Neural Texture Enhancement

Mutian Tong, Carl von Bonin
Columbia University in the City of New York

Abstract
We propose a novel approach to procedural texture generation for use in video game scenes utilizing a modified Pix2PixHD model to artificially enhance first-person-view renders of low-fidelity scenes and extract the synthetic textures. To increase performance, we take advantage of the additional information available through the explicit scene representation to pass normal and depth in addition to low-fidelity color maps to the model, and utilize texture coordinate data to extract textures effectively.

Contents

1 Introduction

1.1 Motivation

Technical advancements in both hardware and software solutions have dramatically expanded the boundaries of graphical fidelity. As a result, player expectations for visually stunning, immersive environments have surged. Modern games are judged not only by their mechanics and narrative depth but also by the realism and artistry of their visual assets.

Meeting these expectations can be challenging for game developers. Creating high-quality game assets, particularly textures, is time-consuming and cost intensive. Artists spend a significant amount of time creating different texture maps for each object all of which must be aligned with the 3D geometry in a tedious process known as UV mapping. This focus on asset production can divert time and resources from the core goal of game development: designing engaging and enjoyable experiences. Excessive effort on graphical fidelity can detract from the creative energy needed to enhance the player's interaction with the game world.

In recent years, we have seen an increase in tools and technologies that streamline the creative process, simplifying the path from idea to execution. Such tools should help artists quickly translate their creative vision into results, reducing repetitive and labor-intensive tasks. Texture generation is one use case of such tools.

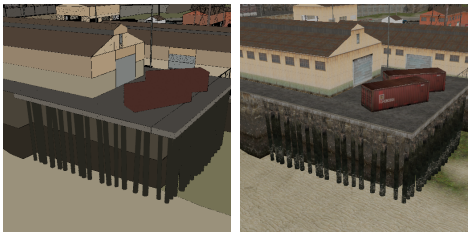
1.2 Prior Work

Following recent developments in generative deep learning models, the field of Neural Rendering has seen increased applicability. In particular, Denoising Diffusion Probabilistic Models[1] have facilitated the generation of photorealistic images, which directly relates to our texture generation problem. As such, various diffusion based texture and material generation pipelines have already been developed. Examples include *AI Texture Generator* by Polycam[2], Unity's *Muse*[3] and *Dream Textures* by Carson Katri[4]. However, these approaches focus on generating individual textures from textual prompts, and do not take scene context into account.

2 Overview

2.1 Methodology

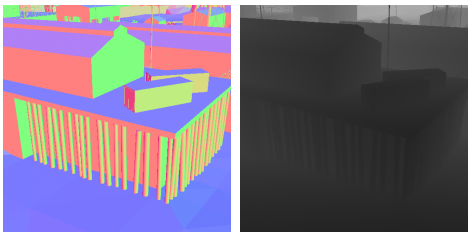
Instead of directly generating individual textures using a Diffusion model, we designed a multi-step pipeline that relies on full-frame renders of scenes with very primitive, low-fidelity textures applied to them. A model then generates a re-textured version of each frame, from which we can then extract the desired texture(s). This approach ensures the model is given access to the full context of the scene and generates textures that are stylistically coherent with each other.



Scene rendered with low-fidelity textures (left) and expected enhanced output (right)

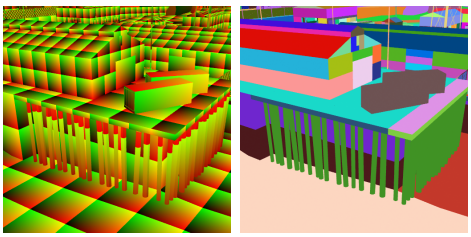
2.2 Auxiliary frame data

Since we have full access to the geometry when rendering the low-fidelity images, we can also export the normal and depth passes as additional data for the model. This is especially important considering the input color image likely contains large areas featuring the same static color without any shading, making depth and face orientation ambiguous.



Normal (left) and depth (right) passes

Finally, to facilitate texture extraction, we can also save the texture coordinate (UV) data for each pixel, as well as a map assigning a unique color to each individual texture used in the frame.



UV map (left) and material segmentation map (right)

2.3 Full pipeline

To generate textures for a scene, we start by manually assigning low-fidelity textures, which can be as simple as just plain colors, and render the color, depth, normal, UV and material segmentation maps from different camera angles. A generative model is then used to predict the color map for each frame if it had high-fidelity textures. Finally, we extract the desired texture(s) from each frame using the UV data, and combine the results.

3 Data acquisition

3.1 Requirements

To train a model for this task, a large image dataset is needed containing not only video game frames rendered with high- and low-fidelity maps, but also the corresponding normal, depth, UV, and material segmentation maps.

3.2 *Half-Life 2*

Due to the lack of a large enough dataset fitting our exact needs, we needed a way to generate one ourselves. We began searching for a game that already featured fully textured environments, and that we could easily modify for our needs. The 2004 First-person-shooter *Half-Life 2* came to mind, due to its use of photo-based textures and the availability of development tools publicly released by its editor, Valve Software. Additionally, due to the popularity of the game and its modding scene, many tools and resources have been developed around it.

One such resource of particular interest in our case is the Source Engine Blender collection, an archive of over 500 different maps from Valve's titles, including *Half-Life 2*, ported to the Blender 3D modeling and rendering software.

3.3 Exporting color and auxiliary maps from Blender

Having access to the game's maps in Blender meant we could animate a camera path scanning each scene from various angles, and render an image sequence. We rendered each image with flat shading to remove any shadows or lighting effects from the output, which could pollute the data. Exporting normal and depth maps was trivial, as Blender has built-in functionality for that. For texture coordinate and material segmentation maps, we used Blender's Python API to apply special materials to each object automatically. These materials could be configured to directly output the raw texture coordinates or a random color for each object.

4 Model description

4.1 Model Setup and Training Specifics

For recovering high- quality view images from low- quality view together with series input maps, we could reformulate the problem as training a model that can recover the most likely distribution for each of the pixels in the original input image that matches the target distribution in our dataset. With this pixel-aligned, distribution-minimizing problem in mind, we decided to tackle the problem with a variation of the conditional GAN model that can most effectively utilize the

per- pixel correspondence in our problem setting. More specifically, we chose to adapt a Resnet- based, encoder- decoder structure GAN model with a multi- scale discriminator setup to solve this problem, similar to the approach used in [5] and [6].

A complete structural overview of our final network pipeline is shown in the figure above. Unlike the Pix2Pix series of works, which use explicitly-numbered segmentation maps alongside instance maps as inputs, we avoid making any one-to-one correspondence segmentation maps. The exception is the material maps, which we render directly using the Blender engine. This design choice is due to the significant variation in object types across different game scenes. Instead, we use a 3-channel RGB low-quality view image as input and concatenate it with four other types of maps—each expanded into 3 channels—resulting in a 15-channel tensor for each single view. While the higher-dimensional input provides richer information for the generation task, our preliminary experiments have shown that the model can still generate high-quality results even without some input maps, such as the normal maps. However, determining the minimum set of maps required for optimal performance remains an open question.

For model training, we utilized a dataset comprising 10K image pairs in total, including normal, UV, depth, and material maps for each pair of low- and high-quality images. Since the data was collected from five distinct game scenes, and images within each scene are highly correlated, we adopted a train/test split strategy that used one scene (the train station dataset) as the holdout group to evaluate the model's generalization ability. For training on the remaining data, we employed a learning rate of 0.0002 0.0002, a batch size of 16, and trained the model on a single A6000 GPU for 9 hours across 60 epochs. To improve the model's generalization ability, we incorporated a discriminator matching loss alongside a perceptual VGG loss [7].

4.2 Adopting Attention Mechanisms for Better Spatial Understanding

Since the release of our baseline pipeline, significant advancements have been made in deep learning and computer vision. Among these, the transformer architecture has gained widespread attention for its ability to model long-range dependencies in NLP tasks and capture global spatial relationships in computer vision problems. Vanilla GAN models have also demonstrated potential for incorporating attention mechanisms, with works such as [8] showing promise in maintaining long-distance dependency relations using lightweight attention modules.

Inspired by the structure of SAGAN, we enhanced our generator's convolutional layers with attention modules. Among various attention mechanisms, we selected the Convolutional Block Attention Module (CBAM) [9] for its simplicity and effectiveness. As depicted in the full pipeline graph above, we integrated two CBAM modules into our pipeline: one after the down-sampling layers and another after the ResBlocks. This setup enhances both low-level feature expressiveness and semantic information understanding. Furthermore, consistent with practices in [8] and [10], we applied spectral normalization to every convolutional layer within the three discriminators. This addition stabilized the training process [10] and accelerated convergence compared to the baseline Pix2PixHD model, as evidenced by our preliminary ablation studies.

In addition to these modifications, we designed a channel attention mechanism within the generator's up-sampling layers. This mechanism improves feature selection and minimizes information loss during the up-

sampling process, following the approach outlined in[11]. Moreover, we replaced all ReLU activation functions in the original Pix2PixHD model with Mish activation functions to mitigate vanishing gradients and improve convergence[12].

4.3 Iterative Texture Recovery from Output Views

To extract a specific texture from an output view, we can use the material segmentation map corresponding to the view to select all pixels coming from said texture. By sampling the texture coordinate map at the location of each of these pixels, we obtain the coordinates where each pixel lies on the texture's albedo map. Of course, one view often does not contain enough information to extract a full texture, but we can repeat the process for multiple views to iteratively fill the texture.

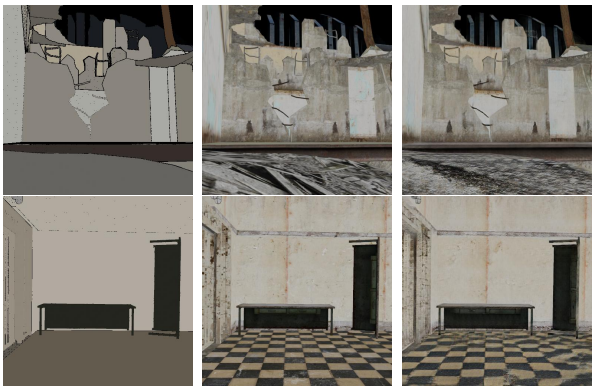
Something to consider during this process is that the farther away a pixel is from the camera, the less accurate that pixel's color will be. This is because far away surfaces cover less pixels, and as such those pixels' colors are actually averages from multiple samples of the underlying texture, depending on the type of texture filtering used when rendering the training set. While filling in a texture, we want to prioritize colors coming from closer surfaces, and not override them if a view contains the same pixels but viewed from farther away. We can accomplish this by sampling the depth map corresponding to each view, and keeping a depth buffer in memory when generating each texture. If the depth value associated with a pixel is higher than the saved value, we disregard that pixel.

Texture extracted without depth testing (left) vs with depth testing (right)

5 Results

5.1 Evaluating on in- domain data

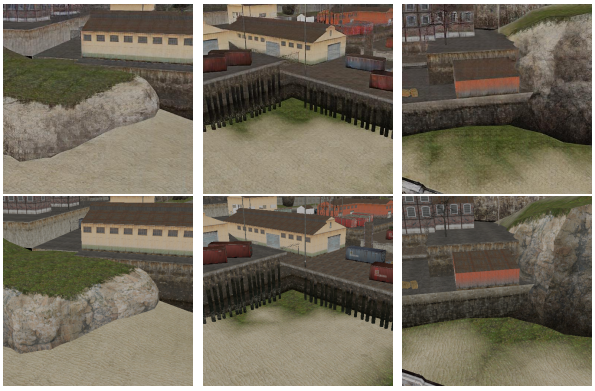
We chose the perform our test with our last model trained on the 9K dataset with the train station scene dataset as a holdout group. The examples we select for present here are all coming from different scenes and include both indoor and outdoor ones. In general, our model has shown a detailed and precise recovery of information when compared with group truth even at those places where per- material details are insufficient on the input images.



Input view (left), Ground Truth View (Middle), Synthesized View (Right)

We also test our model's performance when handling multiple continuous views to see if it can give a relatively temporally coherent synthesis results, which is critical to the quality of multiple- view texture extraction method

we proposed. And the results are also satisfying, as shown below for selected frames in 50 frames we tested. The comparison between our synthesized frames selections and the ground truth are as follows.



Synthesized views (1st row), ground truth views (2nd row)
1st (left), 25th (middle), 50th (right) frame with in the coast scene

We also inlcude the comparison among the models trained for different number of epochs here, and our improved models showed a quick convergence ability on such a large dataset with a relatively well reconstruction for only 10 epochs and more detailed and color- accurate results after 30 epochs.

Model tested on the same input after 10 epochs (left) and 30 epochs (right).

5.2 Generalization and ablation discussions

Since the model performs rather superior and accurate image synthesis capabilities for in-domain data, it naturally leads us to question its generalization ability for out- of- domain (OOD) examples. Since in our case it is hard for us to test the model on other games we rather choose to test it on our holdout train station scene as a representative of OOD examples here. Though the graphic style were constant throughout the game across different scenes, a completely unseen scene still challenges our model with unseen texture details during training time and the spatial relations among unseen objects.

Synthesized results on OOD data

Here we show the results from two OOD examples. The first synthesized image still retains a high level of details and consistency across the image, including the intricate texture on the front- ground road, though the background with trees still look a little blurry. The second example, though, shows a worse case where the model had the problem dealing with the opacity of front- ground street lamps and shows its difficulty handling with this kind of strict condition. We hypothesize that this kind of opacity issue can be largely solved with strong regularization term based on the material mask and depth information together and left it for future improvement. Finally, we set up an effective ablation studies to test our model's improvement upon the vanilla pix2pixHD pipeline. Under our setting, we keep track of the first few predicted images during the training stage of both our modified model and the baseline model. Since both models show the ability to converge after an enough number of epochs, we chose to collect the results both at an earlier stage of the training process with the number chosen to be 4 under our setting, with all other training parameters hold unchanged. For the results, the adapted pipeline has shown a much better spatial and color awareness at such a beginning stage of training, compared to the result from baseline model where the image is almost in grey scale and shows a lack of geometry details. This shows how effective our pipeline is

comparing with the purely convolutional networks used in pix2pixHD.

1st row: Results with baseline model, from left to right: input, output, GT
2nd row: Results with our model, from left to right: input, output, GT

5.3 Extracted textures

What follows are some examples of textures extracted from different views.

From left to right: A sand, road, and barn window texture

We observe significantly higher quality on tiled textures that are repeated across a scene, since those textures are likely to be shown in their entirety for multiple frames.

6 Conclusions

6.1 Summary

This approach does produce promising results, and greatly improves coherence between different textures in the same scene over other methods. However, it does come at a quality cost due to the texture extraction component which impacts the maximum resolution and is very reliant on temporal coherence between frames.

6.2 Future work

The current pipeline is specifically trained on images from one game and as such will always produce outputs that resemble it. An improved version of this model would be trained on a more diverse data set and could be fine tuned to specific styles. This will require significant data acquisition efforts, especially considering our current model's noticeable drop in performance when used on out-of-domain data. Additionally, the relatively low output resolution has a direct impact on the quality of the generated textures.

Another important aspect we have not addressed is the creation of detail maps such as normal, bump, and roughness. These are essential to achieving graphically fidelity and a full solution should be able to generate them in addition to albedo.

References

[1] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020.

[2] Polycam. AI Texture Generator for Blender, Unreal, Unity.

[3] Polycam. Unity Muse: Unlock your Creative Potential with AI.

[4] Carson Katri. Dream Textures.

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

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

[7] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution, 2016.

[8] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks, 2019.

[9] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module, 2018.

[10] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks, 2018.

[11] Bingchen Liu, Yizhe Zhu, Kunpeng Song, and Ahmed Elgammal. Towards faster and stabilized gan training for high-fidelity few-shot image synthesis, 2021.

[12] Diganta Misra. Mish: A self regularized non-monotonic activation function, 2020.