CS230

RenderGAN: GAN Based Texture Rendering

Joon Jung
joonjung@stanford.edu
Project Code: https://github.com/jjbits/RenderGAN

Abstract

In this project, we've attempted to replace 3D texture rendering with generative texture transferring. We have trained Pix2Pix GAN on various 3D plane models with different hyperparameters to tune the networks with different performance characteristics. Our experiment results show merely increasing the number of models or the number of captured images of them do not give out optimal results.

1 Introduction

Modern graphics rendering requires intensive computations, performing high resolution texture sampling and shading in million times fold. As the result, the memory related operations have become the most performance constraining components in the graphics pipeline.

In recent, various researches on extracting and fusing content representations and style representations from different image domains, using deep learning, have indeed significantly progressed. [GEB16; Iso+17; Zhu+17] The content and style extraction and fusion is also called as 'texture transferring'.

What if we can extract, from a high resolution texture, the style representation and fuse it with a polygon only 3D model acting as the content representation source. Then we can think of the 3D rendering process as the texture transferring process. If we can achieve this fusion in a seamless manner, this becomes equivalent to the texture rendering process of a 3D rendering pipeline. Therefore, it would be possible to completely replace the pipeline with a deep learning model.

This project starts from this motive. We are replacing the texture rendering process in a 3D pipeline with a generative texture transferring using the generator model from Pix2Pix[Iso+17]. The figure 1 shows the normal rendering pipeline and the proposed rendering pipeline with a deep learning generator side by side.

The input to our generative model is a 3D rendered image with no texturing, rendered only with a single color. The target image is another 3D rendered image with texturing. Then our generative DL model generates an output image trying to match the target image. One example pair of the input and target images are shown in the figure 2.

2 Related Work

Image Style Transfer Using Convolutional Neural Networks Work of [GEB16] shows how to fuse the structure of one image domain with the style of another image domain using Variational Auto Encoders. By extracting the intermediate results from different CNN layers, one can extract different features of the given image. More deeper the layer goes down, one can obtain more higher level features of the image. One such a high level feature is the texture of an image, which we are trying to extract and fuse with the content of the same image.

CS230: Deep Learning, Winter 2019, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

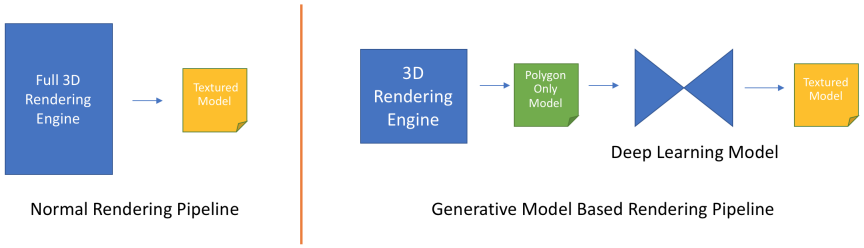

Figure 1: Pipelines

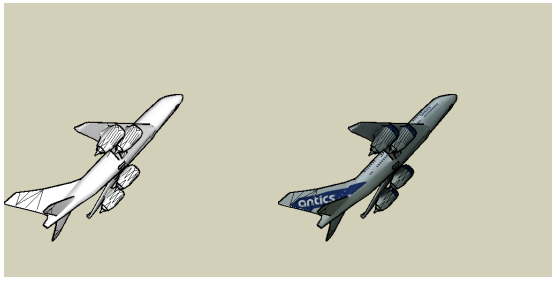

Figure 2: Left: an example input image. Right: an example target image.

Texture Transferring There have been various researches on how to transfer textures bypassing the traditional 3D rendering pipeline. StyleBlit[Sýk+19] attempts to optimize style transferring by using local guidance such as normal values or texture coordinates. This method requires the implementer to fine tune the texture related parameters by hand. TextureGAN[Xia+17] performs texture transferring using Generative Adversarial Networks[Goo+14], employing sophisticated various multiple loss objectives to control the target texture transferring.

Image-to-Image Translation with Conditional Adversarial Networks Our work here is primarily based on Pix2Pix[Iso+17]. Using conditional GAN, Pix2Pix achieves high performance style transferring with relatively low computational cost. In addition to the regular adversatrial loss $\mathcal{L}_{adv}$, it uses $L1$ pixel distance loss to guide the generator to produce accurate results.

3 Dataset and Features

Throughout the project, we collected multiple datasets with varying plane categories and varying number of models to train the networks. More details and the training results can be referenced from the section 5. Initially, we collected about 700 plane models of all kinds with 14 random view image captures of the model in each from Shapenet[Cha+15]. Then, in order to collect higher quality models, we switched our model database to SketchUP's 3D Wherehouse[Ske]. We selected 49 passenger airplane models from it and generated around 17000 random view 512x512 RGB image captures in total.

For the image collecting and pre-processing, we implemented custom python OBJ loader, random perspective image capturing tools using SketchUp's Ruby API and many python utillity scripts[Cod]. Following what was done in Pix2Pix work[Iso+17], the images were added with random jitters to assist the training.

One non-conventional aspect of this project is that we really do not need to worry about overfitting since the goal is to faithfully mimic the target textured image in the training dataset. We want as much as of overfitting actually. For this reason, we did not have any dev/test sets put aside but just randomly picked 3D models out of the training set to see how our DL renderer performed.

4 Methods

Pix2Pix[Iso+17], which we have referenced as our base framework for this project, employs several distinguishing features for its objective function and its implementation. It uses $L1$ pixel distancing to improve its faithful regeneration of the target image pixels. It uses conditional GAN(cGAN) to improve the generated image's sharpness. In order to promote the generator's layers to share the relevant features, it uses skip-connections for its U-Net generator.

4.1 Objectives

The overall objective function of Pix2Pix is

$$G^* = \arg\max_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G)$$

for the generator G and the discriminator D. The conditional GAN loss is

$$\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{x,y}[\log D(x, y)] + \mathbb{E}_{x,z}[\log D(x, G(x, z))].$$

This conditional term differs from the usual GAN's in such that the discriminator uses the input image x in addition to using the generated output image $G(y)$ and y as shown in the figure 3.

The L1 loss is

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[||y - G(x, z)||_1].$$

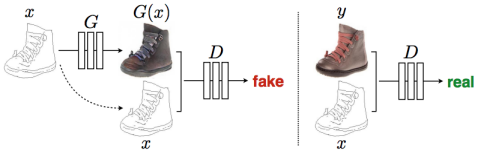

Figure 3: Role of an input image x for cGAN, excerpted from [Iso+17].

4.2 Network Architectures

Both the generator and discriminator use modules consisted of Convolution-BatchNorm-ReLu. The generator is consisted of encoder and decoder pair.
Encoder:

C64−C128−C256−C512−C512−C512−C512−C512 .

U-Net decoder:

CD512−CD1024−CD1024−C1024−C1024−C512−C256−C128 .

The U-Net decoder numbering indicates the skip connections between each layer i in the encoder and each layer $n - i$ in the decoder, except for the last encoder layer and the first decoder layer.

For the discriminator, we have used 70 x 70 PatchGan discriminator.
Discriminator:

C64−C128−C256−C512 .

4.3 Optimization

One gradient step is equally applied to train D and G, but the objective for D is divided by 2 in order to slow down the learning relative to G. Stochastic Gradient descent with Adam optimizer is used with learning rate of 0.0002 and momentum parameters of $\beta_1 = 0.5$ and $\beta_2 = 0.999$. All the experiments were ran for 200 epochs

5 Experiments/Results/Discussion

5.1 Collecting The Right Dataset

During our model training, we realized the most important hyperparameter of our project was on selecting the right dataset. For example, the number of plane model categories greatly influenced the generator's ability to draw the correct plane shape. As the result, for the major part of this work, we've spend significant amount of time coming up with a right dataset to train the generator to faithfully mimic the target images. Also since our goal here was to come up with a generator in capable of drawing similar images to the target images in the training set, we did not worry about overfitting the generator and about the generalization error too much.

5.2 Number Of Plane Categories Chosen

As the first attempt, we collected several hundreds of different plane models in all kinds and randomly captured their images with different camera positions and angles. The figure 4 shows two results of the trained generator. As shown in the middle column of the figure, the generator struggled generating the correct shape of the planes with this dataset. This led us to narrow down the plane categories and we decided our train model to be only in the passenger airplane category.

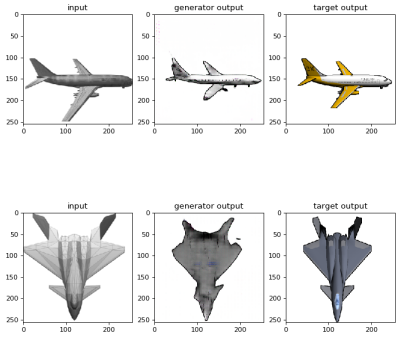

Figure 4: Multiple plane categories dataset

5.3 Training On A Single Model

As the next step, we picked one passenger plane model and captured 1000 and 5000 random images to train. The figure 5 shows the result for 5000 captures. We can see there is no more shape corruption problem. The model has also produced fairly descent shading. We can see the generator even has mimicked the letters on the plane wing fairly well, seen from a distance.

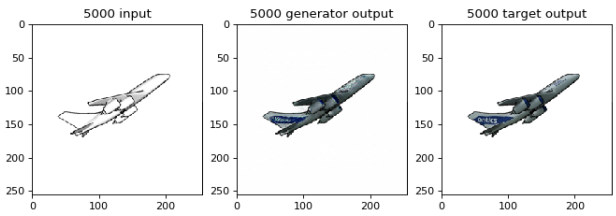

Figure 5: 1000 vs 5000 capturens on a single model

5.4 Training On Multiple Passenger Models

We then increased the number of models used and varied the number of captured images to see how the results varied. The figure 6 shows the results. The first part number of the title indicates the number of the models used and the trailing number indicates the number of the images captured in total. For example, 49-14700 indicates the training dataset with 49 different passenger planes with 14700 image captures in total. Comparing to the single model dataset, we can see these datasets did not really produce better results.

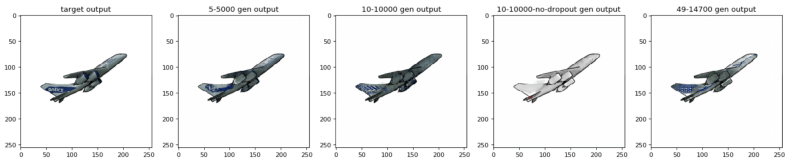

Figure 6: Multiple models

5.5 Accuracy Measure

Our primary accuracy measure is by human eys inspection, which unfortunately can be subjective. In order to supplement it, we've used Mean Squared Errors between the target image and the generated image as well. The following table shows the various MSE for the different datasets.

dataset	1-5000	5-5000	10-10000	49-14700
MSE	6.377	6.968	7.571	7.493

The single model 5000 dataset achieves the lowest MSE as well as the best result with human eye inspection. However, there are other examples which are generated from the generators trained with more heterogeneous datasets producing some impressive results. One such example is shown in the figure 7.

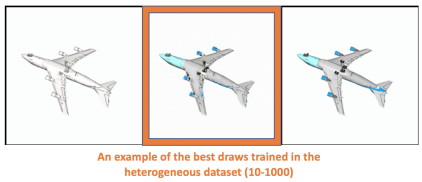

Figure 7: 10-10000 dataset best example

5.6 Overfitting With No Dropout

Since overfitting was allowed in this project scenario, we were curious to see what would be the effect of not using dropout for training the generator. As shown in the figure 6, 4th column, not using dropout produced a bad result with underfitting. Based on this result, we can say dropout doesn't just help with overfitting, but also help with underfitting as well.

6 Conclusion/Future Work

Since our usage scenario is to reuse the training data later in the deployed environment, we do not have to worry about overfitting. With this fact, and since Pix2Pix model is good at producing a fairly good result with less data, our simple dataset(single model, 5000 captured images) has produced the best result. However, a lot of our 3D models have common texturing features, such as having some colors on the tail wings or the head of the planes. If we could control the generator to focus on these features for example, we could have produced good results with more generalized networks and could have been able to see more creative results.

References

[Goo+14] Ian Goodfellow et al. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 2672–2680. URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

[Cha+15] Angel X. Chang et al. *ShapeNet: An Information-Rich 3D Model Repository*. Tech. rep. arXiv:1512.03012 [cs.GR]. Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[GEB16] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. “Image Style Transfer Using Convolutional Neural Networks”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[Iso+17] Phillip Isola et al. “Image-to-Image Translation with Conditional Adversarial Networks”. In: *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. 2017.

[Xia+17] Wenqi Xian et al. “Texturegan: Controlling deep image synthesis with texture patches”. In: *arXiv preprint arXiv:1706.02823* (2017).

[Zhu+17] Jun-Yan Zhu et al. “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networkss”. In: *Computer Vision (ICCV), 2017 IEEE International Conference on*. 2017.

[Sýk+19] Daniel Sýkora et al. “StyleBlit: Fast Example-Based Stylization with Local Guidance”. In: *Computer Graphics Forum* 38.2 (2019), pp. 83–91.

[Cod] Project Code. URL: https://github.com/jjbits/RenderGAN.

[Ske] SketchUp. URL: https://3dwarehouse.sketchup.com/search/?q=airplane.