From Pencil Lines to 3D Realms: Sketch Stylization with NeRF

Steven Hyun, Pooja Ravi, and Itay Kozlov

Abstract—In this paper, we propose two distinct approaches to generate and stylize a 3D object from a 2D set of sketches. To our knowledge, no prior work has been done on directly stylizing 3D models generated from sketches using neural radiance fields (NeRF). First, we explored going from the sketches to stylized sketches and then to a 3D model. This enables coloring in the sketches based on a style image to construct a 3D object with a specific style. We also investigated few-shot NeRF and different types of sketch generation methods in our approach. The second method was going from sketches to a stylized 3D object directly. This required prior conditioning based on text or images that could be fed in to guide the NeRF model directly. This approach allowed not only color editing but also texture editing through the manipulation of density functions. In this paper, we demonstrate that going from sketch to stylized 3D object is not only possible but can be achieved through various methods.

GitHub repository: https://github.com/01pooja10/Sketch2D-To-Style3D

Index Terms—Neural radiance fields, sketch to 3D, Neural style transfer, CLIP

1 Introduction

In recent times, the field of 3D modeling has been pervasive in various industries, going all the way from animation and video games to life-saving applications in healthcare research. Neural Radiance Fields (or NeRF) have been at the heart of recent progress in this sphere, allowing efficient compression of complex 3D scenes into distributed neural representations. However, using NeRFs for 3D model creation is not an easy task, and requires large samples of perspective images to properly capture complex 3D scenes. Some of these challenges were addressed by introducing few-shot NeRF methods, reducing the number of images required for training [1] and recent advances were able to increase the flexibility of NeRF models through stylization schemes [2], [3], allowing to stylize the generated 3D image using example images, or text. However, one challenge remains, the requirement for obtaining various high-quality photo-realistic images to train the NeRF models on the 3D scene.

It was the goal of this project to bridge this gap between the artist's vision and the 3D model by using a Sketch to Stylized NeRF scheme. This will allow an artist to generate a much simpler set of images in the form of sketches to be provided to the model, along with text representation of the desired 3D model style for more flexibility and control. This way, a vibrant 3D model of the artist's vision will only be a few sketches and one sentence away. The main contributions of this work include:

1. Two novel methods for constructing a sketch to NeRF pipeline were explored, these taking the general forms 'sketch to style to NeRF' or 'sketch to stylized NeRF'.
2. Some initial testing of a few-shot sketch to NeRF stylization were explored, paving the way for further work and exploration in the field.
3. Existing NeRF modification through text approaches were augmented by adding W-MSE and sparsity terms to the loss function [1], with initial testing showing promising improvement when training on sketch-like images.

• *S. Hyun is with the MSc in Applied Computing program, University of Toronto, Canada.*
E-mail: steven.hyun@mail.utoronto.ca

• *P. Ravi is with the MSc in Applied Computing program, University of Toronto, Canada.*
E-mail: p.ravi@mail.utoronto.ca

• *I. Kozlov is with the MSc in Applied Computing program, University of Toronto, Canada.*
E-mail: itay.kozlov@mail.utoronto.ca

2 Related Work

2.1 SNeRF

The authors of SNeRF [4] present a novel method to stylize images post-rendering i.e. after obtaining the outputs from NeRF. They propose the usage of two loss functions that are minimized in parallel wherein the computational graph for one doesn't depend on or affect the other. The NeRF and stylization modules are trained and optimized in tandem. Note that here, the images are stylized once the outputs have been rendered through NeRF. Further, the stylized images (from different views) are again set as the target images while training NeRF and this helps capture style elements of each view separately.

2.2 CLIP-NeRF

CLIP-NeRF [5] is a multi-modal manipulation of neural radiance fields guided by CLIP. It uses a disentangled conditional NeRF that allows conditional generation based on

1. See related work section, 2.3.

shape and appearance conditioning. It also enforces view consistency while allowing CLIP to modify the shape and appearance of neural radiance fields. However, the authors have not released the full code for the conditional NeRF. Therefore, we use the color-editing code that they have uploaded, which demonstrates the color editing of neural radiance fields using CLIP without shape deformation.

2.3 NEF

The authors of NEF [6] present a way to reconstruct 3d shapes from edges. NEF adopts the overall structure of NeRF but instead uses it to reconstruct parametric curves from a 2D set of images. It introduces different modifications to enable the reconstruction of 2d edges into a 3d shape. Most pertinent of all to us are the loss functions: W-MSE and sparsity. Edges tend to be very sparse. If NeRF is trained only on the edge maps of an image, it tends to degenerate into a local optima [6]. W-MSE is a modified MSE loss function that gives a higher weight to edge rays/pixels. It is defined as:

$$\sum_{r \in R_i} W(r) ||C(r) - \hat{C}(r)||^2$$

where

$$W(r) = \begin{cases} \frac{|C^+|}{|C^+|+|C^-|} & C(r) <= \eta \\ \frac{|C^-|}{|C^+|+|C^-|} & C(r) > \eta \end{cases}$$

$|C^+|$ denotes the number of edge pixels and $|C^-|$ denotes the number of non edge pixels. This essentially gives higher weight to any pixels that should be part of an edge and lower weight to any pixels that are not an edge. This assumes that the edge pixels are more sparse compared to non-edge pixels. The authors set η to 0.3, which we also follow.

Another loss function of interest is the sparsity loss. Edges that are sparse in 2D images should also be sparse in 3D as well. To enforce this sparsity, the authors use a regularization term that encourages sparse densities for non-edge pixels. The authors define sparsity loss as

$$\sum_{i,j} \log(1 + \frac{E(r_i(t_j))^2}{s})$$

where i indexes non-edge pixels and j indexes the samples that were taken along a ray. The authors set s = 0.5, which we also follow.

2.4 InfoNeRF

In this work, the authors present InfoNeRF [1], an information-theoretic regularization technique used to allow NeRF training with a low number of images (as little as 4). This will also be referred to as a few-shot NeRF throughout this report.

While this work is not directly related to the style-to-NeRF objective of this project, it does represent a very important aspect needed for the realistic implementation of sketch to 3D. That is, making the process feasible for potential users by reducing the number of sketches necessary for training down a number producible by hand. Therefore, the InfoNeRF model was experimented with as a potential component of the sketch to 3D pipeline.

The essential idea behind this work is that the lack of information that results from training NeRF on only a few images can be resolved by performing a certain regularization across the rays used when training the model. In particular, the regularization term attempts to minimize the total entropy across each ray, given by $\mathbf{r}$, where the entropy is defined by

$$H(r) = -\sum_{i=1}^{N} p(\mathbf{r}_i) \log(p(\mathbf{r}_i))$$

where $\mathbf{r}_i$ for $i \in \{1, 2, \dots, N\}$ is a sampled point on ray $\mathbf{r}$, and $p(\mathbf{r}_i)$ defines the ray entropy at point $\mathbf{r}_i$. The regularization then uses the opacity

$$M(\mathbf{r}) = \begin{cases} 1 & \text{if } Q(\mathbf{r}_i) > \epsilon \\ 0 & \text{otherwise} \end{cases}$$

where

$$Q(\mathbf{r}) = \sum_{i=1}^{N} 1 - \exp(-\sigma_i \delta_i)$$

with the terms σ_i and δ_i being the observed density and sampling interval at point i respectively. The ray entropy loss is then given by

$$\mathcal{L}_{\text{entropy}} = \frac{1}{|\mathcal{R}_s| + |\mathcal{R}_u|} \sum_{\mathbf{r} \in \mathcal{R}_s \cup \mathcal{R}_u} M(\mathbf{r}) \odot H(\mathbf{r})$$

where $\mathcal{R}_s$ and $\mathcal{R}_u$ denote a set of rays from training images and a set of rays from randomly sampled unseen images respectively. Finally, another regularization term is given for information gain reduction, taking the form

$$\mathcal{L}_{\text{KL}} \left(P(\mathbf{r}) || P(\tilde{\mathbf{r}}) \right) = \sum_{i=1}^{N} p(\mathbf{r}_i) \log \frac{p(\mathbf{r}_i)}{p(\tilde{\mathbf{r}}_i)}$$

where $\tilde{\mathbf{r}}$ are rays obtained through slight rotations from ray $\mathbf{r}$. Using these two regularization terms along with the typical difference loss term enables the few-shot NeRF methodology. Intuitively, the idea is to make dense regions more likely to be bunched together using the entropy loss term and to ensure that rays stay mostly consistent with slight viewing rotations using the KL-divergence term.

3 Proposed Method

3.1 Sketch Extraction

Before being able to train a NeRF [7] model on a set of sketches from different perspectives of the scene, these sketches must first be obtained somehow. Constructing such a dataset, which may need to contain up to 100 different sketches for some of the NeRF methods attempted here, is a highly time-consuming process. Therefore, to test the various methods explored in this project, it was necessary to find a way to generate a large set of sketches automatically. Conveniently, there already exist many data sets of natural images taken from different perspectives of the same scene. Therefore, a simple way to obtain a set of sketch-like images for training involves extracting them from a set of natural images through the edge information contained in them.

Two such methods were tested: one using PIDINet and another using DFI [2], or Dynamic Feature Integration [8].

3.1.1 *PIDINet*

PIDINet [9] is a network that is used to extract the edges within an image. There has been a lot of work on Convolutional Neural Networks as edge detectors. However, convolutions do not have any inductive bias that encodes gradients. This is the reason why the authors of PIDINet came up with a different type of convolution called Pixel Difference Convolution (PDC). Rather than focusing on the values of the pixels themselves, PIDINet focuses on the difference between the pixel values. This gives it an advantage over traditional convolutions where the kernels have to learn from scratch how edges are represented. We opt to use PIDINet as our sketch generation network due to its ability to surpass human perception in edge detection with fast inference.

3.1.2 *Dynamic Feature Integration*

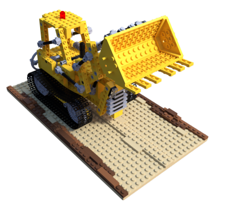
Dynamic Feature Integration (DFI) is a technique created to extract three essential features out of a given image, these being salient object segmentation, edge detection, and skeletal extraction. Two of these, namely object segmentation and edge detection, are essential components for extracting a sketch from a natural image (after all, a sketch can simply be thought of as an edge map of the salient, or most notable, elements of an image).

The simplest method one might think of when performing sketch extraction with DFI is to multiply DFI's outputs together, namely the salient mask and the edge map. Unfortunately, the edge map generated by DFI is not sufficiently consistent in its masking within the salient and non-salient regions, resulting in a highly noisy output after multiplication. However, there is a simple step that can make this method feasible. By passing the salient mask through a Gaussian filter to essentially 'smear' the mask and remove the high frequency components, the output becomes much closer to a sketch-like representation of the original image. One last step that was found to be very helpful was to then remove any other prominent details from the background of the image by amplifying the whole image and clipping it at the max value allowable for the given image format. The overall algorithm used is shown in algorithm 1 where an amplification of 15 was used (obtained through experimentation).

Algorithm 1 Sketch Extraction Using DFI

```
1: procedure ExtractSketch($img$)
2:     $img\_sal, img\_edge \leftarrow DFI(img)$
3:     $img\_sal \leftarrow GaussianFilter(img\_sal)$
4:     $sketch \leftarrow img\_sal * img\_edge$
5:     $sketch \leftarrow clip(sketch * 15)$
6: end procedure
```

2. Note that DFI-generated sketches were not used for testing most of the NeRF stylization approaches in this project as this method was discovered towards the end. So far DFI generated sketches were only tested with InfoNeRF.


(a) Original image

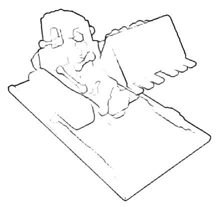

(b) DFI Extracted Sketch

Fig. 1: This shows an example of the sketches extracted using the DFI algorithm described in 1, where figure 1a shows the original image and 1b shows the resulting image.

3.2 Sketch to NeRF

3.2.1 *Sketch to Style to NeRF*

Once we obtain the sketches from PIDINet/DFI, the next step in the first approach is to stylize them. For this firstly, we identify a style image (a painting in our case) to condition the sketches on. For the first approach, we condition sketches on respective style images and further train and retrieve the outputs from NeRF/InfoNeRF. This essentially means that the 2D sketch-like edge maps need to be stylized and for that purpose, we employ a style transfer module.

Style transfer is a widely used technique that preserves the contents/objects in an image while modulating and switching up the style/color elements of the same image. The pretrained weights of multi-style transfer generative network [10] called MSG-Net, have been used. This model performs especially well when the image has a focus object such as the Lego tractor in our case. It uses a co-match layer which prioritizes learning the style-based second-order statistics or the gram matrix intrinsically.

Using the style images (paintings in Figures 3 and 4), we obtain the ground truth images that constitute the training set for NeRF/InfoNeRF. But when the images are fed as is, both NeRF and InfoNeRF weigh the background region much more than necessary. This distorts the final output 3D novel view since the background information is redundant and more often than not affects the structure of the 3D views.

For this reason, we removed the colors in the background by masking them out. By manually singling out the edge pixels, we obtain the masks that preserve the shape of the foreground region (the Lego tractor). All other pixels are set to a value of 0 thereby giving the effect of a white background with the stylized Lego tractor object acting as the foreground region of significance.

Finally, we ended up with a training set of 100 images as shown in Figure 5, and each of the 100 views was stylized individually using the pretrained weights of the model MSG-Net. [3]

3. Of course, only a small number of these images was used to train InfoNeRF, namely we tested it on training sets of 4 and 10 images.

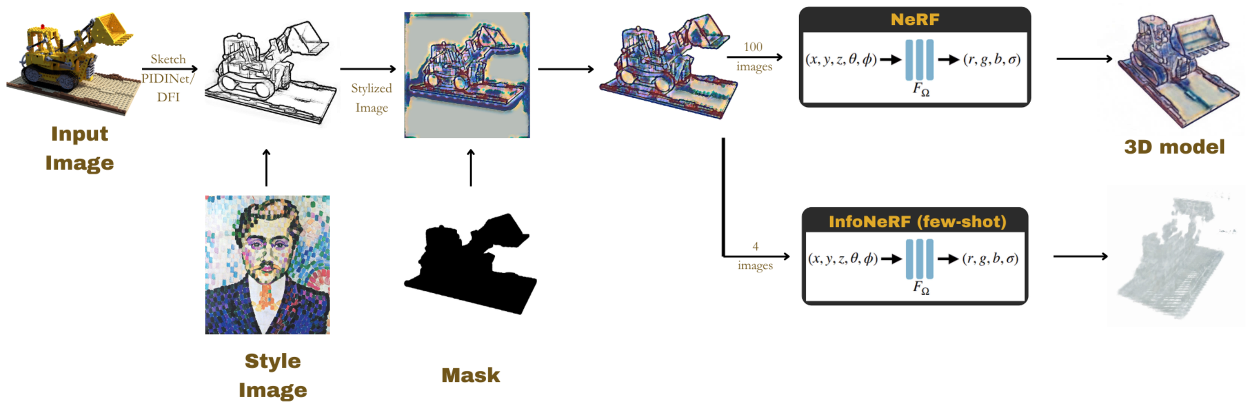


Fig. 2: Approach 1 - Sketch to NeRF flowchart

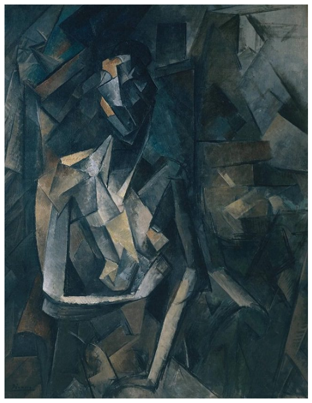


Fig. 3: Style 1 image

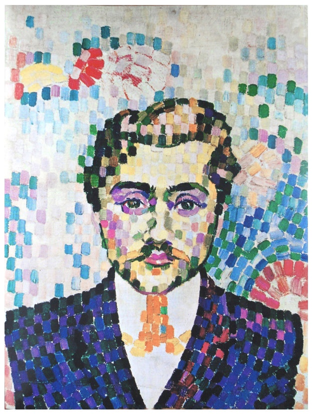


Fig. 4: Style 2 image

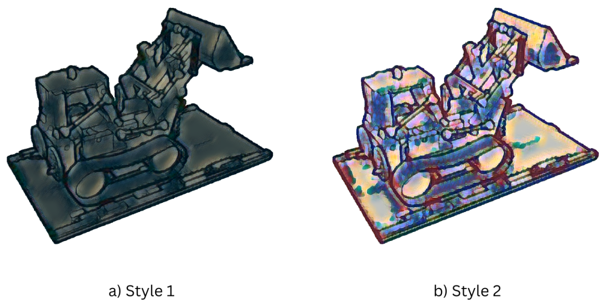


Fig. 5: Stylized input images

This led to the final training phase wherein we adopted and modified a Pytorch-based NeRF implementation and the InfoNeRF implementation available on GitHub for our training datasets. These NeRF models were trained from scratch on the aforementioned stylized image dataset. The 3D views generated were observed over 40,000 training iterations for regular NeRF and 30,000 iterations for InfoNeRF, and results were periodically saved.

3.2.2 Sketch to Stylized NeRF

The approach taken here is to go directly from sketch to stylized 3D. First, we take the 100 training images from the LEGO excavator dataset and apply PIDINet. Then we apply a sharpening filter to sharpen up the edges generated from the model. We also generate a mask of the image using a flood fill algorithm described in algorithm 2. Using the modified loss functions described in section 2.3, we train a standard NeRF model with W-MSE and sparsity loss for 20000 iterations. A sample generation from the NeRF model is shown in figure 6.

Algorithm 2 Flood Fill Algorithm for Extraction of Background

```
1: procedure FLOODFILL($img, threshold$)
2:     $img \leftarrow img > threshold$
3:     $visited \leftarrow array\_like(img)$
4:     for all boundary pixels p in image do
5:         BFS(img, p, visited)
6:     end for
7:     return $visited$
8: end procedure
```

After the NeRF model is trained, we transfer the weights from the NeRF model to CLIP-NeRF described in section 2.2. We also modify CLIP-NeRF with a modified loss where only the background and the edges are considered in calculating the MSE loss. Therefore, the overall loss function for CLIP-NeRF is

$$\mathcal{L} = \mathcal{L}_{edge\_mse} + \mathcal{L}_{clip}$$

where $\mathcal{L}_{clip}$ is the original clip loss used in CLIP-NeRF. $\mathcal{L}_{edge\_mse}$ is defined as

$$\mathcal{L}_{edge\_mse} = ||\text{target}[\text{edge\_mask}] - \text{gen}[\text{edge\_mask}]||_2^2$$

edge_mask = bitwise_or(target, background_mask)

The edge mask is a combination of the background mask and the target image (the edges). With this modified loss function, we train CLIP-NeRF for 10000 iterations with a text prompt. In this paper, we try different styles of sketches and perform an ablation study on whether the modified loss function qualitatively helps the output of the network.

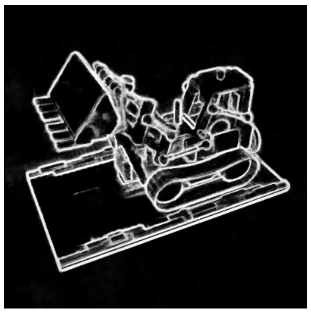


Fig. 6: Output from NeRF on PIDINet Sketches

4 Experimental Results

4.1 2D Style to NeRF Results:

The NeRF model was adapted and modified for two datasets (different painting styles) consisting of 100 images each. The images are 800x800 in resolution but NeRF was trained using half_res mode where resolution is reduced from 800 to 400. Hence the rendered outputs also correspond to the same resolution.

The NeRF model(s) were trained on the NVIDIA GeForce RTX A4000 GPU with 16GB of allocated memory. The training was performed for 40,000 iterations and the model's weights were periodically saved according to the progress made during training. The PSNR values have been displayed in Table 1.

For the purpose of testing the versatility of this approach, two different paintings (style images) were used to transfer style to the pencil sketches obtained from PIDINet. The style images are displayed in Figures 3 and 4 for reference. The reason we chose these images in particular was to observe how texture-dependent our model(s) were. The first style in Figure 3 contains sharper edges and different color palettes for the different geometric shapes in the image. On the contrary, Figure 4 contains many uneven patches and specific colors inside said patches. The training dataset of images shown in Figure 2 was mostly oblivious to the edges present in the first style image but captured the patchy appearance for style image 2.

TABLE 1: PSNR values

Models	Maximum PSNR (dB)
NeRF - Style 1	28.73
NeRF - Style 2	26.42

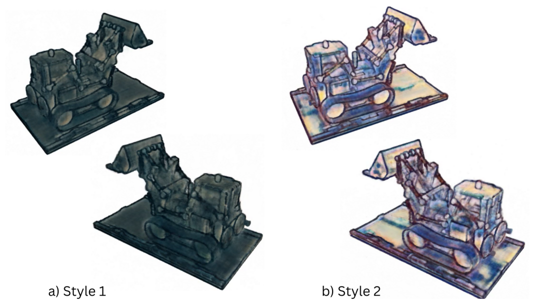


Fig. 7: 3D views from NeRF

Hence, while training, a similar pattern was noticed in the outputs rendered by NeRF. However, the model didn't fully converge and so the resulting 3D views generated (in Figure 7) contained moderately blurred foreground regions. As can be seen, the views for style 1-conditioned images have better color distribution both on and across edges whereas the color elements for style 2 get clustered around the edges instead of spreading out evenly across the object's surface.

4.2 2D Style to 3D InfoNeRF Results:

Here, InfoNeRF stylized sketch rendering was attempted using training sets with 4 and 10 images, and for both PIDINet generated detailed sketches, and DFI generated sparse sketches. The obtained results for this part along with the corresponding initial and stylized sketches are displayed in Table 2. It is evident that increasing the number of training data in InfoNeRF does not have substantial contributions to image quality, and still results in some noisy regions. Interestingly, it was also observed that the InfoNeRF rendering for the sparse sketch outperformed the one for the detailed sketch, retaining at least some of color pattern, unlike the one for the detailed sketch.

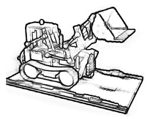
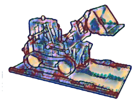
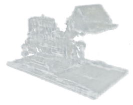
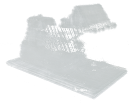
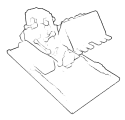
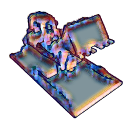
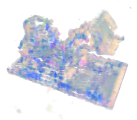
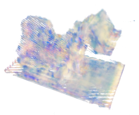
One possible explanation for this is that the regularization function of InfoNeRF seen in Section 2.4 attempts to eliminate the already low variation in the coloration of the high detail sketch, while the stylization of the sparse sketch is naturally more patchy already, as the style tends to stick to the edges. That being said, further exploration is required to properly analyze these results.

4.3 Sketch to CLIP-NeRF Results:

First, we tried training CLIP-NeRF on sketches directly generated from PIDINet with prompts ”A red Lego excavator” and ”A green excavator”. The result is shown in figure 8

The CLIP-NeRF when prompted with information about the texture of an object like ”lego”, tries to produce a lego-like texture as seen in figure 8 with the yellow dots on the surface. However, when prompted with just the color, it only filled in the color instead. Also, the CLIP-NeRF did not converge fully as the loss did not decrease after training for

TABLE 2: InfoNeRF results for 4-shot and 10-short training using both detailed PIDINet and sparse DFI generated sketches, which are then stylized. InfoNeRF results were obtained after 30K iterations for each scenario.

			PSNR (dB)	
Sketch extraction method	Original sketch	Stylized & Masked sketch	4-shot InfoNeRF	10-shot InfoNeRF
PIDINet			 26.40	 27.63
DFI			 29.34	 30.06

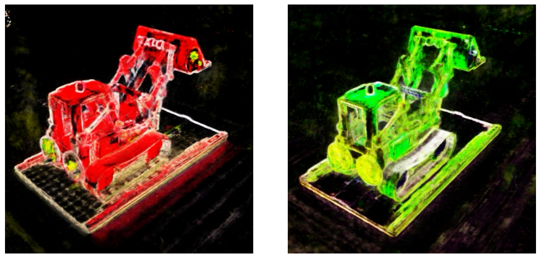

(a) A Red Lego Excavator (b) A Green Excavator

Fig. 8: This shows the output from CLIP-NeRF with the modified edge loss function when prompted with "A Red Lego Excavator" and "A Green Excavator"

more iterations. This is likely due to the "mode collapse" of the NeRF where it gets stuck in a local optimum. This was addressed in NeRF-Art [3] with relative directional CLIP loss whereas CLIP-NeRF uses absolute directional loss defined as the cosine distance between the rendered image and the target text. However, we have tried implementing relative directional CLIP loss for CLIP-NeRF but the output did not change at all. We theorize that this is most likely because we could not find a text prompt that adequately describes the source image. In relative directional CLIP loss, a pair of image/text embedding is required. That is, we need the source image, source text, target image, and target text. If the source text does not adequately describe what the source image is, then it is very hard for the model to converge.

Also, we noticed that the output from these models had very sparse colors. The reasoning for our modified MSE loss was that we wanted CLIP-NeRF to focus on filling in the gaps between the edges with color/texture information. However, we found that when filling in the gaps, the coloring tended to be very sparse. To help mitigate this issue, we flood-filled our training image with one color using a similar algorithm as 2. A sample training set is shown in figure 9. We use the colored-in image and train the CLIP-NeRF in the same way as before with the prompt "A Red Lego Excavator" to see if the prior colors mitigate the sparsity observed above.

The result of using the red color-filled sketches is shown in figure 10. We can see that it does mitigate the problem of sparse colors, especially near the surface below the excavator.

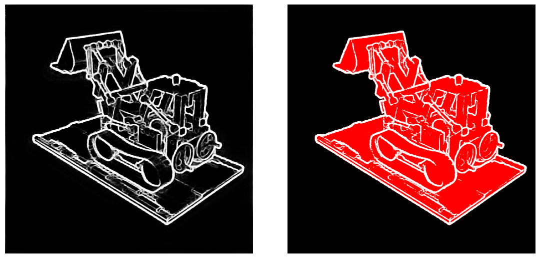

(a) Original (b) Flood-Filled

Fig. 9: 9a shows the original training image that we used. 9b shows the flood-filled coloured image

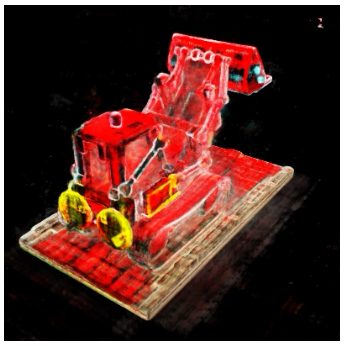
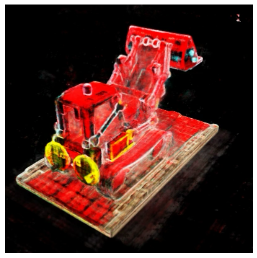


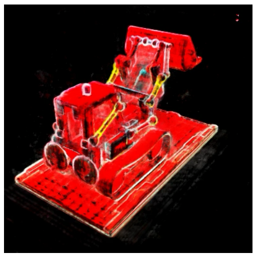
Fig. 10: Output from CLIP-NeRF using Red Colour Filled Sketches

Next, we performed an ablation study on the edge loss added to CLIP-NeRF. Since there are no ground-truth images, it is difficult to quantitatively assess the effect of the edge loss. We use the red color-filled sketches, with the prompt "A Red Lego Excavator" to perform the ablation study.

The result is shown in figure 11. We can see that for figure 11b, the output is less detailed compared to 11a. For instance, the surface where the excavator is sitting on is less Lego-like for 11b. This is because CLIP-NeRF considers the entire image when calculating the MSE loss. Therefore, if the entire image is filled with one color, it is very hard for texture to be added since there is a trade-off between preserving the original image and stylization with the text prompt. This was the intuition behind the edge loss where we only consider the edge and the background in the



(a) Output from Modified CLIP-NeRF



(b) Output from Original CLIP-NeRF

Fig. 11: 11a shows output from CLIP-NeRF with the edge loss. 11b shows the output from unmodified CLIP-NeRF

MSE loss. We wanted the MSE loss to only preserve the edges/background and the CLIP loss to fill in the color and detail in-between the edges. This can be evident through figure 11a where the wheels behind the excavator are colored in while the original CLIP-NeRF was not able to color it in.

5 Conclusion

Our work delves deep into the following problem statement: converting 2D pencil sketches (or edge maps) to stylized 3D views using style transfer methods, neural radiance fields, and text-prompt conditioning for a multi-modal approach. To summarize, we present two novel approaches to tackle this problem; first, we have stylization before rendering images from NeRF wherein we obtain the sketches, pass them through the neural style transfer model (MSG-Net), mask out the background, and then train NeRF on these stylized images. Here stylization is pixel-consistent and the results are in the style of the painting given. We also test a few-shot model called Info-NeRF with 4 and 10 images to test the ability of NeRF to reproduce reliable results with fewer volumes of input data.This is because under more practical circumstances, obtaining 100 views of pencil sketches with pixel-consistent stylization and the corresponding camera parameters is not viable. The second approach deals with stylization using text prompts while outputs are rendered by CLIP-NeRF. Here, we use transfer learning to substitute the NeRF in CLIP-NeRF with our model trained on the pencil sketches. This specifies a single color in the prompt rather than conditioning on a style image/painting.

6 Future work

Some limitations such as computational resource-related constraints obstructed the training process. For example, initially, NeRF was trained on a 12GB GPU but couldn't handle 800x800 images in their full resolution and hence it had to be halved to prevent the out-of-memory error. Further, even when trained on a 16GB GPU the model failed to fully converge (after 40,000 iterations) as such an MLP-query model needs to be trained in a full-fledged manner to preserve the finer details such as crisper edges, clearer textures, and smoother color gradients. This caused the edges to bleed out and some details to be mildly blurred. Another drawback was that during stylization, the style elements from the paintings stuck to the prominent edges instead of spreading evenly across the entirety of the object's surface. One possible reason could be the lack of denser pixels as the input image was a pencil sketch. Edges retain more style information while the spaces in between get filled with a uniform color and while rendering through NeRF also lose the painting texture.

Another future work that we want to explore is around CLIP-NeRF. We want to investigate why there is a mode collapse in CLIP-NeRF. One possible reason is due to its CLIP loss function. We theorize that this loss function does not provide sufficient guidance for the NeRF model to converge. Therefore, we would like to further explore different loss functions like the relative directional clip loss used by NeRF-Art [3].

Hence we plan on addressing these aspects in our future works.

Acknowledgments

We would like to thank Professor David Lindell and our mentor Parsa Mirdehghan for their timely guidance and support. We would also like to thank other course instructors and staff for their involvement with the course and for facilitating our project.

References

[1] M. Kim, S. Seo, and B. Han, "Infonerf: Ray entropy minimization for few-shot neural volume rendering," 2022.

[2] Y. Chen, Q. Yuan, Z. Li, Y. Liu, W. Wang, C. Xie, X. Wen, and Q. Yu, "Upst-nerf: Universal photorealistic style transfer of neural radiance fields for 3d scene," 2022.

[3] C. Wang, R. Jiang, M. Chai, M. He, D. Chen, and J. Liao, "Nerf-art: Text-driven neural radiance fields stylization," 2022.

[4] T. Nguyen-Phuoc, F. Liu, and L. Xiao, "Snerf: Stylized neural implicit representations for 3d scenes," 2022.

[5] C. Wang, M. Chai, M. He, D. Chen, and J. Liao, "Clip-nerf: Text-and-image driven manipulation of neural radiance fields," 2022.

[6] Y. Ye, R. Yi, Z. Gao, C. Zhu, Z. Cai, and K. Xu, "Nef: Neural edge fields for 3d parametric curve reconstruction from multi-view images," 2023.

[7] 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," *CoRR*, vol. abs/2003.08934, 2020. [Online]. Available: https://arxiv.org/abs/2003.08934

[8] J.-J. Liu, Q. Hou, and M.-M. Cheng, "Dynamic feature integration for simultaneous detection of salient object, edge, and skeleton," *IEEE Transactions on Image Processing*, vol. 29, pp. 8652–8667, 2020.

[9] Z. Su, W. Liu, Z. Yu, D. Hu, Q. Liao, Q. Tian, M. Pietikäinen, and L. Liu, "Pixel difference networks for efficient edge detection," 2021.

[10] H. Zhang and K. Dana, "Multi-style generative network for real-time transfer," *arXiv preprint arXiv:1703.06953*, 2017.