Texture Mixer: A Network for Controllable Synthesis and Interpolation of Texture

Ning Yu[1,2,4] Connelly Barnes[3,4] Eli Shechtman[3] Sohrab Amirghodsi[3] Michal Lukáč[3]
[1]University of Maryland [2]Max Planck Institute for Informatics
[3]Adobe Research [4]University of Virginia
ningyu@mpi-inf.mpg.de connelly@cs.virginia.edu {elishe, tamirgho, lukac}@adobe.com

Abstract

This paper addresses the problem of interpolating visual textures. We formulate this problem by requiring (1) by-example controllability and (2) realistic and smooth interpolation among an arbitrary number of texture samples. To solve it we propose a neural network trained simultaneously on a reconstruction task and a generation task, which can project texture examples onto a latent space where they can be linearly interpolated and projected back onto the image domain, thus ensuring both intuitive control and realistic results. We show our method outperforms a number of baselines according to a comprehensive suite of metrics as well as a user study. We further show several applications based on our technique, which include texture brush, texture dissolve, and animal hybridization [1].

1. Introduction

Many materials exhibit variation in local appearance, as well as complex transitions between different materials. Editing materials in an image, however, can be highly challenging due to the rich, spatially-varying material combinations as we see in the natural world. One general research challenge then is to attempt to enable these kinds of edits. In particular, in this paper, we focus on textures. We define "texture" as being an image-space representation of a statistically homogeneous material, captured from a top-down view. We further focus on allowing a user to both be able to accurately control the placement of textures, as well as create plausible transitions between them.

Because of the complex appearance of textures, creating transitions by interpolating between them on the pixel domain is difficult. Doing so naïvely results in unpleasant artifacts such as ghosting, visible seams, and obvious repetitions. Researchers in texture synthesis have therefore developed sophisticated algorithms to address this problem. These may be divided to two families: non-parametric methods such as patch-based synthesis (e.g. [10, 9, 2]) and parametric methods (e.g. [15, 32]), including neural network synthesis approaches (e.g. [11, 37, 20, 26, 27]). Previously, researchers used sophisticated patch-based interpolation methods [7, 8] with carefully crafted objective functions. However, such approaches are extremely slow. Moreover, due to the hand-crafted nature of their objectives, they cannot learn from a large variety of textures in the natural world, and as we show in our comparisons are often brittle and frequently result in less pleasing transitions. Further, we are not aware of any existing feedforward neural network approaches that offer both fine-grained controllable synthesis and interpolation between multiple textures. User-controllable texture interpolation is substan-

[1]Demos, videos, code, data, models, and supplemental material are available at GitHub.

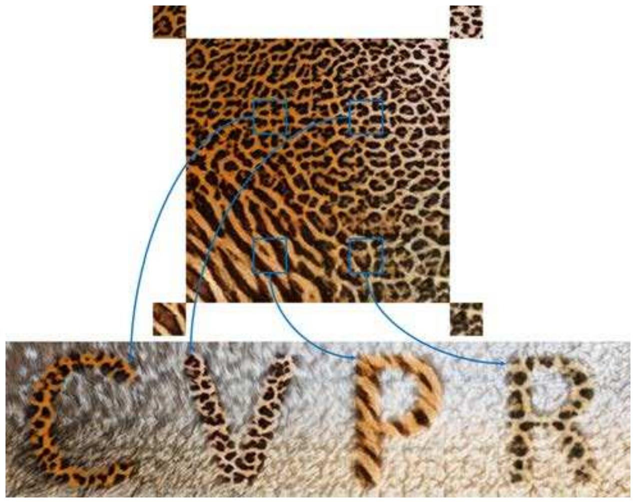
Figure 1. Texture interpolation and texture painting using our network on the *animal texture* dataset. The top part shows a 1024×1024 palette created by interpolating four source textures at the corners outside the palette. The bottom part shows a 512×2048 painting of letters with different textures sampled from the palette. The letters are interpolated by our method with the background, also generated by our interpolation.

tially more challenging than ordinary texture synthesis, because it needs to incorporate adherence to user-provided boundary conditions and a smooth transition for the interpolated texture.

In our paper, we develop a neural network approach that we call "Texture Mixer," which allows for both user control and interpolation of texture. We define the *interpolation* of texture as a broad term, encompassing any combination of: (1) Either gradual or rapid *spatial transitions* between two or more different textures, as shown in the palette, the letters, and the background in Figure 1, and (2) *Texture dissolve*, where we can imagine putting two textures in different layers, and cross-dissolving them according to a user-controlled transparency, as we show in our video. Previous neural methods can create interpolations similar to our dissolves by changing the latent variable [17, 21, 27, 28, 5]. Thus, in this paper we focus primarily on high-quality spatial interpolation: this requires textures to coexist in the same image plane without visible seams or spatial repetitions, which is more difficult to achieve. Our feedforward network is trained on a large dataset of textures and runs at interactive rates.

Our approach addresses the difficulty of interpolating between textures on the image domain by projecting these textures onto a latent domain where they may be linearly interpolated, and then decoding them back into the image domain to obtain the desired result. In order to satisfy the two goals of *controllability* and *visual realism*, we train our network simultaneously for both tasks. A *reconstruction task* ensures that when a texture is passed through an encoder and then a decoder (an autoencoder), the result will be similar to the input. This allows the user to specify texture at any given point of the output by example. An *interpolation task* uses a discriminator to ensure that linear interpolations of latent tensors also decode into plausible textures, so that the regions of the output not directly specified by the user are realistic and artifact-free. For this task, we can view our network as a conditional Generative Adversarial Network (GAN). In effect, we thus train an autoencoder and a conditional GAN at the same time, using shared weights and a shared latent space.

To perform the interpolation task, we take texture samples that user specifies, and project them into latent space using a learned encoder. Given these latent tensors, our network then uses three intuitive latent-space operations: tiling, interpolation, and shuffling. The tiling operation extends a texture spatially to any arbitrary size. The interpolation operation uses weighted combinations of two or more textures in latent domain. The shuffling operation swaps adjacent small squares within the latent tensor to reduce repetitions. These new latent tensors are then decoded to obtain the interpolated result.

Our main contributions are: (1) a novel interactive technique that allows both user control and interpolation of texture; (2) several practical and creative applications based on our technique; (3) a new suite of metrics that evaluate user controllability, interpolation smoothness, and interpolation realism; and (4) the state-of-the-art performance superior to previous work both based on these metrics, and based on a user study if we consider them holistically.

2. Related Work

The problem of user-controllable texture interpolation has so far been under-explored. It is however closely related to several other problems, most significantly texture synthesis, inpainting, and stylization.

Texture synthesis algorithms can be divided into two families. The first one is parametric, with a generative texture model. These algorithms include older, non-neural methods [15, 32], and also more recent deep learning-based methods that are based on optimization [11, 12, 33, 35] or trained feedforward models [37, 20, 26, 27]. Where the underlying model allows spatially varying weights for combination, it may be used to cross-dissolve textures. However, we are not aware of any existing texture synthesis techniques in this family that enables spatial transition between different textures.

The second family of texture synthesis algorithms is non-parametric, in which the algorithm produces output that is optimized to be as close as possible to the input under some appearance measure [10, 38, 9, 24, 23, 30, 25, 39, 2, 7, 22]. These can be formulated to accept two different inputs and spatially vary which is being compared to, facilitating interpolation [7, 8]. As we mentioned before, such approaches are slow, and due to the hand-crafted nature of their objectives, they tend to be brittle.

Recently, generative adversarial networks (GANs) [13, 34, 1, 14] have shown improved realism in image synthesis and translation tasks [18, 45, 46]. GANs have also been used directly for texture synthesis [26, 19, 44], however, they were limited to a single texture they were trained on. A recent approach dubbed PSGAN [3] learns to synthesize a collection of textures present in a single photograph, making it more general and applicable to texture interpolation; it is not, however, designed for our problem as it cannot interpolate existing images. We show comparisons with PSGAN and it cannot reconstruct many input textures, even after running a sophisticated optimization or jointly associating PSGAN with an encoder. Moreover, PSGAN can suffer from mode collapse.

Texture synthesis and image inpainting algorithms are often closely related. A good hole filling algorithm needs to be able to produce some sort of transition between textures on opposite ends of the hole, and so may be used in a texture interpolation task. A few recent deep learning-based methods showed promising results [40, 42, 29, 41].

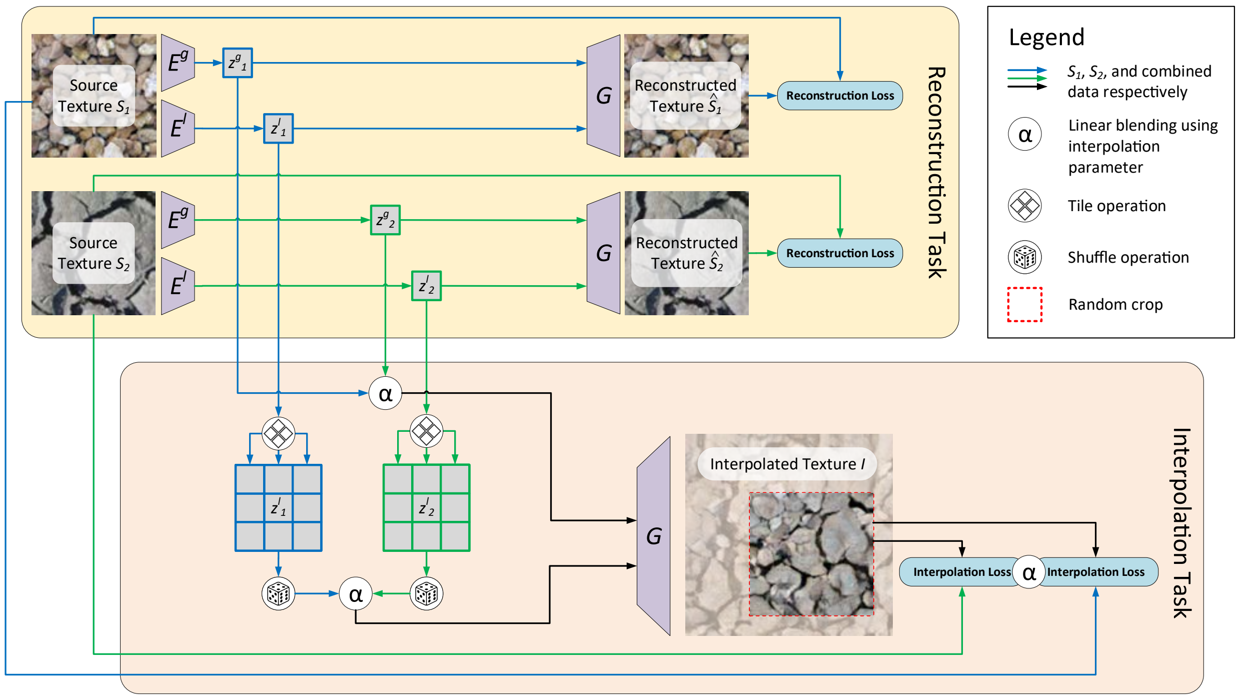

Figure 2. A diagram of our method. Background color highlights each of the tasks. Trapezoids represent trainable components that share weights if names match. Rounded rectangles represent the losses. Arrows and circles represent operations on tensor data.

Finally, some neural stylization approaches [12, 26, 17, 28] based on separating images into content and style components have shown that, by stylizing a noise content image, they can effectively synthesize texture [11]. By spatially varying the style component, texture interpolation may thus be achieved.

3. Our network: Texture Mixer

In this section, we explain how our network works. We first explain in Section 3.1 how our method is trained. We then show how our training losses are set up in Section 3.2. Finally, we explain in Section 3.3 how our method can be either tested or used by an end user.

3.1. Training setup

We aim to train our network simultaneously for two tasks: *reconstruction* and *interpolation*. The *reconstruction task* ensures that every input texture after being encoded and then decoded results in a similar texture. Meanwhile, the *interpolation task* ensures that interpolations of latent tensors are also decoded into plausible textures.

Our method can be viewed as a way of training a network containing both encoders and a generator, such that the generator is effectively a portion of a GAN. The network accepts a source texture S as input. A *global encoder* $E^g(S)$ encodes S into a latent vector z^g, which can also be viewed as a latent tensor with spatial size 1×1. A *local encoder* $E^l(S)$ encodes the source texture into a latent tensor z^l, which has a spatial size that is a factor m smaller than the size of the input texture: we use $m = 4$. The generator $G(z^l, z^g)$ concatenates z^l and z^g, and can decode these latent tensors back into a texture patch, so that ideally $G(E^l(S), E^g(S)) = S$, which encompasses the reconstruction task. Our generator is fully convolutional, so that it can generate output textures of arbitrary size: the output texture size is directly proportional to the size of the local tensor z^l. A discriminator D^{rec} is part of the reconstruction loss. An identical but separately trained discriminator D^{itp} evaluates the realism of interpolation.

Note that in practice, our generator network is implemented as taking a global tensor as input, which has the same spatial size as the local tensor. This is because, for some applications of texture interpolation, z^g can actually vary spatially. Thus, when we refer to G taking a global latent vector z^g with spatial size 1×1 as input, what we mean is that this z^g vector is first repeated spatially to match the size of z^l, and the generator is run on the result.

We show the full training setup in Figure 2. We will also explain our setup in terms of formulas here. As is shown in the upper-left of Figure 2, the network is given two real source texture images S_1 and S_2 from the real texture dataset $\mathcal{S}$. Each local encoder E^l encodes S_i ($i \in \{1, 2\}$) to a local latent tensor $z^l_i = E^l(S_i)$. Meanwhile, each global encoder E^g encodes S_i to a global latent vector z^g_i, denoted as $z^g_i = E^g(S_i)$. These latent variables are shown in green and blue boxes in the upper-left of Figure 2.

For the *reconstruction task*, we then evaluate the reconstructed texture image $\hat{S}_i = G(z_i^l, z_i^g)$. These are shown in the upper center of Figure 2. For each reconstructed image $\hat{S}_i$, we then impose a weighted sum of three losses against the original texture S_i. We describe these losses in more detail later in Section 3.2.

For the *interpolation task*, we pose the process of multiple texture interpolation as a problem of simultaneously (1) synthesizing a larger texture, and (2) interpolating between two different textures. In this manner, the network learns to perform well for both single and multiple texture synthesis. For single texture synthesis, we enlarge the generated images by a factor of 3×3. We do this by tiling z_i^l spatially by a factor of 3×3. We denote this tiling by $T(z_i^l)$, and indicate tiling by a tile icon in the lower-left of Figure 2. We chose the factor 3 because this is the smallest integer that can synthesize transitions over the four edges of z_i^l. Such a small tiling factor minimizes computational cost. The tiling operation can be beneficial for regular textures. However, in semiregular or stochastic textures, the tiling introduces two artifacts: undesired spatial repetitions, and undesired seams on borders between tiles.

We reduce these artifacts by applying a random shuffling to the tiled latent tensors $T(z_i^l)$. In Figure 2, this shuffling operation is indicated by a dice icon. Random shuffling in the latent space not only results in more varied decoded image appearance and thus reduces visual repetition, but also softens seams by spatially swapping pixels in the latent space across the border of two z_i^l tensors.

We implement the random shuffling by row and column swapping over several scales from coarse to fine. For this coarse to fine process, we use scales that are powers of two: $s_i = 2^i$ for $i = 0, 2, \ldots, n$. We set the coarsest scale n to give a scale s_n that is half the size of the local tensor z_i^l. For each scale s_i, we define a grid over the tiled latent tensor $T(z^l)$, where each grid cell has size $s_i \times s_i$. For each scale s_i, we then apply a random shuffling on cells of the grid for that scale: we denote this by P_i. This shuffling proceeds through grid rows first in top-down and then bottom-up order: each row is randomly swapped with the succeeding row with probability 0.5. Similarly, this is repeated on grid columns, with column swapping from left to right and right to left. Thus, the entire shuffling operation is:

$$P\big(T(z_i^l)\big) = P_0 \circ P_1 \circ \cdots \circ P_n\big(T(z_i^l)\big) \qquad (1)$$

We visualize this shuffling procedure in the supplementary material. We also want the synthesized texture to be able to transit smoothly between regions where there are user-specified texture constraints and regions where there are none. Thus, we override the original z_i^l without shuffling at the 4 corners of the tiled latent tensor. We denote such shuffling with corner overriding as $\tilde{P}\big(T(z_i^l)\big)$.

If we apply the fully convolutional generator G to a network trained using a single input texture and the above shuffling process, it will work for single texture synthesis. However, for multiple texture interpolation, we additionally apply interpolation in the latent space before calling G, as inspired by [27, 17, 3]. We randomly sample an interpolation parameter $\alpha \sim U[0,1]$, and then interpolate the latent tensors using α. This is shown by the circles labeled with α in Figure 2. We linearly blend the shuffled local tensors $\tilde{P}\big(T(z_1^l)\big)$ and $\tilde{P}(T(z_2^l)))$, which results in the final interpolated latent tensor Z^l:

$$Z^l = \alpha \tilde{P}\big(T(z_1^l)\big) + (1-\alpha)\tilde{P}\big(T(z_2^l)\big) \qquad (2)$$

In the same way, we blend z_1^g and z_2^g to obtain

$$Z^g = \alpha z_1^g + (1-\alpha) z_2^g \qquad (3)$$

Finally, we feed the tiled and blended tensors into the generator G to obtain an interpolated texture image $I = G(Z^l, Z^g)$, which is shown on the right in Figure 2. From the interpolated texture, we take a random crop of the same size as the input textures. The crop is shown in the red dotted lines in Figure 2. The crop is then compared using appropriately α-weighted losses to each of the source textures. We use spatially uniform weights α at training time because all the real-world examples are *spatially homogeneous* and we do not want our adversarial discriminator to detect our synthesized texture due to it having spatial variation. In contrast, at testing time, we use spatially varying weights.

3.2. Training losses

For the *reconstruction task*, we use three losses. The first loss is a pixel-wise L_1 loss against each input S_i. The second loss is a Gram matrix loss against each input S_i, based on an ImageNet-pretrained VGG-19 model. We define the Gram loss L_{Gram} in the same manner as Johnson *et al.* [20], and use the features relu$i$_1 for $i = 1, \ldots, 5$. The third loss is an adversarial loss L_{adv} based on WGAN-GP [14], where the reconstruction discriminator D^{rec} tries to classify whether the reconstructed image is from the real source texture set or generated by the network. The losses are:

$$L_{\text{pix}}^{\text{rec}} = \|\hat{S}_1 - S_1\|_1 + \|\hat{S}_2 - S_2\|_1 \qquad (4)$$

$$L_{\text{Gram}}^{\text{rec}} = L_{\text{Gram}}(\hat{S}_1, S_1) + L_{\text{Gram}}(\hat{S}_2, S_2) \qquad (5)$$

$$L_{\text{adv}}^{\text{rec}} = L_{\text{adv}}(\hat{S}_1, S_1 | D^{\text{rec}}) + L_{\text{adv}}(\hat{S}_2, S_2 | D^{\text{rec}}) \qquad (6)$$

The L_{adv} term is defined from WGAN-GP [14] as:

$$L_{\text{adv}}(A, B|D) = D(A) - D(B) + GP(A, B|D) \qquad (7)$$

Here A and B are a pair of input images, D is the adversarially trained discriminator, and $GP(\cdot)$ is the gradient penalty regularization term.

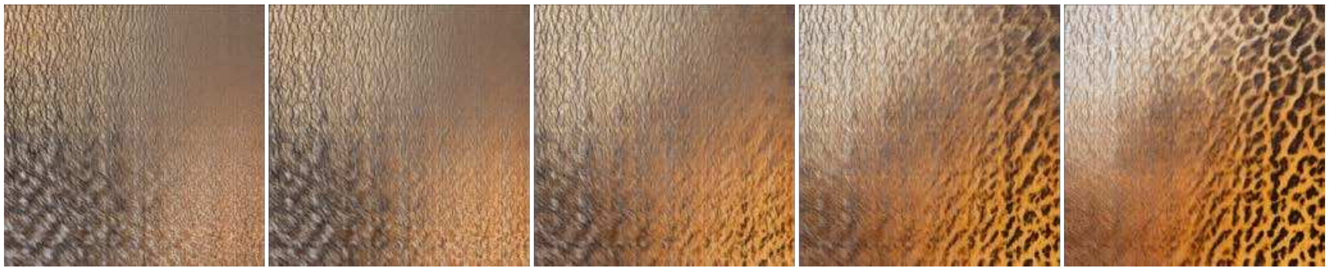
Figure 3. A sequence of dissolve video frame samples with size 1024×1024 on the *animal texture* dataset, where each frame is also with effect of interpolation.

For the *interpolation task*, we expect the large interpolated texture image to be similar to some combination of the two input textures. Specifically, if $\alpha = 1$, the interpolated image should be similar to source texture S_1, and if $\alpha = 0$, it should be similar to S_2. However, we do not require pixel-wise similarity, because that would encourage ghosting. We thus impose only a Gram matrix and an adversarial loss. We select a random crop I_{crop} from the interpolated texture image. Then the Gram matrix loss for interpolation is defined as an α-weighted loss to each source texture:

$$L_{\text{Gram}}^{\text{itp}} = \alpha L_{\text{Gram}}(I_{\text{crop}}, S_1) + (1-\alpha) L_{\text{Gram}}(I_{\text{crop}}, S_2) \quad (8)$$

Similarly, we adversarially train the interpolation discriminator D^{itp} for the interpolation task to classify whether its input image is from the real source texture set or whether it is a synthetically generated interpolation:

$$L_{\text{adv}}^{\text{itp}} = \alpha L_{\text{adv}}(I_{\text{crop}}, S_1 | D^{\text{itp}}) + (1-\alpha) L_{\text{adv}}(I_{\text{crop}}, S_2 | D^{\text{itp}}) \quad (9)$$

Our final training objective is

$$\min_{E^l, E^g, G} \max_{D^{\text{rec}}, D^{\text{itp}}} \mathop{\mathbb{E}}_{S_1, S_2 \sim \mathcal{S}} (\lambda_1 L_{\text{pix}}^{\text{rec}} + \lambda_2 L_{\text{Gram}}^{\text{rec}} + \lambda_3 L_{\text{adv}}^{\text{rec}} + \lambda_4 L_{\text{Gram}}^{\text{itp}} + \lambda_5 L_{\text{adv}}^{\text{itp}}) \quad (10)$$

where $\lambda_1 = 100$, $\lambda_2 = \lambda_4 = 0.001$, and $\lambda_3 = \lambda_5 = 1$ are used to balance the order of magnitude of each loss term, which are not sensitive to dataset.

We provide details related to our training and architecture in the supplementary document, such as how we used progressive growing during training [21].

3.3. Testing and user interactions

At testing time, we can use our network in several different ways: we can interpolate sparsely placed textures, brush with textures, dissolve between textures, and hybridize different animal regions in one image. Each of these applications utilizes spatially varying interpolation weights.

Interpolation of sparsely placed textures. This option is shown in the palette and background in Figure 1. In this scenario, one or more textures are placed down by the user in the image domain. These textures are each encoded to latent domain.

In most cases, given input textures, our method is able to achieve inherent boundary matching and continuity. However, because of the trade-off between reconstruction and interpolation losses, there might be a *slight* mismatch in some cases. To make the textures better agree at boundary conditions, we postprocess our images as follows. Suppose that the user places a source textured region as a boundary condition. We first replace the reconstructed regions with the source texture. Then, within the source texture, we use graph cuts [24] to determine an optimal seam where we can cut between the source texture and the reconstruction. Finally, we use Poisson blending [31] to minimize the visibility of this seam.

Texture brush. We can allow the user to brush with texture as follows. We assume that there is a textured background region, which we have encoded to latent space. The user can select any texture to brush with, by first encoding the brush texture and then brushing in the latent space. For example, in Figure 1 we show an example of selecting a texture from a palette created by interpolating four sparsely created textures. We find the brush texture's latent domain tensors, and apply them using a Gaussian-weighted brush. Here full weight in the brush causes the background latent tensors to be replaced entirely, and other weights cause a proportionately decreased effect. The brush can easily be placed spatially because the latent and image domains are aligned with a resizing factor m related to the architecture. We show more results in the supplementary material.

Texture dissolve. We can create a cross-dissolve effect between any two textures by encoding them both to latent domain and then blending between them using blending weights that are spatially uniform. This effect is best visualized in a video, where time controls the dissolve effect. Please see our supplementary video for such results. Figure 3 shows a sequence of video frame samples with gradually varying weights.

Animal hybridization. We generalize texture interpolation into a more practical and creative application - animal hybridization. Figure 4 shows an example. Given two aligned animal regions in one image and a hole over the

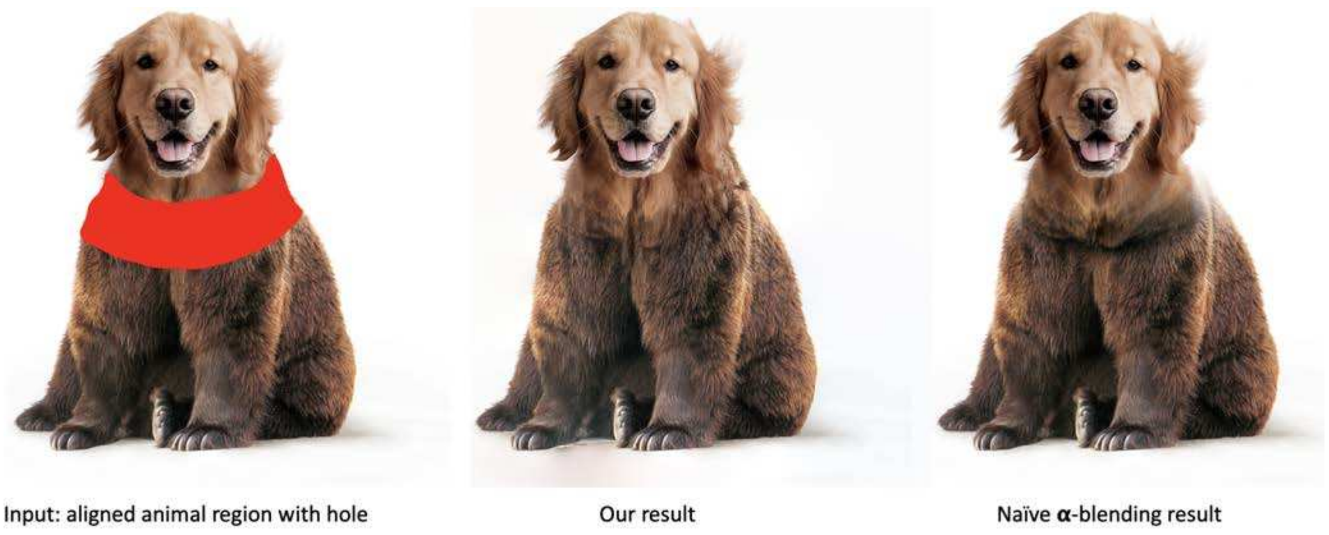

Figure 4. An animal hybridization example of size 1260×1260 between a dog and a bear. Our interpolation between the two animal furs is smoother, has less ghosting, and is more realistic than that of the Naïve α-blending.

transition region, we can sample source texture patches adjacent to the hole and conduct spatial interpolation among those textures. We fill the hole using our interpolated texture. Finally, we use graph cuts [24] and Poisson blending [31] to postprocess the boundaries. Technical details and more examples are shown in the supplemental material.

4. Experiments

In this section, we demonstrate experimental comparisons. We first introduce our own datasets in Section 4.1. We then present in Section 4.2 a suite of evaluation metrics for interpolation quality. In Section 4.3 we list and compare against several leading methods from different categories on the task of texture interpolation. In Section 4.4 we describe a user study as a holistic comparison. Finally, we conduct in Section 4.5 the ablation study by comparing against three simplified versions of our own method.

We propose to learn a model per texture category rather than a universal model because: (1) there are no real-world examples that depict interpolation between distinct texture categories; (2) there is no practical reason to interpolate across categories, e.g., fur and gravel; and (3) like with other GANs, a specific model per category performs better than a universal one due to the model's capacity limit.

4.1. Datasets

Training to interpolate frontal-parallel stationary textures of a particular category requires a dataset with a rich set of examples to represent the intra-variability of that category. Unfortunately, most existing texture datasets such as DTD [6] are intended for texture classification tasks, and do not have enough samples per category (only 120 in the case of DTD) to cover the texture appearance space with sufficient density.

Therefore, we collected two datasets of our own: (1) the *earth texture* dataset contains Creative Commons images from Flickr, which we randomly split into 896 training and 98 testing images; (2) the *animal texture* dataset contains images from Adobe Stock, randomly split into 866 training and 95 testing images. All textures are real-world RGB photos with arbitrary sizes larger than 512×512. Examples from both are shown in our figures throughout the paper.

We further augmented all our training and testing sets by applying: (1) color histogram matching with a random reference image in the same dataset; (2) random geometric transformations including horizontal and vertical mirroring, random in-plane rotation and downscaling (up to $\times 4$); and (3) randomly cropping a size of 128×128. In this way, we augmented $1,000$ samples for each training image and 100 samples for each testing image.

4.2. Evaluation

We will compare previous work with ours, and also do an ablation study on our own method. In order to fairly compare all methods, we use a horizontal interpolation task. Specifically, we randomly sampled two 128×128 squares from the test set. We call these the side textures. We placed them as constraints on either end of a 128×1024 canvas. We then used each method to produce the interpolation on the canvas, configuring each method to interpolate linearly where such option is available.

To the best of our knowledge, there is no standard method to quantitatively evaluate texture interpolation. We found existing generation evaluation techniques [34, 16, 4, 21] inadequate for our task. We, therefore, developed a suite of metrics that evaluate three aspects we consider crucial for our task: (1) user controllability, (2) interpolation smoothness, and (3) interpolation realism. We now discuss these.

User controllability. For interpolation to be considered controllable, it has to closely reproduce the user's chosen texture at the user's chosen locations. In our experiment, we measure this as the reconstruction quality for the side textures. We average the LPIPS perceptual similarity measure [43] for the two side textures. We call this *Side Perceptual Distance (SPD)*.

We also would like the center of the interpolation to be similar to both side textures. To measure this, we consider the Gram matrix loss [20] between the central 128×128 crop of the interpolation and the side textures. We report the sum of distances from the center crop to the two side textures, normalized by the Gram distance between the two. We call this measure the *Center Gram Distance (CGD)*.

Interpolation smoothness. Ideally, we would like the interpolation to follow the shortest path between the two side textures. To measure this, we construct two difference vectors of Gram matrix features between the left side texture and the center crop, and between the center crop and the right side texture, and measure the cosine distance between the two vectors. We expect this *Centre Cosine distance (CCD)* to be minimized.

For smoothness, the appearance change should be gradual, without abrupt changes such as seams and cuts. To measure such, we train a *seam classifier* using real samples from the training set as negative examples, and where we create synthetic seams by concatenating two random textures as positive examples. We run this classifier on the center crop. We call this the *Center Seam Score (CSS)*. The architecture and training details of seam classifier are the same as those of D^{rec} and D^{itp}.

Interpolation realism. The texture should also look realistic, like the training set. To measure this, we chose the Inception Score [34] and Sliced Wasserstein Distance (SWD) [21], and apply them on the center crops. This gives *Center Inception Score (CIS)* and *Center SWD*, respectively. For *CIS*, we use the state-of-the-art *Inception-ResNet-v2* inception model architecture [36] finetuned with our two datasets separately.

We also found these metrics do not capture undesired repetitions, a common texture synthesis artifact. We, therefore, trained a *repetition classifier* for this purpose. We call this the *Center Repetition Score (CRS)*. The architecture and training details of repetition classifier are almost the same as those of the seam classifier except the input image size is 128×256 instead of 128×128, where the negative examples are random crops of size 128×256 from real datasets and the positive examples are horizontally tiled twice from random crops of size 128×128 from real datasets.

4.3. Comparisons

We compare against several leading methods from different categories on the task of texture interpolation. These

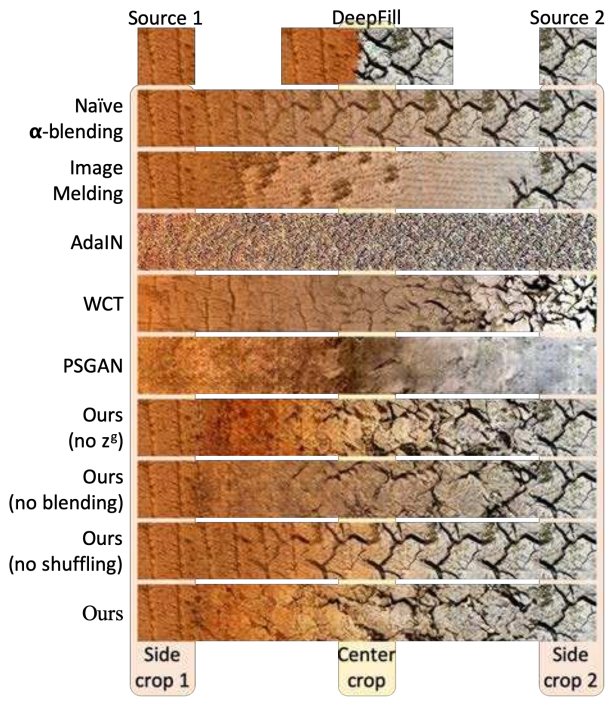

Figure 5. Qualitative demonstrations and comparisons of horizontal interpolation in the size of 128×1024 on the *earth texture* samples. We use the two side crops with the orange background for SPD measurement, and the center crop with the light yellow background for the other proposed quantitative evaluations. For the DeepFill [42] method, since the default design is not suitable for inpainting a wide hole due to lack of such ground truth, we instead test it on a shorter interpolation of size 128×384.

include: naïve α-blending, Image Melding [7] as a representative of patch-based techniques, two neural stylization methods - AdaIN [17] and WCT [28], a recent deep hole-filling method called DeepFill [42], and PSGAN [3] which is the closest to ours but without user control. Most these had to be adapted for our task. See more details in the supplementary material. Fig. 5 contains a qualitative comparison between the different methods. Note that in this example: (1) the overly sharp interpolation of DeepFill, (2) the undesired ghosting and repetition artifacts of naïve α-blending and ours (no shuffling), (3) the incorrect reconstruction and less relevant interpolation of AdaIN, WCT, and PSGAN, (4) the appearance mismatch between source and interpolation of Image Melding, (5) the lack of smoothness of ours (no z^g), and (6) the undesired fading of ours (no blending). More qualitative comparisons are shown in the supplementary material. We also report qualitative results, including the user study and the ablation experiments, in Table 1, that contains average values for the two datasets - *earth texture* and *animal texture*. Figure 6 summarizes the quantitative comparisons.

Table 1. Quantitative evaluation averaging over the *earth texture* and *animal texture* datasets. We highlighted the **best**, second best and very high values for each metric. We also indicate for each whether higher (⇑) or lower (⇓) values are more desirable.

	Controllability		Smoothness		Realism			User study		Testing time
	SPD ⇓	CGD ⇓	CCD ⇓	CSS ⇓	CRS ⇓	CIS ⇑	CSWD ⇓	PR	p-value	
Naïve α-blending	0.0000	1.255	0.777	0.9953	0.4384	22.35	60.93	0.845	$< 10^{-6}$	0.02 s
Image Melding [7]	0.0111	1.289	0.865	0.0005	0.0004	**29.45**	47.09	0.672	$< 10^{-6}$	6 min
WCT [28]	0.8605	1.321	0.988	0.0020	0.0000	9.86	46.89	0.845	$< 10^{-6}$	7.5 s
PSGAN [3]	1.1537	1.535	1.156	0.0069	0.0005	26.81	35.90	0.967	$< 10^{-6}$	1.4 min
Ours (no z^g)	0.0112	1.207	0.680	0.0078	0.0010	21.04	21.54	-	-	-
Ours (no blending)	0.0103	1.272	0.817	0.0125	0.0009	22.24	52.29	-	-	-
Ours (no shuffling)	0.0107	**1.129**	**0.490**	0.0534	0.2386	26.78	**20.99**	-	-	-
Ours	0.0113	1.177	0.623	0.0066	0.0008	26.68	22.10	-	-	0.5 s

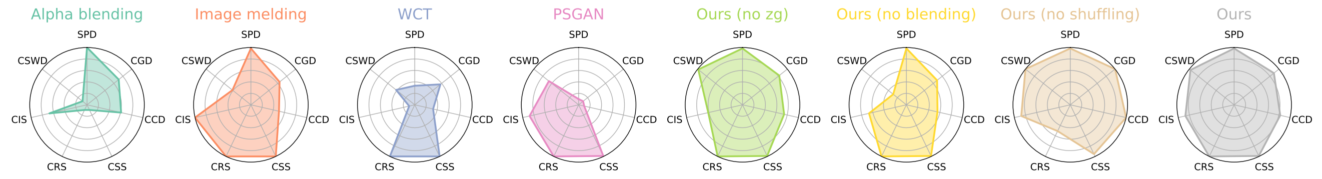

Figure 6. Radar charts visualizing Table 1. Values have been normalized to the unit range, and axes inverted so that higher value is always better. The first four are baseline methods and next three ablation candidates, with the last entry representing our full method. Our method scores near-top marks all around and shows balanced performance according to all metrics.

4.4. User study

We also conducted a user study on Amazon Mechanical Turk. We presented the users with a binary choice, asking them if they aesthetically prefer our method or one of the baseline methods on a random example from the horizontal interpolation task. The user study webpage and sanity check (to guarantee the effectiveness of users' feedback) are shown in the supplementary material. For each method pair, we sampled 90 examples and collected 5 independent user responses per example. Tallying the user votes, we get 90 results per method pair. We assumed a null hypothesis that on average, our method will be preferred by 2.5 users for a given method pair. We used a one-sample permutation t-test to measure p-values, using 10^6 permutations, and found the p-values for the null hypothesis are all $< 10^{-6}$. This indicates that the users do prefer one method over another. To quantify this preference, we count for each method pair all the examples where at least 3 users agree in their preference, and report a *preference rate (PR)* which shows how many of the preferences were in our method's favor. Both PR and the p-values are listed in Table 1.

4.5. Ablation study

We also compare against simplified versions of our method. The qualitative results for this comparison are shown in Figure 5. We report quantitative result numbers in Table 1, and visualized them in Figure 6. We ablate the following components:

Remove $\mathbf{z}^{\mathbf{g}}$. The only difference between z^g and z^l is in the tiling and shuffling for z^l. However, if we remove z^g, we find texture transitions are less smooth and gradual.

Remove texture blending during training. We modify our method so that the interpolation task during training is performed only upon two identical textures. This makes the interpolation discriminator D^{itp} not be aware of the realism of blended samples, so testing realism deteriorates.

Remove random shuffling. We skip the shuffling operation in latent space and only perform blending during training. This slightly improves realism and interpolation directness, but causes visually disturbing repetitions.

5. Conclusion

We presented a novel method for controllable interpolation of textures. We were able to satisfy the criteria of controllability, smoothness, and realism. Our method outperforms several baselines on our newly collected datasets. As we see in Figure 6, although some baseline method may achieve better results than ours on one of the evaluation criteria, they usually fail on the others. In contrast, our method has consistent high marks in all evaluation categories. The user study also shows the users overwhelmingly prefer our method to any of the baselines. We have also demonstrated several applications based on this technique and hope it may become a building block of more complex workflows.

Acknowledgement

The authors acknowledge the Maryland Advanced Research Computing Center for providing computing resources and acknowledge the photographers for licensing photos under Creative Commons or public domain.

References

[1] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017. 2

[2] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics (ToG)*, 28(3):24, 2009. 1, 2

[3] U. Bergmann, N. Jetchev, and R. Vollgraf. Learning texture manifolds with the periodic spatial GAN. In *Proceedings of the 34th International Conference on Machine Learning*, pages 469–477, 2017. 2, 4, 7, 8

[4] M. Bińkowski, D. J. Sutherland, M. Arbel, and A. Gretton. Demystifying mmd gans. *arXiv preprint arXiv:1801.01401*, 2018. 6

[5] D. Chen, L. Yuan, J. Liao, N. Yu, and G. Hua. Stylebank: An explicit representation for neural image style transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1897–1906, 2017. 2

[6] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, , and A. Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014. 6

[7] S. Darabi, E. Shechtman, C. Barnes, D. B. Goldman, and P. Sen. Image melding: Combining inconsistent images using patch-based synthesis. *ACM Trans. Graph.*, 31(4):82–1, 2012. 1, 2, 7, 8

[8] O. Diamanti, C. Barnes, S. Paris, E. Shechtman, and O. Sorkine-Hornung. Synthesis of complex image appearance from limited exemplars. *ACM Transactions on Graphics (TOG)*, 34(2):22, 2015. 1, 2

[9] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346. ACM, 2001. 1, 2

[10] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *iccv*, page 1033. IEEE, 1999. 1, 2

[11] L. Gatys, A. S. Ecker, and M. Bethge. Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 262–270, 2015. 1, 2, 3

[12] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2414–2423, 2016. 2, 3

[13] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 2

[14] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5767–5777, 2017. 2, 4

[15] D. J. Heeger and J. R. Bergen. Pyramid-based texture analysis/synthesis. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 229–238. ACM, 1995. 1, 2

[16] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, pages 6626–6637, 2017. 6

[17] X. Huang and S. J. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *ICCV*, pages 1510–1519, 2017. 2, 3, 4, 7

[18] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 2

[19] N. Jetchev, U. Bergmann, and R. Vollgraf. Texture synthesis with spatial generative adversarial networks. *arXiv preprint arXiv:1611.08207*, 2016. 2

[20] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pages 694–711. Springer, 2016. 1, 2, 4, 7

[21] T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017. 2, 5, 6, 7

[22] A. Kaspar, B. Neubert, D. Lischinski, M. Pauly, and J. Kopf. Self tuning texture optimization. In *Computer Graphics Forum*, volume 34, pages 349–359. Wiley Online Library, 2015. 2

[23] V. Kwatra, I. Essa, A. Bobick, and N. Kwatra. Texture optimization for example-based synthesis. In *ACM Transactions on Graphics (ToG)*, volume 24, pages 795–802. ACM, 2005. 2

[24] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: image and video synthesis using graph cuts. *ACM Transactions on Graphics (ToG)*, 22(3):277–286, 2003. 2, 5, 6

[25] S. Lefebvre and H. Hoppe. Appearance-space texture synthesis. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 541–548. ACM, 2006. 2

[26] C. Li and M. Wand. Precomputed real-time texture synthesis with markovian generative adversarial networks. In *European Conference on Computer Vision*, pages 702–716. Springer, 2016. 1, 2, 3

[27] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang. Diversified texture synthesis with feed-forward networks. In *Proc. CVPR*, 2017. 1, 2, 4

[28] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang. Universal style transfer via feature transforms. In *Advances in Neural Information Processing Systems*, pages 386–396, 2017. 2, 3, 7, 8

[29] G. Liu, F. A. Reda, K. J. Shih, T.-C. Wang, A. Tao, and B. Catanzaro. Image inpainting for irregular holes using partial convolutions. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018. 2

[30] W. Matusik, M. Zwicker, and F. Durand. Texture design using a simplicial complex of morphable textures. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 787–794. ACM, 2005. 2

[31] P. Pérez, M. Gangnet, and A. Blake. Poisson image editing. *ACM Transactions on graphics (TOG)*, 22(3):313–318, 2003. 5, 6

[32] J. Portilla and E. P. Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International journal of computer vision*, 40(1):49–70, 2000. 1, 2

[33] E. Risser, P. Wilmot, and C. Barnes. Stable and controllable neural texture synthesis and style transfer using histogram losses. *arXiv preprint arXiv:1701.08893*, 2017. 2

[34] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016. 2, 6, 7

[35] O. Sendik and D. Cohen-Or. Deep correlations for texture synthesis. *ACM Transactions on Graphics (TOG)*, 36(5):161, 2017. 2

[36] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017. 7

[37] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. S. Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*, pages 1349–1357, 2016. 1, 2

[38] L.-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 479–488. ACM Press/Addison-Wesley Publishing Co., 2000. 2

[39] Y. Wexler, E. Shechtman, and M. Irani. Space-time completion of video. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (3):463–476, 2007. 2

[40] C. Yang, X. Lu, Z. Lin, E. Shechtman, O. Wang, and H. Li. High-resolution image inpainting using multi-scale neural patch synthesis. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, page 3, 2017. 2

[41] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang. Free-form image inpainting with gated convolution. *arXiv preprint arXiv:1806.03589*, 2018. 2

[42] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang. Generative image inpainting with contextual attention. *arXiv preprint arXiv:1801.07892*, 2018. 2, 7

[43] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 7

[44] Y. Zhou, Z. Zhu, X. Bai, D. Lischinski, D. Cohen-Or, and H. Huang. Non-stationary texture synthesis by adversarial expansion. *ACM Trans. Graph.*, 37(4):49:1–49:13, 2018. 2

[45] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017. 2

[46] J.-Y. Zhu, R. Zhang, D. Pathak, T. Darrell, A. A. Efros, O. Wang, and E. Shechtman. Toward multimodal image-to-image translation. In *Advances in Neural Information Processing Systems*, 2017. 2