TRITON: Neural Neural Textures for Better Sim2Real

Anonymous Author(s)
Affiliation
Address
email

Abstract: Unpaired image translation algorithms can be used for sim2real tasks, but most fail to generate temporally consistent results. We present a new approach that combines differentiable rendering with image translation to achieve temporal consistency over indefinite timescales, using surface consistency losses and *neural neural textures*. We call this algorithm TRITON (Texture Recovering Image Translation Network): an unsupervised, end-to-end, stateless sim2real algorithm that leverages the underlying 3d geometry of input scenes by generating realistic-looking learnable neural textures. By settling on a particular texture for the objects in a scene, we ensure consistency between frames statelessly. Unlike previous algorithms, TRITON is not limited to camera movements — it can handle the movement of objects as well, making it useful for downstream tasks such as robotic manipulation. We demonstrate the superiority of our approach both qualitatively and quantitatively, using robotic experiments and comparisons to ground truth photographs. We show that TRITON generates more useful images than other algorithms do.

Keywords: sim2real, image translation, differentiable rendering

1 Introduction

Current sim2real image translation algorithms used for robotics [1, 2] often have difficulty generating consistent results across large time-frames particularly when the objects in an environment are allowed to move. This makes training good robotic policies using such sim2real challenging. In this paper, we discuss an algorithm called TRITON (Texture Recovering Image Translation Network) that combines neural rendering, image translation, and two special surface consistency losses to create surface-consistent translations. We use the term surface-consistent to refer to the desirable quality of preserving the visual appearance of object surfaces as they move, or are viewed from another angles.

TRITON is applicable when we have a simulator capable of generating a realistic distribution of geometric data, but when we do not have any information about the surfaces of those objects (which we need to render realistic images). For example, in our experiments we use a robotic arm model provided by the manufacturer that contains no material information, and then learn to render the materials of the arm using TRITON. As opposed to requiring an artist to create 3d models with meaningful textures, TRITON can generate these from scratch using a set of photographs capturing the distribution of states in a simulation, without requiring any matches between domains.

We conduct multiple experiments to confirm TRITON's advantage over prior image translation approaches commonly used for sim2real including CycleGAN [3] and CUT [4]. Important, we show the advantages of the proposed approach in real-robot sim2real experiments, learning the textures and training a robot policy solely based on the images generated by TRITON.

Submitted to the 6th Conference on Robot Learning (CoRL 2022). Do not distribute.

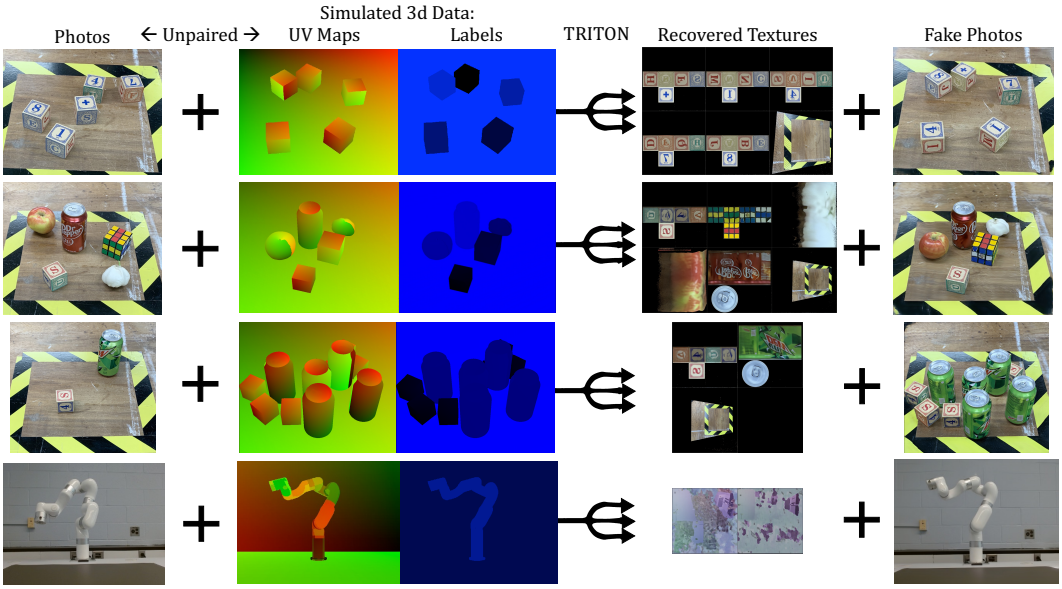

Figure 1: This is an overview of what TRITON accomplishes. TRITON learns the textures of 3d objects to help translate simulated images into photographs. It does this with unpaired data. Each row is a different dataset.

2 Related Works

Unsupervised Image to Image and Video to Video Translation Generative models have been very successful in creating images unconditionally [5, 6], creating images conditioned on text [7, 8] and creating images when trained on paired data [9]. Generative models that translate images between unpaired data is a challenging task as well, but has been done before [10, 3, 11, 12]. Of particular interest are sim2real image to image translation algorithms such as [1, 2, 13, 14]. These algorithms aim to translate simulated images into realistic ones, for various purposes such as data augmentation or video game enhancement. In particular, Ho et al. [1], Rao et al. [2] are used to help train robots. Our paper uses an architecture based on Pfeiffer et al. [13], which is based on MUNIT [11]. This area of research has been extended to video to video translation, where optical flow is often used to create a stateful translation where temporal coherence is preserved [15, 16]. These algorithms fail to provide temporal coherence over long time periods, however. Another work from Rivoir et al. [17], is closely related to ours and uses neural rendering to translate videos with temporal coherence even when the camera spins around all the way.

View Synthesis View synthesis is a field of research where we predict the outputs of unseen camera viewpoints. NeRF [18] has spawned a large body of research on end to end view synthesis, resulting in works such as [19, 20, 21]. Related to our work, which also uses geometry based backbone are [22, 23, 24]. All of these works, however, only work with static scenes. There are NeRF-based approaches that work on dynamic scenes [25, 26, 27] but none of these interactive. Menapace et al. [28] is interactive, but its conditioned on learned actions and not compatible with physics simulators that would be used to train robots.

Neural Textures Thies et al. [29] introduced the concept of neural textures and image translators in the context of a deferred rendering pipeline, used in other works such as [17] which is closely related to our project. Rivoir et al. [17] differs from TRITON in a few very important aspects though, the biggest being that it only synthesizes new camera views, where the only difference between each scene is the placement of the camera. In contrast, our algorithm takes in scenes where several objects are placed in different places or deformed, so that one static global 3d model of the environment is not sufficient to accomplish our tasks. In addition, TRITON uses a new type of neural texture called a "neural neural texture", which is represented continuously using a fourier feature network, as described in [30].

3 Formulation

TRITON's goal is to turn 3d simulated images into realistic fake photographs (by training it without any matching pairs), while maintaining high surface consistency. It does this by simultaneously learning both an image translator and a set of realistic textures. These translations can be useful for downstream tasks, especially robotic policy learning from camera data, enabling sim2real.

In contrast to previous works involving "neural textures" [29, 17], TRITON can handle more than just camera movements: the positions of objects in the training data can be moved around or deformed as well between data samples. With high surface consistency, surfaces of translated objects will look the same even when moved around or viewed from different camera angles.

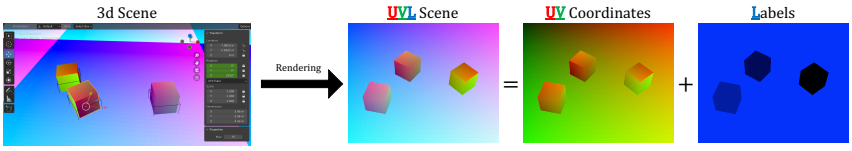

Figure 2: A scene is obtained by rendering a 3d state into an image, where the first two color channels represent u, v coordinates and the third channel l represents object labels.

There are two components in every dataset: A set of simulated 3d scenes, and a set of photographs. Datasets usually have many more scenes than photographs, since scenes can be generated automatically. A photograph is an image tensor, having (r, g, b) values between 0 and 1, is denoted as $p \in [0, 1]^{H \times W \times 3}$ where 3 refers to the three (r, g, b) channels, and H, W are the height and width respectively.

In this work, a scene refers to a rendered 3d simulation state - which includes the position, rotation, and deformation of every object (including the camera). We represent each 3d state with a simple (r, g, b) image, in a simple enough format that any decent simulator should able to provide. A scene has the same dimensions as a photograph and is denoted as $s \in [0, 1]^{H \times W \times 3}$. However, unlike p, s's three (r, g, b) channels encode a special semantic meaning: (u, v, l) as depicted in Figure 2. The channels u and v refer to a texture coordinate, whereas l refers to a label value that identifies every object in a scene. In a given dataset, each object gets a different label. Likewise, each object is assigned a different texture. We assume that every dataset has L different label values, L different objects, and that we must learn L different neural textures.

4 Method

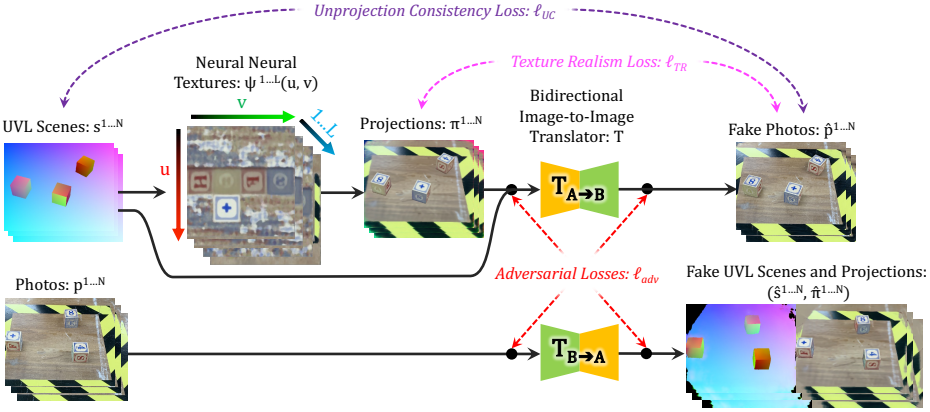

Figure 3: This figure is an overview TRITON. Inputs are on the left, and outputs are on the right. The dashed arrows indicate losses. Although $\pi^{1...N}$ and $\hat{p}^{1...N}$ look similar, they are not identical - ℓ_{TR} encourages them to look as similar as possible.

In this section, we describe how TRITON works to translate images from domain A (simulation) to domain B (photos) while maintaining surface consistency. TRITON introduces a learneable neural

neural texture with two novel surface consistency losses to an existing image translator. In the end, TRITON is able to effectively generate photorealistic images.

4.1 Neural Neural Texture Projection

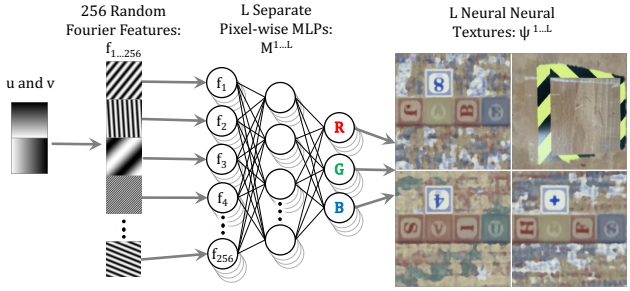

Figure 4: This figure details how our neural neural textures are calculated in Figure 3. They are fully differentiable, and represented continuously along the texture's u and v axes using Fourier feature networks.

Instead of feeding a scene s directly into the image translator $T_{A\to B}$, we first apply learnable textures ψ to its surfaces as shown in Figure 3. Previous works called these learnable textures "neural textures", and were parametrized by a discrete grid of differentiable texels. In contrast, we call our learnable textures as "neural nerual textures", because our textures themselves are represented as a neural network function, parameterized continuously over UV space. Our neural textures $\psi^{1\dots L}$ are learned jointly with our image translator T.

These neural neural textures are represented implicitly by a neural network that maps UV values to RGB values. For each texture $\psi^i \in \psi^{1\dots L}$,

$$\psi^i : (u, v) \in \mathbb{R}^2 \to (r, g, b) \in \mathbb{R}^3 \tag{1}$$

Given a scene s, we obtain projection π by applying a texture $\psi^i \in \psi^{1\dots L}$ to every pixel $(u, v, l) \in s$ individually, where texture index $i = I(l)$ is decided by the label value l of that scene pixel.

$$\pi_{[x,y]} = \psi^i(s_{[x,y,1]}, s_{[x,y,2]}) \tag{2}$$

where $x \in \{1 \dots W\},\ y \in \{1 \dots H\}$. Texture index $i = I(l)$ where $l = s_{[x,y,3]}$ and the function $I(\cdot)$ scales and discretizes l into integers. Subscripts mean multidimensional indexing.

We now discuss the implementation of our neural neural networks $\psi^{1\dots L}$. Each neural neural texture ψ^i is a function consisting of two components: a multi-layer perceptron M^i and a static set of 256 random spatial frequencies $f_{1\dots256} \in \mathbb{R}^{256\times2}$. Given a two-dimensional (u, v) vector, the spatial frequencies are used to generate a high dimensional embedding of (u, v) as input for M^i, where M^i is a function $\mathbb{R}^{256} \to \mathbb{R}^3$ mapping that embedding to an (r, g, b) color value:

$$\psi^i(u, v) = M^i(\sin(g_1),\ \cos(g_1),\ \sin(g_2),\ \cos(g_2),\ \dots\ \sin(g_{256}),\ \cos(g_{256})\ ) \tag{3}$$

where $g_k = f_k \cdot (u, v)$, $f_k \in \mathcal{N}(\mu = 0,\ \sigma = 10)$ is a two dimensional random gaussian vector, and $\cdot$ is the dot product operator. The hyperparamters 256 and 10 are selected empirically.

In practice, we found that TRITON learns faster and more stably with our neural neural textures than it does with raster neural textures. The generated textures are also smoother and less noisy. See the supplementary material for more details.

4.2 Image Translation

Our image translation module T is bidirectional: $T_{A\to B}$ translates sim to real (aka domain A to domain B), and $T_{B\to A}$ translates real to sim (aka domain B to domain A).

$$\hat{p} = T_{A\to B}(\pi, s) \quad \text{and} \quad (\hat{\pi}, \hat{s}) = T_{B\to A}(p) \tag{4}$$

T is based on the image translation module used in Rivoir et al. [17], which is a based on a variant [13] of MUNIT [11]. T uses the same network architectures as MUNIT, and also inherits its adversarial losses ℓ_{adv}, cycle consistency losses ℓ_{cyc} and reconstruction losses ℓ_{rec}. The main differences

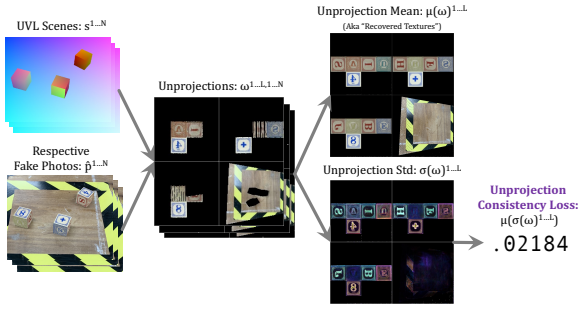

Figure 5: This is the unprojection consistency loss used in Figure 3. The Unprojection Mean is not used in any losses, but help to illustrate the content in Section 4.4.

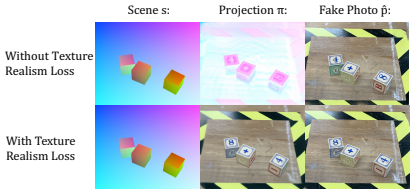

Figure 6: The neural texture looks more realistic with texture realism loss enabled. Note that the blocks show different letters because of different initializations and the symmetry of a cube - both solutions are valid texture assignments.

between our image translation model and MUNIT are that we use a different image similarity loss Ω, and like Pfeiffer et al. [13] and Rivoir et al. [17] our style code is fixed (making it unimodal) and noise is injected into the latent codes during training to prevent overfitting. During inference however, this intermediate noise is removed and our image translation module is deterministic.

MUNIT translates images by encoding both domains A and B into a common latent space, then decoding them into their respective domains B and A. In our translation module, we define fake photos $\hat{p} = T_{A\to B}(\pi, s) = G_B(E_A(\pi, s))$ and fake projection/uvl scenes $(\hat{\pi}, \hat{s}) = T_{B\to A}(p) = G_A(E_B(p))$ where E_A, E_B, G_A, G_B are encoders and generators for domains A and B respectively.

We define an image similarity loss Ω between two images x, y that returns a score between 0 and 1, where 0 means perfect similarity:

$$\Omega(x, y) = L_2(x, y) - \text{msssim}(x, y) \tag{5}$$

This function combines mean pixel-wise L_2 distance (used in MUNIT) with multi-scale structural image similarity msssim, introduced in [31]. Note that this loss can also be applied to the latent representations obtained by E_A and E_B, because like images those tensors are also three dimensional.

We have cycle consistency loss $\ell_{cyc} = \Omega\left((\pi, s), T_{B\to A}\left(\hat{p}\right)\right) + \Omega\left(p, T_{A\to B}\left(\hat{\pi}, \hat{s}\right)\right)$, similarity loss $\ell_{rec} = \Omega\left((\pi, s), G_A\left(E_A(\pi, s)\right)\right) + \Omega\left(p, G_B\left(E_B\left(p\right)\right)\right)$ (which is effectively an autoencoder loss), and content similarity loss $\ell_{con} = \Omega\left(E_A\left(\pi, s\right), E_B\left(\hat{p}\right)\right) + \Omega\left(E_A\left(p\right), E_A\left(\hat{\pi}, \hat{s}\right)\right)$. We also have adversarial losses ℓ_{adv} that come from two discriminators D_A and D_B, targeting domains A and B respectively, using the LS-GAN loss introduced in Mao et al. [32]. In total, our image translator loss is $\ell_T = \ell_{cyc} + \ell_{rec} + \ell_{con} + \ell_{adv}$.

4.3 Unprojection Consistency Loss

To keep the object surfaces consistent, we impose a pixel-wise "unprojection consistency loss" by unprojecting surfaces in fake photos back into the common texture space. Given a UVL scene s and its respective fake photo $\hat{p}$, the unprojection ω is obtained from assigning (r, g, b) values at each pixel location (x, y) of $\hat{p}$ to its corresponding (u, v, l) coordinates according to s. For simplicity, we note

$$\omega^l_{[u,v]} = \mathbb{E}\,\hat{p}_{[x,y]} \quad s.t.\ (u, v, l) = s_{[x,y]}, \tag{6}$$

where the expectation $\mathbb{E}$ means we aggregate multiple (r, g, b) vectors being assigned to the same (u, v, l) coordinate by averaging them. In practice, (u, v, l) are real numbers between $[0, 1]$, where as each ω have to be rasterized, i.e., represented by L images with a size of $(D \times D \times 3)$, where L is the number of labels and $D \times D$ is the resolution of the unprojection. Therefore, we discretize and scale corresponding (u, v, l) to integers so that each (r, g, b) vector will be assigned to a pixel on ω. For the exact implementation, please see our supplementary material.

We obtain N unprojections $\{\omega^{l,i}_{[u,v]}\}_{i=1}^N$ from a batch of N UVL scenes and fake photos. The unprojection consistency loss is defined as the per pixel-channel standard deviation of ω over the batch

$$\ell_{UC} = \frac{1}{L \times D \times D \times 3} \sum_{l,u,v,c} \sigma(\{\omega^{l,i}_{[u,v,c]}\}_{i=1}^N), \tag{7}$$

where $\sigma(\cdot)$ stands for the standard deviation function and $c \in \{1 \dots 3\}$ is the channel index of ω. We minimize ℓ_{UC} to encourage unprojections to be consistent across the batch. Intuitively, if ℓ_{UC} were 0, it would mean the object surfaces in translations $\hat{p}$ appear exactly the same in every scene. In addition, we call the mean of unprojections $\mu(\{\omega^{l,i}\}_{i=1}^{N})$ as "Recovered Textures", visualzed in Figure 1 and Section 4.2.

4.4 Texture Realism Loss

To encourage the neural textures to look as realistic as possible, we try to make the projections look like the final output by introducing a "texture realism" loss ℓ_{TR}. ℓ_{TR} is an image similarity loss that makes π look like its translation $\hat{p}$.

$$\ell_{TR} = \Omega(\pi, \hat{p}) \tag{8}$$

Without ℓ_{TR}, ψ can look wildly different each time you train it. As seen in the top row of Figure 6, the neural texture looks very unrealistic — the colors are completely arbitrary. By adding texture realism loss ℓ_{TR}, we make the textures ψ more realistic. In practice, this makes TRITON less likely to mismatch the identity of translated objects.

5 Experiments

5.1 Datasets

We constructed two datasets AlphabetCube-3 and RobotPose-xArm to benchmark different image translators in many perspectives. In our setting, each dataset is composed of two sets of unpaired images: UVL scenes from a simulator and real photographs. Note that the images in all datasets are unpaired and UVL scenes only rely on rough 3d models of the objects in the scene without need of precisely aligning objects in the simulator to the real world. We use AlphabetCube-3 for image translation quality evaluation, and RobotPose-xArm for sim2real policy learning evaluation.

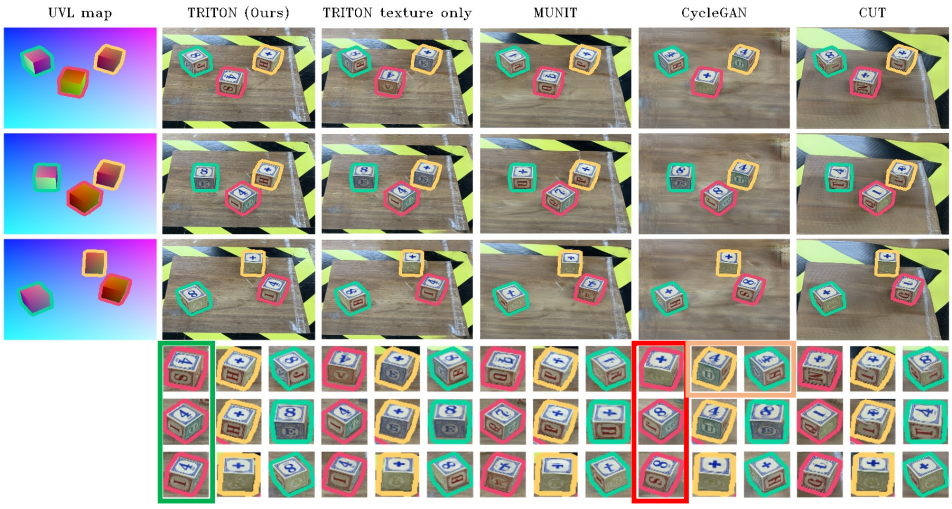

Figure 7: Comparison of image translation methods. The first column is the UVL map as the input. The other columns are results using different image translation methods. Each row of images shows a different random placement of objects in the scene. TRITON consistently outputs high quality results over different object arrangements. "TRITON texture only" stands for TRITON without two surface consistency losses ℓ_{UC} and ℓ_{TR}.

5.2 Image Translation Evaluation on AlphabetCube-3

AlphabetCube-3 dataset features a table with three different alphabet blocks on the top. The dataset contains 300 real photos and 2000 UVL scenes from a simulator. Each photograph features these three cubes with a random position and rotation. In this section, we evaluate the image translation accuracy of the TRITON and other image translation methods using AlphabetCube-3.

Figure 7 shows qualitative results on the dataset. From Figure 7 we find that TRITON consistently outputs high quality results over different object arrangements. The textures keep aligned even when the position of blocks change dramatically over multiple scenes (See examples in green rectangle in Figure 7). In contrast, though MUNIT [11] and CycleGAN [3] manage to generate realistic images, the surfaces of the cubes are either consistent over scenes (See exsamples in the red rectangle) or replicated by mistake (See examples in the orange rectangle). Also note that the floor background of the outputs from CycleGAN are quite different from the ground truth. CUT [4] fails to generate meaningful images regarding both the foreground blocks and the background.

We further conduct quantitative experiments and the results are shown as Table 1. In this evaluation we manually align the simulator with 14 photos of different real world scenes and generate the UVL maps for translation. We measured the LPIPS [33] and $l2$-norm between the translated images and the real images using multiple configurations. 'Masked' means we mask out the background (which is the floor in AlphabetCube-3 dataset) and only measure the translation quality w.r.t three foreground blocks. For the 'unmasked' tests, we compare the whole image without any masking instead, and the background contributes much more to the losses due to its larger area. Similar to the qualitative results, TRITON consistently outperforms other methods under different metrics and configurations. Further ablations TRITON w/o $\psi, \ell_{UC}, \ell_{TR}$ and TRITON w/o ℓ_{UC}, ℓ_{TR} show the importance of the new losses we introduce.

Table 1: Quantitative results on AlphabetCube-3. LPIPS [33] and $l2$-norm between the translated images and the real images are reported. We exclude backgrounds in the 'Masked' configuration. TRITON consistently outperforms other methods in different metrics and configurations

	Masked		Unmasked	
	LPIPS ($\times 10^{-1}$) ↓	L2 ($\times 10^{-2}$) ↓	LPIPS ($\times 10^{-1}$) ↓	L2 ($\times 10^{-2}$) ↓
CycleGAN	0.450	0.559	6.02	4.25
CUT	0.469	0.553	5.40	6.37
TRITON w/o $\psi, \ell_{UC}, \ell_{TR}$	0.437	0.500	4.34	4.25
TRITON w/o ℓ_{UC}, ℓ_{TR}	0.442	0.586	2.85	3.64
TRITON	**0.286**	**0.479**	**1.17**	**1.23**

5.3 Sim2real Transfer for Robot Learning

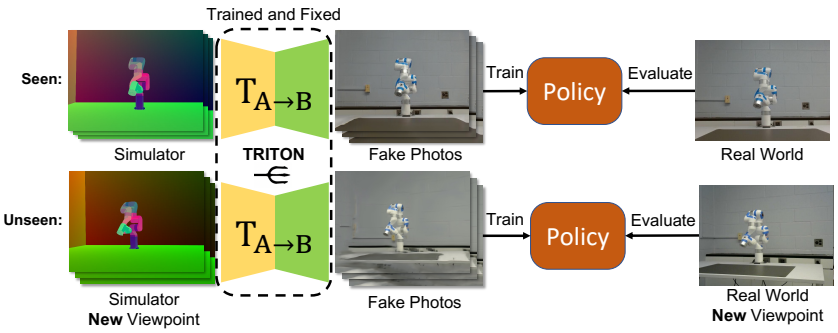

Figure 8: Sim2real framework of using TRITON. We evaluate in two settings: Seen and Unseen. In Unseen, we use a new camera viewpoint to train and evaluate the policy. This new viewpoint is not used for training TRITON.

In this section we demonstrate how TRITON improves sim2real transfer for robot policy learning. To this end, we first train TRITON on a dataset consist of unpaired UVL scenes from a robot simulator (Gazebo) and photos from the real robot. Once TRITON is trained, we learn the robot policy while only utilizing the simulator. The input of the policy is the fake photo $\hat{p}$ translated from the simulated UVL scene using TRITON's translator $T_{A \to B}$. Finally, the trained policy is directly evaluated on the real robot, taking real photos p as input.

Ideally, a good image translation model makes $\hat{p}$ similar to p, so that the policy trained on synthesized photos will transfer to real domain seamlessly and will have better performance. All our policies are learned with zero real-world robot interaction with the environment.

Task We use a robot pose imitation task for our experiments. Given the input of a photo of real robot pose, the policy outputs robot controls to replicate that target pose. We measure the angular error between the replicated and target poses as our evaluation metric: $\sqrt{\sum_j (\hat{a}_j - a_j)^2}$, where $\hat{a}_j$ and a_j are replicated and target joint angles respectively, and j is the joint index. We use an xArm robot which has 7 joints. We also attached patterns and tapes on the robot to make its texture more challenging to model.

Robot policy and Baselines Since the sim2real formulation allows benefiting from abundant training data using the simulator, we use behavior cloning to train a good robot policy. The policy is implemented with a CNN [34]. We compare TRITON against two baseline image translation methods: CycleGAN [3] and CUT [4], by replacing TRITON with each baseline method respectively in the above pipeline. The robot policy learning method is same for all the methods.

Evaluation Settings We introduce two main evaluation settings, **Seen** and **Unseen**. In the **Seen** setting, the policy is trained and tested using the same camera viewpoint that is used during TRITON (or a baseline translator) training. Whereas in **Unseen** setting, we introduce a camera at a new viewpoint for training and testing the policy. That is, the image translator has to generate the fake photos out of its training distribution (seen camera viewpoints), which becomes a challenging task for the translator. We also vary input image resolutions to show the power of photo-realistic images in higher resolutions.

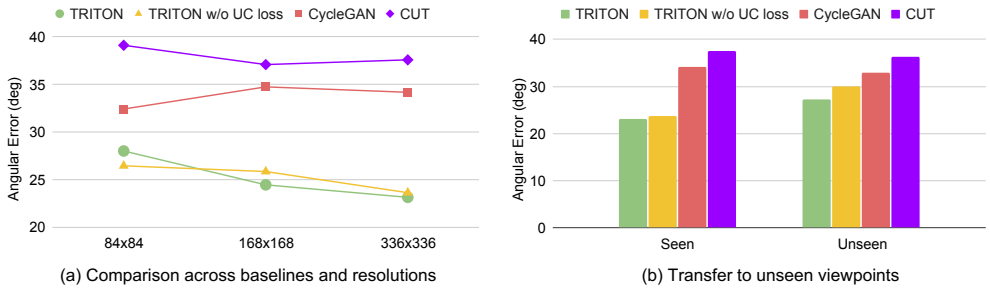

Figure 9: Evaluation results of sim2real robot learning. We compare TRITON against two baselines and one TRITON variant w/o ℓ_{UC} across 3 different input image resolutions. TRITON outperforms CycleGAN [3] and CUT [4] consistently across (a) all resolutions and (b) unseen viewpoints. Results in (a) are from Seen setting only. Results in (b) are from 336x336 resolution.

Results Figure 9 shows the evaluation results of the sim2real transfer. Both Seen and Unseen evaluations show that TRITON outperforms baselines consistently. In Figure 9(a), we find that TRITON continuously improves from low to high resolutions due to its ability to generate more photo-realistic images compared to the baselines. From Figure 9(b), we confirm that TRITON outperforms the others also in the Unseen setting, showing that TRITON learns image translations that are generalizable to new viewpoints while others are limited. In the comparison against TRITON w/o ℓ_{UC}, TRITON outperforms this variant consistently except the lowest resolution. This again shows the effectiveness of ℓ_{UC} to generate photo-realistic images, and such high quality images are important in higher resolutions for sim2real transfer.

6 Limitaions

TRITON relies on 3D models of objects in order to generate photo-realistic images, making it less applicable when the geometry of an environment is unknown. Acquiring such 3D models could be challenging in the wild, especially in robots allowed to roam the world. TRITON is not able to handle transparent objects, because the UVL format is opaque. Moreover, the probability of incorrectly matching textures to objects increases as the number of object classes increases.

References

[1] D. Ho, K. Rao, Z. Xu, E. Jang, M. Khansari, and Y. Bai. RetinaGAN: An Object-aware Approach to Sim-to-Real Transfer. *arXiv*, 2020.

[2] K. Rao, C. Harris, A. Irpan, S. Levine, J. Ibarz, and M. Khansari. RL-CycleGAN: Reinforcement Learning Aware Simulation-To-Real. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 00:11154–11163, 2020. doi:10.1109/cvpr42600.2020.01117. RL-CycleGAN.

[3] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. *arXiv*, 2017.

[4] T. Park, A. A. Efros, R. Zhang, and J. Zhu. Contrastive learning for unpaired image-to-image translation. *CoRR*, abs/2007.15651, 2020. URL https://arxiv.org/abs/2007.15651.

[5] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks, 2014. URL https://arxiv.org/abs/1406.2661.

[6] T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks, 2018. URL https://arxiv.org/abs/1812.04948.

[7] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. Zero-shot text-to-image generation, 2021. URL https://arxiv.org/abs/2102.12092.

[8] C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. Denton, S. K. S. Ghasemipour, B. K. Ayan, S. S. Mahdavi, R. G. Lopes, T. Salimans, J. Ho, D. J. Fleet, and M. Norouzi. Photorealistic text-to-image diffusion models with deep language understanding, 2022. URL https://arxiv.org/abs/2205.11487.

[9] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks, 2016. URL https://arxiv.org/abs/1611.07004.

[10] X. Su, J. Song, C. Meng, and S. Ermon. Dual Diffusion Implicit Bridges for Image-to-Image Translation. *arXiv*, 2022.

[11] X. Huang, M.-Y. Liu, S. Belongie, and J. Kautz. Multimodal Unsupervised Image-to-Image Translation. *arXiv*, 2018.

[12] M.-Y. Liu, T. Breuel, and J. Kautz. Unsupervised Image-to-Image Translation Networks. *arXiv*, 2017. UNIT.

[13] M. Pfeiffer, I. Funke, M. R. Robu, S. Bodenstedt, L. Strenger, S. Engelhardt, T. Roß, M. J. Clarkson, K. Gurusamy, B. R. Davidson, L. Maier-Hein, C. Riediger, T. Welsch, J. Weitz, and S. Speidel. Generating large labeled data sets for laparoscopic image processing tasks using unpaired image-to-image translation. *arXiv*, 2019.

[14] S. R. Richter, H. A. AlHaija, and V. Koltun. Enhancing photorealism enhancement. *arXiv:2105.04619*, 2021.

[15] M. Chu, Y. Xie, J. Mayer, L. Leal-Taixé, and N. Thuerey. Learning temporal coherence via self-supervision for GAN-based video generation. *ACM Transactions on Graphics (TOG)*, 39 (4):75:1–75:13, 2020. ISSN 0730-0301. doi:10.1145/3386569.3392457.

[16] L. Amsaleg, B. Huet, M. Larson, G. Gravier, H. Hung, C.-W. Ngo, W. T. Ooi, Y. Chen, Y. Pan, T. Yao, X. Tian, and T. Mei. Mocycle-GAN. *Proceedings of the 27th ACM International Conference on Multimedia*, pages 647–655, 2019. doi:10.1145/3343031.3350937.

[17] D. Rivoir, M. Pfeiffer, R. Docea, F. Kolbinger, C. Riediger, J. Weitz, and S. Speidel. Long-Term Temporally Consistent Unpaired Video Translation from Simulated Surgical 3D Data. *arXiv*, 2021.

[18] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[19] P. P. Srinivasan, B. Deng, X. Zhang, M. Tancik, B. Mildenhall, and J. T. Barron. NeRV: Neural Reflectance and Visibility Fields for Relighting and View Synthesis. *arXiv*, 2020.

[20] S. J. Garbin, M. Kowalski, M. Johnson, J. Shotton, and J. Valentin. FastNeRF: High-Fidelity Neural Rendering at 200FPS. *arXiv*, 2021.

[21] C. Lin, W. Ma, A. Torralba, and S. Lucey. BARF: bundle-adjusting neural radiance fields. *CoRR*, abs/2104.06405, 2021. URL https://arxiv.org/abs/2104.06405.

[22] G. Riegler and V. Koltun. Free View Synthesis. *arXiv*, 2020.

[23] G. Riegler and V. Koltun. Stable View Synthesis. *arXiv*, 2020.

[24] J. Y. Zhang, G. Yang, S. Tulsiani, and D. Ramanan. NeRS: Neural Reflectance Surfaces for Sparse-view 3D Reconstruction in the Wild. *arXiv*, 2021.

[25] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer. D-NeRF: Neural Radiance Fields for Dynamic Scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[26] K. Park, U. Sinha, J. T. Barron, S. Bouaziz, D. B. Goldman, S. M. Seitz, and R. Martin-Brualla. Nerfies: Deformable neural radiance fields. *ICCV*, 2021.

[27] Y. Du, Y. Zhang, H.-X. Yu, J. B. Tenenbaum, and J. Wu. Neural radiance flow for 4d view synthesis and video processing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[28] W. Menapace, S. Lathuilière, A. Siarohin, C. Theobalt, S. Tulyakov, V. Golyanik, and E. Ricci. Playable Environments: Video Manipulation in Space and Time. *arXiv*, 2022.

[29] J. Thies, M. Zollhöfer, and M. Nießner. Deferred Neural Rendering: Image Synthesis using Neural Textures. *arXiv*, 2019.

[30] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng. Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains. *arXiv*, 2020.

[31] J. Snell, K. Ridgeway, R. Liao, B. D. Roads, M. C. Mozer, and R. S. Zemel. Learning to Generate Images with Perceptual Similarity Metrics. *arXiv*, 2015.

[32] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. P. Smolley. Least Squares Generative Adversarial Networks. *arXiv*, 2016.

[33] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. *arXiv*, 2018.

[34] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning, 2013. arXiv:1312.5602.

A Appendix

A.1 Neural Neural Textures: More Details

In Section 4.1, we mentioned that neural neural textures learn better than raster neural textures. TRITON can be ablated to use raster textures. Let's call this version "Raster TRITON". Raster TRITON is extremely sensitive to the resolution of its neural texture, and can be numerically unstable if that resolution is too high. In comparison, regular TRITON's neural neural textures do not have a specific resolution: they are continuously defined over the UV domain using Fourier feature networks.

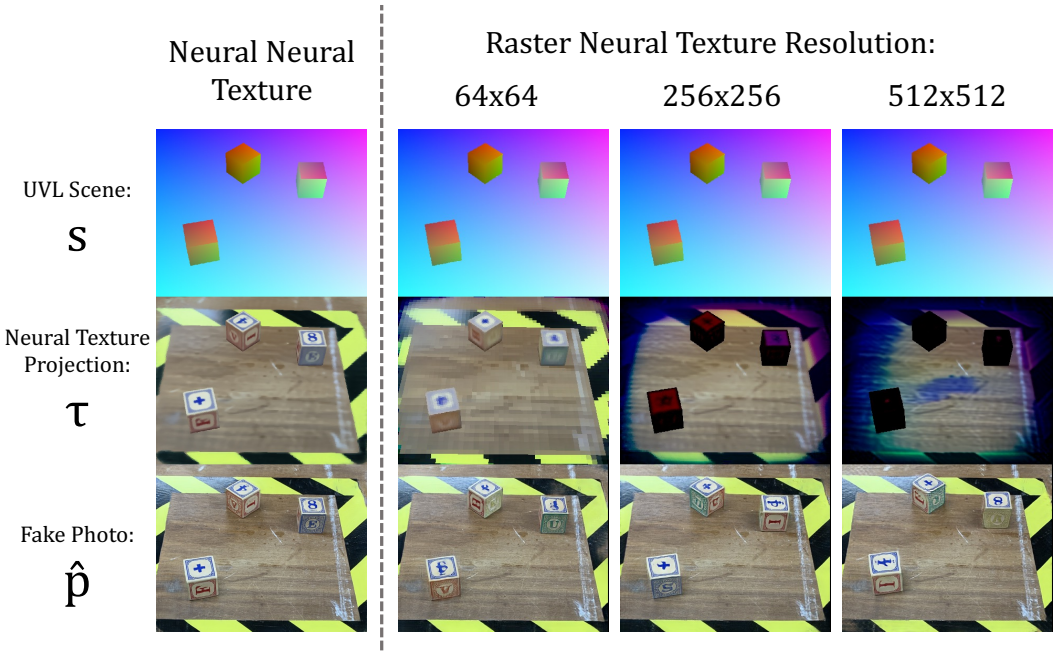

Figure 10: The resolution of the raster neural texture seriously impacts performance in "Raster TRITON". Each column has a different neural texture resolution. The neural texture only looks like the translation when the resolution is low.

In this paragraph we offer possible explanations for these observations. With raster-based textures, only the side of an object that currently is seen in a view is allowed to change during each update, because the gradient can not be propagated into pixels which are not seen. If this means changing overall brightness of an object for example, the brightness of that object can not be changed over the entire object until every side has been seen - which will not happen in a single iteration, since the batch size is limited.

In addition to not propagating the gradient to unseen sides of objects, it also can not propagate to texels in-between the UVL values seen in a scene image. When the raster neural texture is too large, aliasing effects occur: if you have a raster texture with very high resolution, the loss gradient is less likely to be passed to a given texel because the chance that a given UV value in a scene will be rounded to that pixel's coordinates is very small. In practice, this makes large raster neural textures unstable and limits us to using small amounts of detail. With neural neural textures, this aliasing problem is mostly mitigated, because when a certain texture receives a gradient at particular UV coordinates, the areas of the texture between those coordinates are also changed.

A.2 Unprojection Consistency Loss: More Details

Here, we give more details about unprojection consistency loss ℓ_{UC}.

The exact equation for ℓ_{UC} is as follows: to formally define ℓ_{UC} we define unprojections $\omega^{1...N,1...L}$ and mean unprojections (aka recovered textures) $\bar{\omega}^{1...L}$ with each $\omega^{n,i} \in \mathbb{R}^{3\times128\times128}$:

$$\omega^{n,i}_{c,U,V} = \mathbb{E}\,\hat{p}^{n}_{c,x,y} \quad \text{and} \quad \bar{\omega}^{i}_{c,u,v} = \frac{1}{N}\sum_{n=1}^{N}\omega^{n,i}_{c,u,v} \quad \text{and}$$

$$\ell_{UC} = \frac{\sum_{i=1}^{L}\sum_{c=1}^{3}\sum_{U=1}^{128}\sum_{V=1}^{128}\sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(\bar{\omega}^{i}_{c,u,v} - \omega^{i,n}_{c,U,V}\right)^2}}{3\cdot128\cdot128\cdot L} \tag{9}$$

where $U = \lfloor 128u \rfloor$ and $V = \lfloor 128v \rfloor$ and $i = I(l)$ with $u = s^{n}_{1,x,y}$, $v = s^{n}_{2,x,y}$, $l = s^{n}_{3,x,y}$ for all $n \in \{1...N\}$, $i \in \{1...L\}$, $x \in \{1...W\}$, $y \in \{1...H\}$, $U \in \{1...128\}$, $V \in \{1...128\}$.

The hyperparameter 128 we described in Section 4.3 refers to the resolution of the unprojections used to calculate ℓ_{UC}. In Figure 12 we see that if we were to set it higher, the gradient wouldn't affect the neural texture as densely.

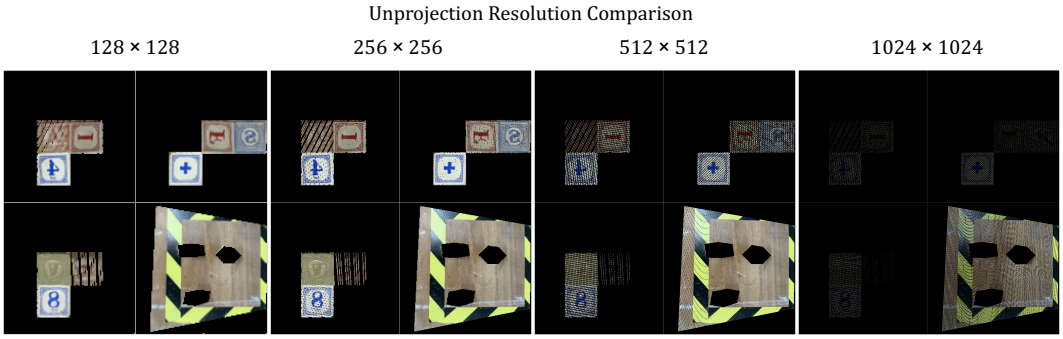

Figure 11: Here, we unproject a single fake photo $\hat{p}^{i}$. The resolution of the unprojection matters for unprojection consistency loss. The larger it is, the more precise the alignment will be but the less likely a given UV value is to be assigned a loss greater than 0, creating numerical instability and slowing down learning. When the unprojection resolution is too high, only a few areas of the neural texture will receive a gradient during backpropogation. In practice we use the resolution 128×128.

A.3 Training Procedures

In this section we detail the training procedures used to create results in Section 5 from Triton, CycleGAN, CUT, TRITON w/o ℓ_{UC}, ℓ_{TR} (aka TRITON with neither unprojection consistency loss nor texture reality loss), and TRITON w/o $\psi, \ell_{UC}, \ell_{TR}$ (aka TRITON without textures, and therefore also without unprojection consistency loss or texture reality loss).

A.3.1 TRITON

We train TRITON for 200,000 iterations on all datasets. This takes about 36-48 hours on an NVIDIA RTX A5000. The dimension of each input scene is defined by hyperparameter height H; the width is scaled to match the aspect ratio of the original input. To avoid running out of video memory, we randomly crop each input to a 256×256 subset and run TRITON on that cropped input. We use batch size 5 during training.

We found that the best way to train TRITON is to start with smaller H values and progressively increase that value during training. For the first 50,000 iterations we use $H = 320$, then for the next 50,000 iterations $H = 420$, then for the last 100,000 iterations $H =$ (the height of the original input scene).

Like in [17], we use three Adam optimizers: two for the translation module's encoders and decoders, and another for the neural texture. For the translation module's optimizers, we use learning rate

1×10^{-4}, and for the neural neural texture we use learning rate 1×10^{-3}. Every 100,000 iterations all learning rates are halved.

During evaluation, we render the neural neural texture onto a raster grid of pixels. Because the neural neural texture has a 2d manifold, it can be closely approximated by an RGB image. Using this method lets us avoid evlauating ψ's fourier feature network on every frame by using a raster image as a lookup table. This decreases video memory usage and speeds up calulations during inference.

A.3.2 TRITON w/o ℓ_{UC}, ℓ_{TR}

This ablation is also known as "Texture-Only" TRITON, because it has a learnable texture ψ but no consistency losses. Its trained the same way that we train TRITON normally as discussed above in A.3.1, except the surface consistency losses ℓ_{UC} and ℓ_{TR} are omitted. Effectively, it performs better than MUNIT-like TRITON (aka TRITON w/o $\psi, \ell_{UC}, \ell_{TR}$, discussed in A.3.3)

A.3.3 TRITON w/o $\psi, \ell_{UC}, \ell_{TR}$

This ablation is very similar to MUNIT, as TRITON is based on MUNIT and this ablation has neither learnable texture nor consistency losses. Like in subsection A.3.2, this ablation shares the same training procedures as TRITON.

A.3.4 CycleGAN

We use the original implementation of CycleGAN, and stick to the reccomended 200 epochs, as it tends to overfit if you go further. For our tests, we run CycleGAN on multiple different scales of the input scenes, with input sizes 286×286 (CycleGAN's default), 320×320, 420×420 and 512×512. After translation, we stretch the image into the input's aspect ratio. We also set $\lambda_{identity} = 0$, meaning we disable the identity loss. The reported results are the calculated using the best results among these resolutions. This dramatically improves CycleGAN's performance when translating UVL scenes to photographs, as this loss was built with the assumption that some parts of the input should not be changed during translation (which is not true in our case).

A.3.5 CUT

We use the original implementation of CUT, and stick to the recommended 400 epochs, as it tends to overfit if you go further. Like with CycleGAN in A.3.4, we run CUT on multiple different resolutions of the input scenes, with input sizes 286×286 (CUT's default), 320×320, 420×420 and 512×512. After translation, we stretch the image into the input's aspect ratio. The reported results are the calculated using the best results among these resolutions.

A.4 More Results

A.4.1 Unprojection Comparisons

In this section we expand on the results in Subsection 5.1 by showing the mean unprojection (aka recovered texture) for different image translation algorithms. With the 14 UVL images and corresponding fake photos (or photos) as calculated in 5.1, we average their unprojections. If a translation algorithm is consistent, these unprojections should align well and the average $bar\omega$ should not be blurry.

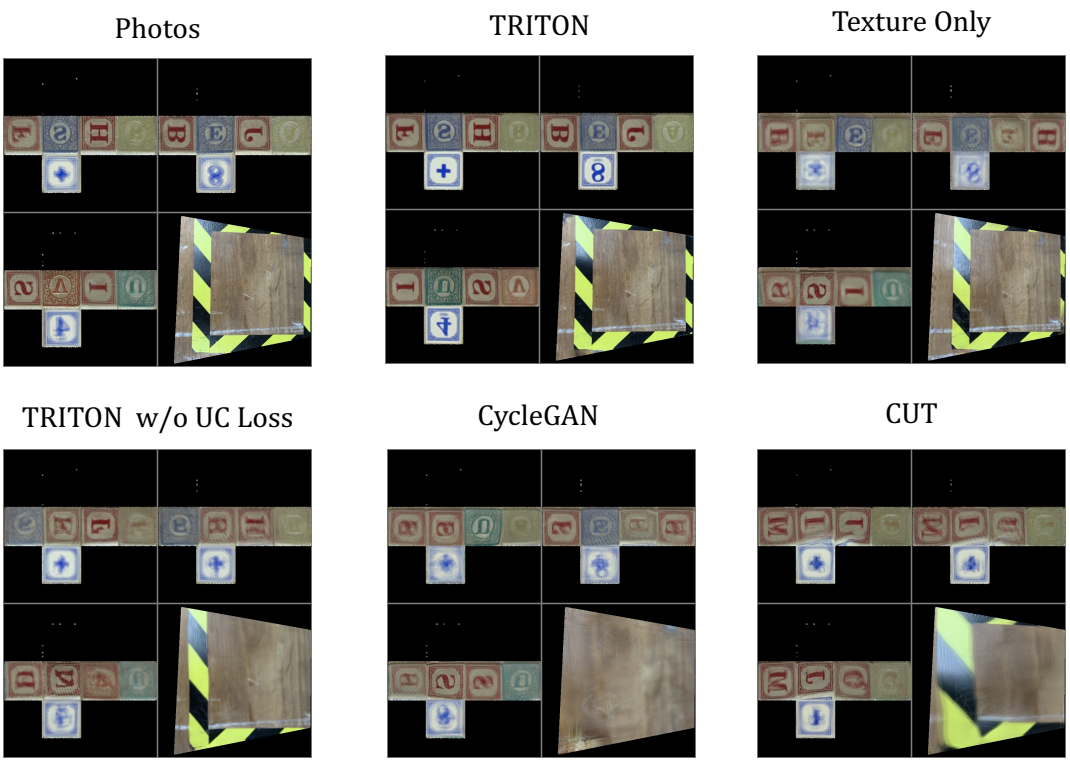

Figure 12: In this figure, we compare the recovered textures $\bar{\omega}$ (aka mean unprojections) of the 14 labeled images between different algorithms, and compare it to a ground truth. In this diagram, we align all the unprojections to make them easier to compare. Note how TRITON is more crisp and matches the unprojected photographs better than the other algorithms, which are blurrier and have the wrong colors and letters on each side of the alphabet blocks. CycleGAN and CUT for example incorrectly duplicate letters between alphabet blocks.

A.4.2 Videos

In this section we have links to animations that help demonstrate various aspects of TRITON. If you are viewing this document on a computer, click a thumbnail to view the video.

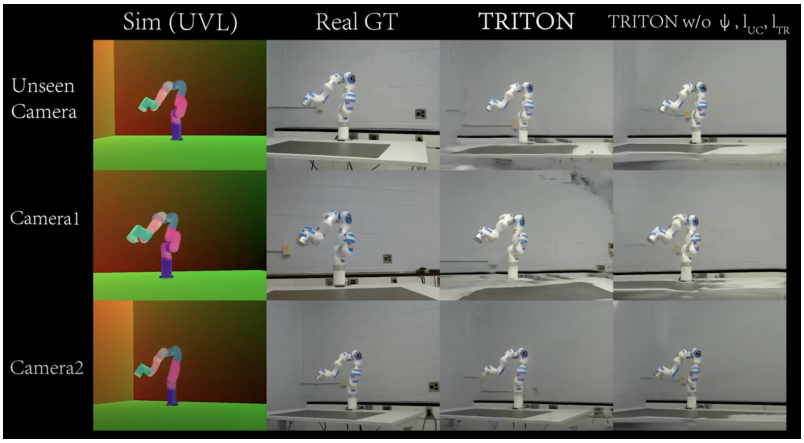

Figure 13: This animation shows a side-by-side comparison of different camera viewpoints of the results discussed in Section 5.3, as well as the inputs, ground truth video, TRITON and TRITON ablation videos. Url: youtu.be/kecK_cJgLT8

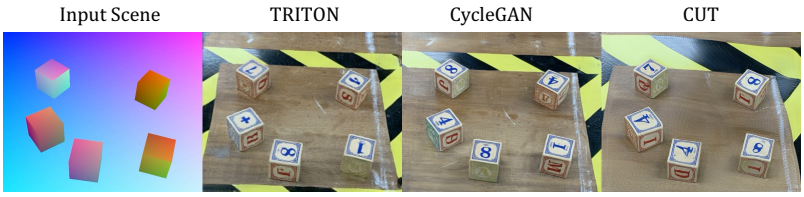

Figure 14: This animation shows an animation of various algorithms on a dataset with 5 alphabet blocks moving around. When watching, note how the numbers on the top of the cubes in both CycleGAN and CUT change over time, while with TRITON they stay the same. Url: youtu.be/0t0xiVS_8D0

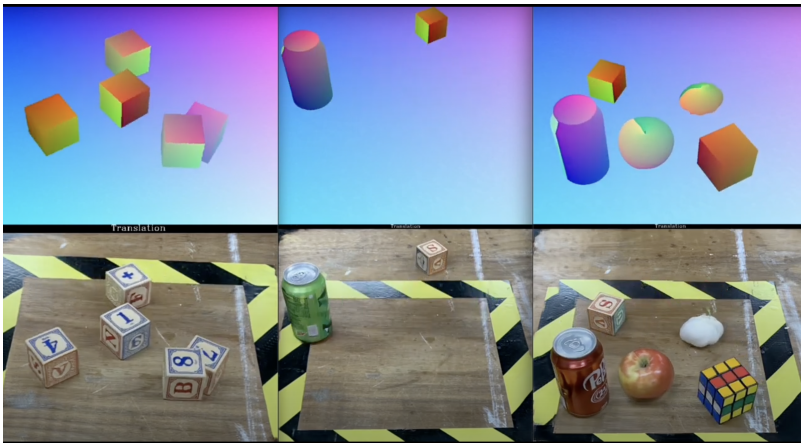

Figure 15: This video shows the results of TRITON applied on three of the datasets shown in Figure 1. The top row shows the input scenes s, and the bottom row shows the fake photographs $\hat{p}$. Url: youtu.be/0t0xiVS_8D0