details for each component; source code is available at https://github.com/crockwell/pixelsynth.

Depth Module D: The Depth Module takes in a $256 \times 256 \times 3$ image and predicts a depth for each pixel, yielding a $256 \times 256 \times 1$ depthmap. For fair comparison, we follow the U-Net used in SynSin [6], which consists of 8 encoding blocks that are mirrored by 8 decoding blocks.

Encoder: Each encoder block consists of a convolution (size: 4×4 / stride 2 / padding 1) followed by BatchNorm and leaky ReLU (negative slope of 0.2). Each block halves the width and height. The first convolution has 32 filters (mapping 3 channels to 32 channels); filter counts double at each block until reaching 256; they then remain constant.

Decoder: The decoder mirrors the encoder. Each block consists of a ReLU, $2\times$ bilinear upsampling, convolution (3×3 / stride 1 / padding 1), and BatchNorm (except the last layer). Mirroring the encoder, filter counts remain the same (256) until the feature map has been upsampled to $\frac{1}{16}$th of the input size. Filter counts then start halving. The last block does not have BatchNorm and has a final tanh at the top of the network.

Projector π: The Projector takes a colored point cloud and pose $\mathbf{p}$ and projects it as if seen at $\mathbf{p}$. This produces a $256 \times 256 \times 3$ image along with an indication that of which pixels were projected to and which need to be outpainted. We implement the projection with the point cloud rendering functions from Pytorch3D [3]. Our design decisions follow SynSin [6] for fair comparison: We alpha-composite points in the z-buffer and accumulate within a radius of 4 pixels.

We find that one change is important for autoregressive outpainting: we do not consider reprojected pixels at the edge of the visible region. Pixels that fall just outside the projected point cloud's silhouette can be non-zero: each rendered pixel is a function of the projected points within a radius, and points just outside the silhouette are the result of interpolating some points and the missing regions. If we do not remove this, autoregressive outpainting begins with a border that is the mean color, which it tends to continue. We prevent this by treating border pixels as background/to-be-outpainted.

Outpainter O: The Outpainter takes as input the 256×256 reprojection from the Projector, possibly including large missing regions, and outpaints to a full image. This consists of performing autoregressive outpainting on the latent space of an autoencoder. Crucially, the autoregressive model follows an image-specific order because each image and new pose yields different missing regions. We describe the autoencoder, followed by the autoregressive model, then the autoregressive order. Our design decisions aim to make lightweight versions of VQVAE2 [4] for the autoencoder and the PixelCNN++ [5] used in Locally Masked Convolutions [2] for the autoregressive model.

Autoencoder: We follow a lightweight adaptation of VQ-

VAE2 [4] that maps a $256 \times 256 \times 3$ input to a $32 \times 32 \times 1$ quantized embedding space Z and back. The encoder consists of first 3 convolution blocks followed by 2 ResNet blocks, then 2 convolution blocks and 2 ResNet blocks. The encoder produces a $32 \times 32 \times 64$ continuous output; each pixel in the encoded continous space is quantized to an embedding $Z_{i,j,1} \in \mathbb{Z}_1^{512}$. The decoder first upsamples using a transpose convolution followed by a convolution. Then, it mirrors the encoder with 2 ResNet blocks followed by 2 transpose convolution blocks to produce a $256 \times 256 \times 3$ output.

Autoregressive model: The autoregressive model is a lightweight PixelCNN++ [5] that produces, as output, a distribution over the 512 possible quantized embedding values. Every convolution in the network uses locally-masked convolutions [2] for custom completion ordering. We follow the general design used by Locally Masked Convolutions [2] on CIFAR-10 consisting of 30 Gated ResNet blocks with 160 filters. However, we reduce its computational cost by using 12 Gated ResNet blocks with 80 filters, keeping everything else constant. For more details, we refer the reader to [2].

Autoregressive ordering: Autoregressive outpainting follows an image-specific order. We must use this custom order because outpainting works best when one predicts adjacent pixels in a sequence using as much known data as possible. In some scenarios (e.g., extending a center crop), this can be achieved with a fixed order. However, in our case the particular points that must be outpainted depend on the depthmap as seen in the first image and the new pose from which it is projected.

Our order (Figure 4 of main paper) aims to go from closest out, following a spiral pattern. We achieve this by sorting the background/to-be-outpainted pixels in ascending distance to the center of mass of the foreground/projected-to pixels. We start with the closest pixel and add the closest adjacent point not in the generation order. We repeat this process until the entire image is ordering; ties due to pixels having equal distance are broken using a spiral pattern outwards from the center of mass.

Refinement Module R: The Refinement Module takes in a full $256 \times 256 \times 3$ outpainted image and produces the final, refined output of the same size $256 \times 256 \times 3$. This is trained adversarially and so it consists of a generator and discriminator.

Generator: The generator follows BigGAN [1] and SynSin [6] and consists of 8 ResNet blocks capped with a tanh. Following [1], there is an added downsampling block and with noise injection into BatchNorm throughout. Specifically, each ResNet block follows the following structure, consisting of two paths which are added. The first path consists of: a linear layer that injects noise followed by BatchNorm; ReLU; convolution (size: 3×3 / stride 1 / padding 1); a linear layer to inject noise followed by BatchNorm; ReLU; and convolution (size 3×3 / stride 1 / padding 1). The second path consists as a convolution (size 1×1 / stride 1 / padding 0), which is added to the input. Whenever a ResNet block downsamples, it uses average pooling to downsample the input during residual connection; whenever it upsamples, it uses bilinear upsampling.

Discriminator: The discriminator consists of 2 discriminator modules at different scales. Each discriminator contains 5 convolution blocks. Each block contains a convolution (size 4×4 / stride 2), followed by a Leaky ReLU (negative slope 0.2). The middle three blocks additionally contain an instance normalization layer between the conv and leaky ReLU.

2. Implementation Details

Outpainting Inference. The Outpainter's autoregressive model produces its outputs by sampling. The forward pass produces a probability distribution over the vector embedded classes for every missing pixel in the 32×32 image. We find that best results are obtained by generating a set of full completions, followed by selection, and by adjusting the sampling temperature used during inference to balance detail and error.

Sample selection. For each image, we generate 50 completions and select the best. We use a combination of the discriminator loss from R and a classifier entropy. The classifier is trained on MIT Places 365 [7]. Selection uses the average of ranks obtained by: (1) ranking in descending order of discriminator loss (since higher loss tends to correspond to issues with details); (2) ranking in ascending order of entropy (since sensible completions tend to be confident predictions of the classifier).

Sampling Temperature. Sampling temperature is important for balancing diversity and error. On Matterport, we use a sampling temperature of 0.5, which we find reduces strange completions but is still detailed. On RealEstate10K, we also use 0.5 temperature for the 1-completion model. Again, we see this temperature best balances realism with detailed completions. On RealEstate10K's 50-completion model, we find we can increase sampling temperature to 0.7. While this means more completions are not sensible, we can select the best using an automated method. Therefore, the final outputs are more detailed and are still realistic.

Generating Multiple Viewing Directions. To create scenes as approximated in Figure 1 and seen in the supplemental video, support views are synthesized in eight directions: up, left, down, right, up-left, up-right, down-left, and down-right. These directions are selected to give a sense of all directions, and can be used to synthesize interior views without additional outpainting thereafter. When using mul-

tiple support views, we accumulate in a similar manner to the first support view: we lift existing information to 3D, reproject into the new support view, and outpaint as needed. In other words, we do not outpaint the same region again.

3. Experimental Setup

As detailed in the paper, we use two datasets to evaluate. Matterport uses embodied agent navigation to select paired views, while RealEstate10K selects paired frames from real video clips. We therefore use different processes to achieve a shared goal of selecting image pairs with large angle change.

Matterport: Matterport selection is straightforward because of embodiment. This consists of randomly drawing angle change in an embodied agent with a maximum of 120° in each direction. We use a dataset of 3.6k pairs.

RealEstate10K: On RealEstate10K, it is harder to select large angle changes because we are instead selecting from pairs of images in real videos. We select pairs of images such that the pairs have at least 20 degrees angle change. In order to attain such pairs, we allow sampling from anywhere in video clips, which can be over 270 frames apart. To minimize the number of view changes so extreme that input and target view are in different rooms, we limit translation at 1.0 meters, and angle at 60 degrees. We only consider pairs that meet this criteria, rather than resampling. Our filtered test set chooses 3600 pairs from over 3.5 million possible pairs across over 2.4k video clips. All selected evaluation pairs are cached for replicability.

4. Additional Results

We report additional results, including video predictions, which provide the best simultaneous display of quality and consistency.

Additional Qualitative Results: We first report additional qualitative results. Video results give a thorough sense of model quality and consistency, and are in the project webpage. As stated in the paper, the proposed method tends to produce high-quality, consistent scenes. In contrast, baselines such as SynSin - 6X are unable to create content, and ablations such as No 3D Accumulation are wildly inconsistent. Additional frame-level generated images are available in Figures 1 and 2.

Additional Quantitative Results: We report more extensive results for generated image quality. This is an expansion on Table 2 in the paper. Although these automated metrics are poor measures for extrapolation, we present in more detail in Table 1 for completeness. Ground truth depth is available in Matterport, meaning visible and non-visible regions can be attained, similar to SynSin. The same is not true of RealEstate10K, which contains real videos for which extensive ground truth labeling is prohibitive.

We reiterate the caution from the paper that PSNR has poor correlation with perceived quality when there are multiple possible completions (for instance during outpainting): Appearance Flow is competitive with other methods on PSNR but loses to our proposed method 98% of the time in A/B testing.

Limitations: Our primary limitation is outpainting both consistent and detailed content, especially on large view changes. There is a trade-off between the two, as a greater diversity of samples is required for detail, but can result in inaccurate content. While the approach of rejection sampling can improve outpainting errors, they remain a challenge, particularly on Matterport. In fact, on Matterport we reduce sampling temperature to minimize inconsistent completions, which can result in less detail. For instance, in Supp. Figure 2 row 3 column 1, notice missing content tends to repeat visible content rather than ending visible objects and creating new ones.

Table 1: Full PSNR and Perc Sim: Traditional metrics such as PSNR are poor measures for extrapolation tasks, but are reported for reference.

Method	Matterport				RealEstate10K	
	PSNR ↑			Perc Sim ↓	PSNR ↑	Perc Sim ↓
	Both	InVis	Vis			
Tatarchenko *et al.*	13.72	13.59	15.24	3.82	10.63	3.98
Appearance Flow	13.16	13.11	14.75	3.68	11.95	3.95
Single-View MPI	-	-	-	-	12.73	3.45
SynSin	15.05	14.35	17.86	3.13	13.92	2.77
SynSin - Sequential	14.31	13.36	17.65	3.14	13.30	2.78
SynSin - 6X	15.52	14.94	17.98	3.16	14.17	2.78
SynSin - 6X, Sequential	15.61	15.07	17.92	3.17	14.21	2.73
Ours	14.60	13.58	18.08	3.17	13.10	2.88

5. Additional and A/B Testing Details

A/B testing is the primary measure for success throughout experiments. This is common in work on extrapolation, as automated metrics tend to struggle. We detail our A/B testing framework below.

All A/B testing follows a standard A/B testing paradigm where human workers are shown images and are asked which is preferred given an input image. Workers were given instructions and example images and labels. They then had to pass a qualifier assessing whether they understood the task. Workers were also monitored by gold standard sentinel labels. All annotations were gathered using thehive.ai, a website similar to Amazon Mechanical Turk. Tasks are detailed below, and we share worker instructions.

Evaluating Quality via A/B: For comparisons of quality, we use novel view synthesis. A/B testing presents workers with the input image reprojected into a new view, and asks

them to select the final image that makes more sense given this image. The reason we use reprojections as worker input instead of input images is it makes the A/B testing much clearer for workers. Using an input and rotation are very difficult to visualize, and thus difficult to compare across models. We also do not compare to ground truth output, as final images can vary drastically from ground truth and still be highly reasonable.

Instructions are shared in Figure 3. Ties are not allowed; final selection requires agreement of at least two workers. Reprojections use the learned depth from our model, which is effective on large view changes and therefore tend to be accurate, compared to ground truth images.

Evaluating Consistency via A/B: For consistency A/B testing, we use a similar novel view synthesis setup. However, instead of predicting one image, the model predicts two images such that the second generated image is half the rotation and translation of the first.

Workers are then asked to compare generated image pairs across methods. We ask them to do so by showing them a pair of images generated by two competing methods, pairs being stacked vertically. We do not use the input or ground truth images as the goal is not to judge quality, but only to judge consistency. Thus, even if one pair looks less realistic, it should be selected. Full instructions are displayed in Figure 4.

Consistency can be difficult for workers to judge as it requires attending to small regions of each pair of images that may be slightly different. We therefore take several steps to maximize worker success. We use fixed rotations of large size ($\sim 35^\circ$ horizontal, $\sim 17.5^\circ$ vertical) to ensure angle change is apparent. We also constrain the rotation so that it must be in the horizontal and vertical directions, as additionally using roll rotations can make transformations confusing. Movement is also limited to that related to embodied rotation, since movement opposing rotation can make consistency difficult to evaluate. Finally, rotation is randomly selected for each image from one of eight possible directions seen in Figure 1: up, left, down, right, up-left, up-right, down-left, and down-right. This allows us to explicitly specify rotation direction of each image pair to help workers attend to specific regions of image pairs.

Evaluating Consistency via Homography: We validate A/B consistency using PSNR and Perceptual Similarity via homography. We use the same setup as in A/B testing, but instead apply only pure rotations to images. This enables homographies to warp across generated images. We do so in each pair both from intermediate to extreme images and from extreme to intermediate images. PSNR is then calculated on overlapping regions, while Perc Sim is calculated on warped images with non-overlapping regions masked. Scores are averaged across both directions in each pair.

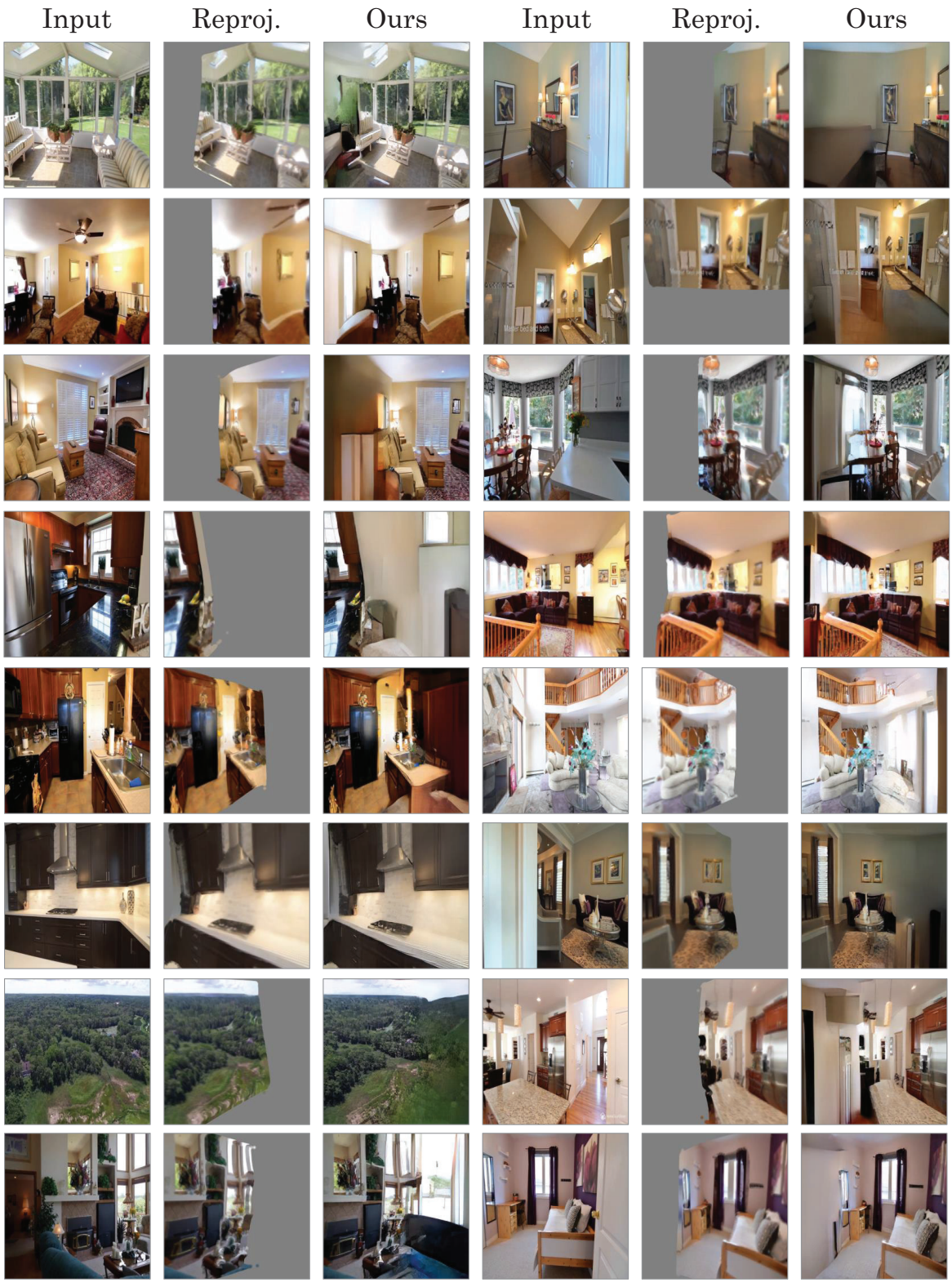


Figure 1: Additional Results on RealEstate10K.

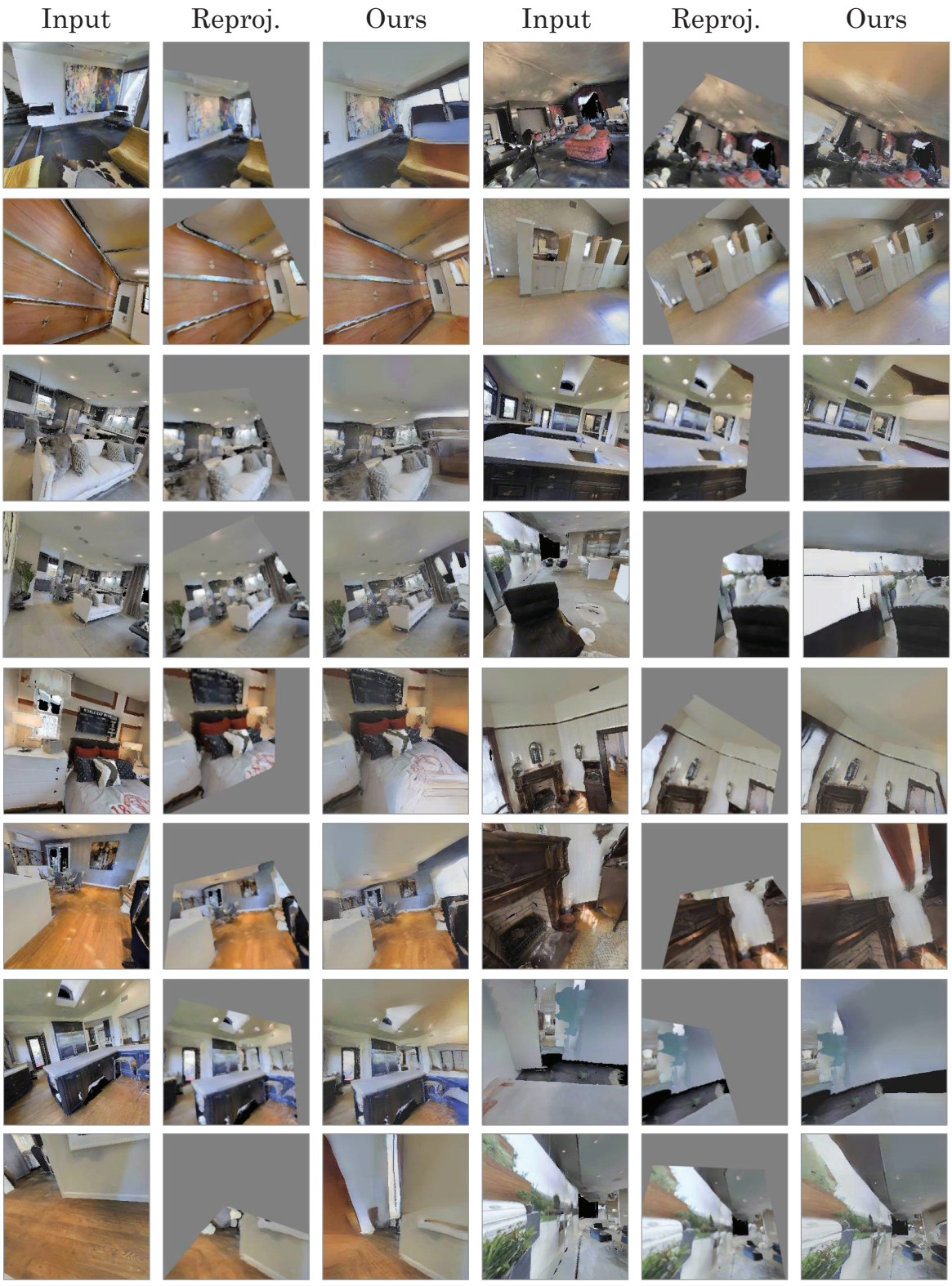


Figure 2: Additional Results on Matterport.

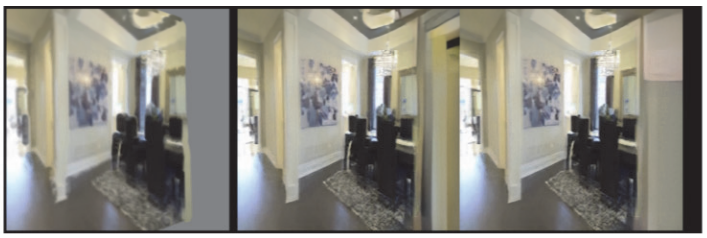
We have a system that is trying to generate images from part of the image. We'd like to identify cases where one version of the system is better than another version. Please select which of two output images is better given an input of part of the image.

Below are some examples.

[Middle] This time, the image in the middle has has a straighter wall than the one on the right. Also the model on the right has blurring

[Right] Sometimes, neither completion is great. However, the one in the middle seems to make less sense than the one on the right, which continues cabinets and feels more like the same room.

[middle] Both generations seem pretty reasonable. However, the one in the middle is more interesting than the one on the right. There is a door on the right side instead of a wall with a strange box near the top.



[middle]. While more detail is preferred, but here the detail does make much sense, and the middle example is better.

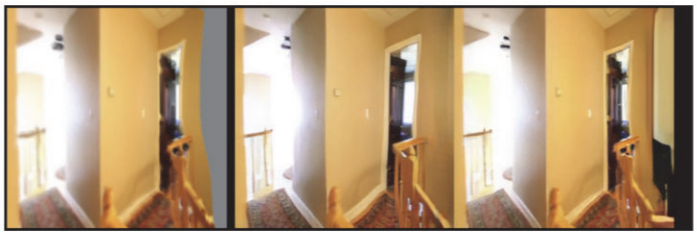


Figure 3: Quality A/B Worker Instructions.

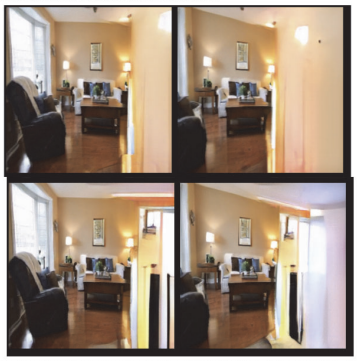
We have a system that is trying to generate consistent images across small image rotations. We'd like to identify cases where one version of the system is better than another version. Please select which of two output pairs is more consistent.

Below are some examples.

[Bottom] The left image is more consistent with the right image on the bottom, compared to the top. The top left image is not consistent with the top right. Note we don't care about quality of each image, only the consistency of the pair (top, or bottom).

[Bottom]. Here, the bottom left image is more consistent with the bottom right, compared to the top left and top right. The top left has some strange blue area in the left image that does not appear in the right image.

[Top] Look on the right side of both images. The bottom image on the right looks very different than the bottom image on the left. The left image has yellow and black area on its right side, while the one on the right has a pink area to the right of the yellow and black. The top pair have a consistent orange / yellow in the same area.



[Top]. Look at the bottom pair of images. The right image has a different shading on the very left side, as opposed to the left image, which has a different shade. So these images are not consistent. Meanwhile, the top pair of images continues a smooth brown (wall) across the pair.

Figure 4: Consistency A/B Worker Instructions.

References

[1] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *ICLR*, 2019. 2

[2] Ajay Jain, Pieter Abbeel, and Deepak Pathak. Locally masked convolution for autoregressive models. In *Conference on Uncertainty in Artificial Intelligence*, pages 1358–1367. PMLR, 2020. 1, 2

[3] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv preprint arXiv:2007.08501*, 2020. 1

[4] Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. In *NeurIPS*, pages 14866–14876, 2019. 1, 2

[5] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P. Kingma. Pixelcnn++: A pixelcnn implementation with discretized logistic mixture likelihood and other modifications. In *ICLR*, 2017. 1, 2

[6] Olivia Wiles, Georgia Gkioxari, Richard Szeliski, and Justin Johnson. Synsin: End-to-end view synthesis from a single image. In *CVPR*, pages 7467–7477, 2020. 1, 2

[7] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017. 2