PedroVerse: Streamlining 3D Assets Stylization in the Wild Using Lightweight Methods

[Amr M Sharafeldin]* SFU
[Stephen Ehebald]* SFU
[Quang Minh Dinh] SFU
[Shuqi Mou S] SFU

Figure 1: PedroVerse: A Blender add-on for stylizing 3D assets by directly editing their albedo and object-space normal maps. Our pipeline enables a wide range of non-photorealistic looks without relying on external tools or engine-specific shaders.

ACM Reference Format:
[Amr M Sharafeldin], [Stephen Ehebald], [Quang Minh Dinh], and [Shuqi Mou S]. 2026. PedroVerse: Streamlining 3D Assets Stylization in the Wild Using Lightweight Methods. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 ABSTRACT

We introduce PedroVerse, a Blender add-on that stylizes 3D assets by editing their UV maps. We twist the arms of classical image editing and lightweight deep learning models to bring hyper-realistic 3D assets into the vibrant, unlimited possibilities of non-photorealistic rendering (NPR).

We define a pipeline composed of a sequence of operations on the albedo and object-space normal maps, including style transfer, recoloring, and a variety of geometric abstraction methods. These, when permuted, can instantly produce a wide range of NPR looks—without needing to process textures externally in photo-editing software or rely on engine-specific shaders that often do not generalize well across game engines or VR environments.

PedroVerse is a tool that seamlessly integrates NPR into the 3D environment design pipeline.

*Denotes equal contribution.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 INTRODUCTION

Despite advances in photorealistic rendering and novel view synthesis based on real-world data[], animation and game studios are increasingly shifting toward stylized representations. These approaches support unique design languages, enabling studios to create distinct visual styles that set their work apart. As a result, they are exploring creative 3D modeling approaches that go beyond realism.

3D assets are the fundamental modular unit of any 3D environment in films, games, and interactive media. Adapting an entire environment can involve tweaking shading in rendering software˜[**?**]{3} or using post-processing effects˜[**?**]{4}, but these methods are time-consuming and not platform-agnostic. For instance, a shader built in Blender often cannot be easily exported to a game engine for rapid prototyping.

Our key insight is that most assets already include UV maps—especially albedo and normal maps. These carry strong visual priors and can be tweaked to guide a particular visual direction. UV maps are also universally supported and inexpensive to render.

We build on this by defining our stylization operations entirely in UV space. This allows us to combine classical image processing with lightweight deep learning to build a flexible, modular pipeline. Palette transfer on albedo and normal maps brings the look to life, with normal maps preserving dynamic lighting. Our system supports quick stylization with simple parameter tweaks—making it efficient and accessible, especially for prototyping and small teams.

We introduce *PedroVerse*, a Blender add-on that stylizes 3D assets by editing their UV maps. The pipeline combines style transfer, recoloring, and geometric abstraction on albedo and object-space normal maps. These can be permuted to instantly generate a wide range of NPR looks—without relying on external texture tools or

engine-specific shaders. *PedroVerse* integrates seamlessly into 3D environment design workflows.

3 RELATED WORK

Prior work has explored stylized rendering by directly manipulating object-space normal maps or applying artistic filters to 2D images. For example,[5] generated NPR appearances by baking object-space normal maps, as they encode direction, and manually painting over them. The painted details vary based on direction due to the direction-dependent variation of the object-space normals—a process commonly referred to as the painterly effect.

In[6], the authors introduced a generalizable Krita extension that applies an 8-bit pixel art filter on 2D images and corresponding normal maps to create dynamic lighting effects. Other approaches, such as[7], proposed a feed forward model that transforms 2D images into painterly representations. Classical algorithms like those in[8] also generate brush stroke-based stylizations using geometric and color-space cues.

Our method extends these ideas by operating directly in UV texture space and integrating both classical and learning-based techniques in a modular pipeline applicable to 3D assets.

4 METHOD

4.1 Pipeline Overview

Our pipeline starts by editing the style of the albedo channel. For that, we rely on a lightweight, pretrained style transfer model. Then we edit the color tone: we provide the option of recoloring the albedo directly or recoloring the style image before style transfer. These two functionalities offer multiple degrees of freedom over the color tone and overall look of the albedo channel. Next, we experiment with different classical filters and lightweight deep-learning stylization methods that introduce structural changes to the albedo maps ranging from brush strokes and watercolor effects to anime-style rendering and, finally, 8-bit pixelation. Finally, to enable dynamic lighting interactivity, we apply the same structural changes to the object-space normal maps, allowing for true dynamic light response.

4.2 Style Transfer

Our style transfer pipeline produces high-quality textures with low latency (2–3 s). We encode both the content and style images into feature representations, then use the content-to-style ratio to interpolate between these encoded features and the original input image features—giving precise control over the blend. As shown in 2, at low content-to-style ratios the style features dominate and the user-provided styles are fully replicated; at high content-to-style ratios more of the original texture's content is retained.

4.3 Color Palette Change

The recolouring method is using a Palette-based Photo Recolouring [9] technique for the artist's ease of use. This component accepts an input image, which may be either an albedo UV map or a texture-transferred style image. A colour palette is then generated using a modified k-means clustering algorithm with k being specified by the user. The clustering is conducted in the LAB space as this representation aligns most effectively with the colour grouping approach outlined in the referenced method. The user will later modify the colours of the palette operating an hsv colour picker for the advantages it provides in people's colour perception.

4.4 Geometric Abstraction

Our geometric abstraction component applies structural stylization filters on both albedo and normal maps. We experimented with four different methods to generate NPR looks. The first was the classical Stroke-Based Rendering algorithm by[8], which achieves a hand-painted look by painting different layers of brush strokes over a blurred input image. The brush strokes are defined as Bézier curves and are swept across the image gradients.

Next, we experimented with a more modern approach proposed by \cite{ref7}. Unlike the classical brushy-style algorithm, neural paint transformers optimize both the spatial placement and color properties of brush strokes by computing the photometric loss between the painted output and the ground-truth image and propagating gradients accordingly. We also experimented with SLIC, which clusters pixels into superpixels by applying k-means in a combined five-dimensional space of CIELAB color and XY coordinates, with a compactness parameter to balance color similarity and spatial proximity [16] Another effect we experimented with was the Pyxelate engine used by[6], which allows us to create 8-bit retro looks. Finally, we experimented with an anime style explained by[12], in which multiple layers of Voronoi patterns are blurred and overlaid on the bilaterally filtered oil-painted UV map. This combination creates an anime-style watercolor effect.

Now, our pipeline allows the same operations to be applied to the object-space normal map. This, however, comes with challenges, as a wrong change in color in the normal map can incorrectly alter the direction in which light bounces off the surface, resulting in artifacts. Thus, we found that image-gradient-aware methods, such as the brush strokes algorithm, are less prone to artifacts than, for example, the neural paint method, which splats brush strokes with a slight degree of noise—causing wrong colors to appear in incorrect areas.

Additionally, when using SLIC on normal maps, we found that setting a low compactness value—forcing the clusters to weigh color similarity more heavily than spatial coherence—helps mitigate sudden shifts in color, which otherwise result in abrupt changes in curvature. Finally, to ensure the pipeline respects the UV space of the normal map, we apply a bitwise mask between the stylized normal and the original input normal map.

5 IMPLEMENTATION

We use the Blender Python API to build the UI. Our style-transfer method runs in TensorFlow, while basic image filters leverage OpenCV and scikit-image. The neural painter transformer model is implemented in PyTorch. For the brush-stroke effect, we rely on a reimplementation by [Someone], which exposes four predefined hyperparameter presets—Impressionist, Expressionist, Colorist Wash, and Pointillist—that users can select in the UI. Pixelation is handled by the open-source Pixelation Engine from [13]. To avoid module-import complications within Blender, each pipeline component is invoked via a subprocess call. Finally, we've packaged the entire system as a single Blender add-on [14].

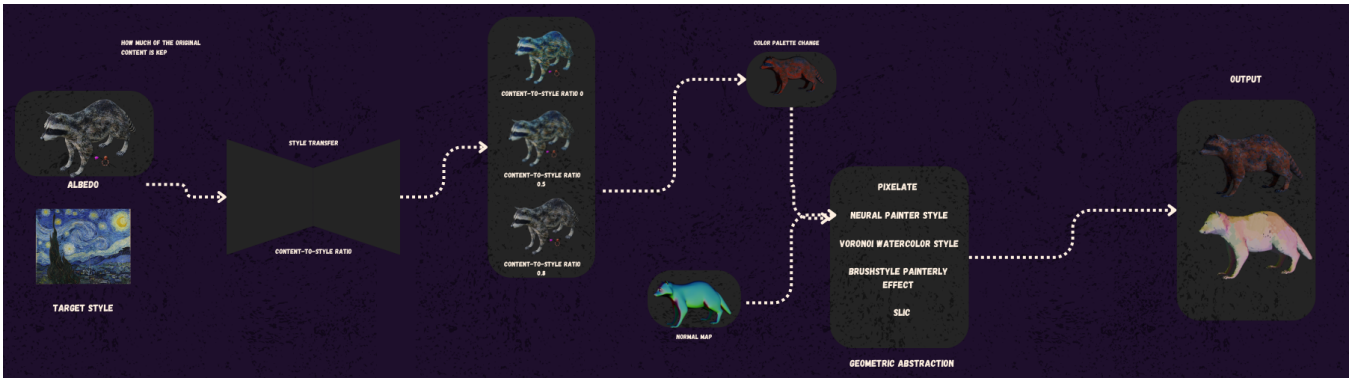

Figure 2: Full pipeline: from input textures through style transfer, palette recoloring, and geometric abstraction applied on both albedo and normal maps.

6 RESULTS

\section{Results}

We proposed a simplified 3D asset stylization pipeline that, despite the simplicity of the methods used and its reliance solely on normal and albedo maps, unlocks a wide variety of applications. Our add-on enables the creation of stylized weapon assets for games and the stylization of modular assets that can be assembled into complete scenes—all with a fully interactive UI offering control over both colors and painted effects. Moreover, the pipeline is highly efficient: even when using the most underoptimized method, *Paint-Transformer*, it takes at most one minute to stylize a single asset. These results are showcased in 3

7 LIMITATIONS AND FUTURE WORK

7.1 Current Limitations

While applying the geometric abstraction methods to the normal map and albedo channels, there is no guarantee of consistency between the patterns and strokes generated across both channels. Additionally, since the methods are sourced from different codebases, we cannot ensure that they represent the most optimized implementations. Some of the code may be outdated, lacking modern practices such as avoiding for-loops or leveraging parallelization.

7.2 Future Improvements

While our pipeline already supports a wide range of styles, it can be further expanded. Staying within the constraints of UV maps, we can explore the use of additional texture maps such as curvature and roughness. For instance, we can build upon the method proposed in [?] to define style transfer in a differentiable manner, leveraging the results from intricate filter decomposition as alpha masks applied to normal maps, roughness, and more.

Furthermore, we are investigating the use of CLIP-based models for style transfer instead of the traditional VGG-based approach. To that end, we have developed a proof-of-concept CLIPstyler implementation with the following command-line configuration:

```
--text "Van Gogh Starry Night" \
--iterations 200
```

and these hyperparameters in `predict.py`:

```
cfg = dict(
    lambda_tv     = 6e-3,
    lambda_patch  = 6500,
    lambda_dir    = 2600,
    lambda_c      = 400,
    crop_size     = 128,
    num_crops     = 128,
    thresh        = 0.8,
    lr            = 3.2e-4,
    max_step      = iterations,
)
```

To integrate our CLIP-based style transfer prototype into a seamless Blender add-on, we expanded the pipeline to load models and run inference entirely via subprocess calls. First, the target asset's albedo or normal UV map is exported from Blender to a temporary PNG file. Then, within Blender we invoke our Python script:

```
python predict.py --image temp_uv.png \
                  --text "Van Gogh Starry Night" \
                  --iterations 200
```

Under the hood this uses PyTorch with CLIP ViT-B/32 for text–image embeddings, a lightweight U-Net to process 512×512 UV patches, and a frozen VGG-19 encoder for content-preservation losses. We stream the script's console output back into Blender's progress UI, and once complete we reload the stylized UV map into the original material slot. On an NVIDIA RTX 4060, a full 200-step run on one 512×512 map takes about 25 minutes; subsequent maps can be batched in the same session to amortize model loading, and for animations our `test_video.py` processes each 512×512 frame in roughly 3 seconds.

.

REFERENCES

[1] Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. *ACM SIGGRAPH*, Article 98, 1–17.
[2] Kerbl, B., Kopanas, G., Leimkühler, T., and Drettakis, G. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM SIGGRAPH*, Article 138, 1–14.
[3] Joey C-quel. 2023. Quick toon shader effects, NPR – Blender tutorial. YouTube. https://www.youtube.com/watch?v=faz3oSuDivE
[4] underscore. 2016. UE4 Tutorial: Toon Shader (Borderlands/Jet Set Radio). YouTube. https://www.youtube.com/watch?v=0UBNXneL1oo

[5] Cody Gindy. 2023. Making 3D animation look painterly (it's easier than you think). YouTube. https://www.youtube.com/watch?v=s8N00rjil_4

[6] Gandeaga, G., Iliash, D., Careaga, C., and Aksoy, Y. 2022. DynaPix: Normal Map Pixelization for Dynamic Lighting. *ACM SIGGRAPH Posters*.

[7] Liu, S., Lin, T., He, D., Li, F., Deng, R., Li, X., Ding, E., and Wang, H. 2021. Paint Transformer: Feed Forward Neural Painting with Stroke Prediction. *Proceedings of ICCV*, 14613–14622.

[8] Hertzmann, A. 1999. Stroke-Based Rendering. *ACM SIGGRAPH 1999 Technical Sketches and Applications*.

[9] Chang, H., Fried, O., Liu, Y., DiVerdi, S., and Finkelstein, A. 2015. Palette-based Photo Recoloring. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 34(4), Article 139, 1–11.

[10] Kwon, G., and Ye, J. C. 2022. CLIPstyler: Image Style Transfer with a Single Text Condition. *CVPR*, arXiv:2112.00374.

[11] Kamra, C. G., Mastan, I. D., and Gupta, D. 2023. SEM-CS: Semantic CLIPStyler for Text-Based Image Style Transfer. *ICIP*, arXiv:2303.06334.

[12] Stylized Station. 2020. How I create Anime-Style Textures in 3D. YouTube. https://www.youtube.com/watch?v=h8llGEKIQT0&t=352s

[13] Sedthh. 2020. Pyxelate: Retro pixel art generator for Python. GitHub. https://github.com/sedthh/pyxelate

[14] Blender Foundation. 2024. Blender Manual: Add-ons Preferences. https://docs.blender.org/manual/en/latest/editors/preferences/addons.html

[15] Reimann, M., Büßemeyer, M., Buchheim, B., Semmo, A., Döllner, J., and Trapp, M. 2024. Artistic Style Decomposition for Texture and Shape Editing. *Visual Computer*, 41, 2107–2122. DOI:10.1007/s00371-024-03521-0.

[16] Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., and Süsstrunk, S. 2012. SLIC Superpixels Compared to State-of-the-Art Superpixel Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11), 2274–2282.

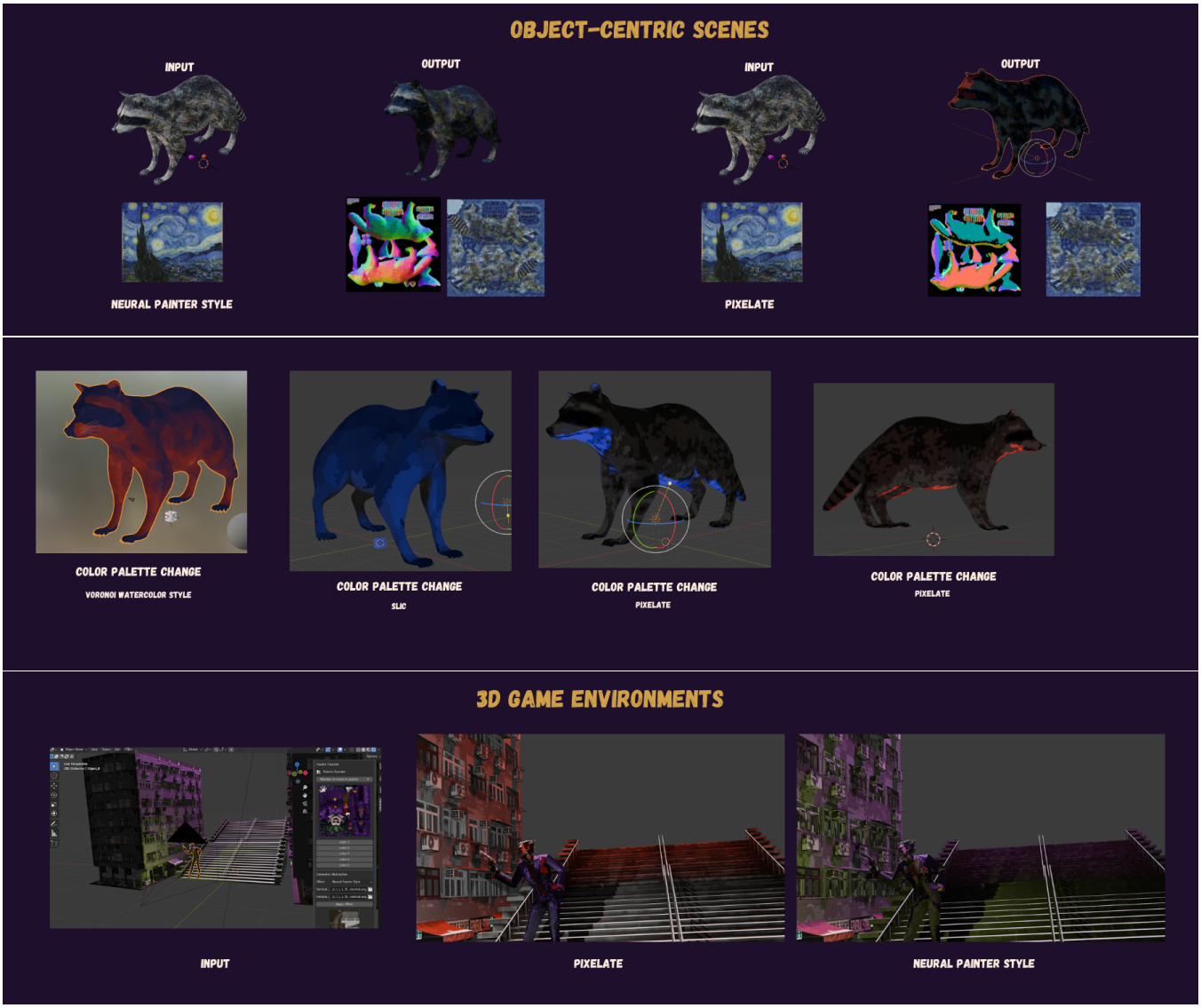

Figure 3: Results of our Blender add-on across both object-centric assets and full 3D environments.