StyleNeRF: 3D Style Transfer With Neural Radiance Fields

Abdullah Hayran, Jake Pencharz, Barbara Rössle
Visual Computing & AI Lab, Technical University of Munich

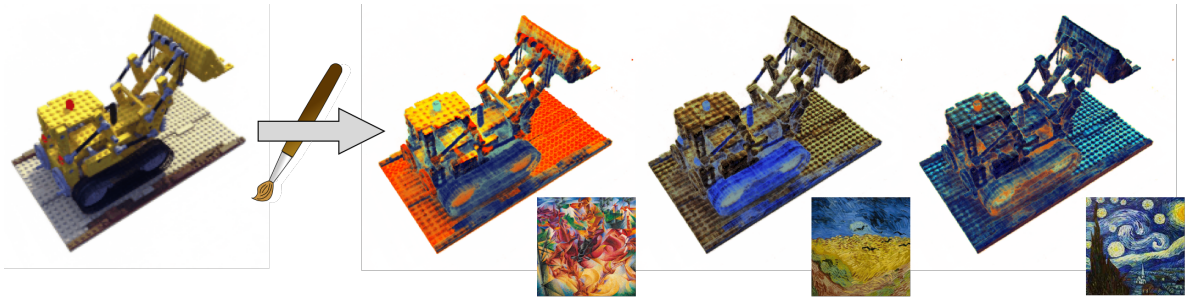

Figure 1: Examples of arbitrary styling applied to the same 3D model represented by a neural radiance field.

Abstract

Neural radiance fields (NeRF) [1] have proven their worth in generating high fidelity novel views of a scene from a set of 2D images. Recently, photometric variations in the input dataset were accounted for by disentangling the scene geometry from its appearance [2]. Inspired by this, we propose a structure preserving, viewpoint consistent 3D style transfer method for NeRF. Our approach consists of two consecutive stages: learning the scene geometry, and transferring the style of an arbitrary 2D image onto the scene. To alleviate the memory requirements during the style transfer stage, we also propose a two-stage backpropagation scheme whereby pixels are processed individually. Finally, we demonstrate the success of our approach using qualitative and quantitative evaluation criteria.

1. Introduction

Recently, neural rendering techniques have gained traction in synthesising novel views of a scene, given a sparse set of 2D input images taken from different angles. These techniques create a 3D model of the scene that can be used to generate images from previously unseen viewing directions. Neural Radiance Fields (NeRF) accomplish this by modelling the 3D scene within the weights of a multi-layer perceptron (MLP) [1]. Upon querying the model with a spatial location, and the desired viewing direction, the MLP returns the view dependent color and volume density of that point in the 3D scene. Then, classical volume rendering techniques are used to integrate samples from multiple points along a ray into the predicted color of a pixel in the rendered image. This approach achieved unprecedented fidelity over a range of scenes [1].

NeRF-W [2] built upon NeRF allowing it to successfully model scenes despite the input images including transient objects and a variety of photometric effects. NeRF-W took advantage of the inherent separability NeRF creates between the scene's color and its geometry. This approach demonstrated that the object's appearance could be varied without harming its structural integrity. One can therefore use NeRF to render 3D objects with arbitrary appearances, making it an ideal candidate algorithm from which to perform 3D style transfer.

2. Related Work

3D style transfer was accomplished by combining NeRF with a well established method of 2D style transfer. This section provides a brief overview of both concepts.

2.1. Neural Radiance Fields

NeRF [1] encodes a volumetric representation of a scene into two MLPs, and then uses volume rendering to render novel views of the underlying scene. The MLPs learn a mapping $F_\Theta : (\mathbf{x}, \mathbf{d}) \rightarrow (\mathbf{c}, \sigma)$ where the inputs are a spatial coordinate $\mathbf{x} = (x, y, z)$ and viewing direction $\mathbf{d} = (\theta, \phi)$. The outputs are a volume density σ at coordinate $\mathbf{x}$ and a view-dependent color $\mathbf{c} = (r, g, b)$. NeRF uses these density and color values to determine the value of each pixel in the raster image using ray tracing [1].

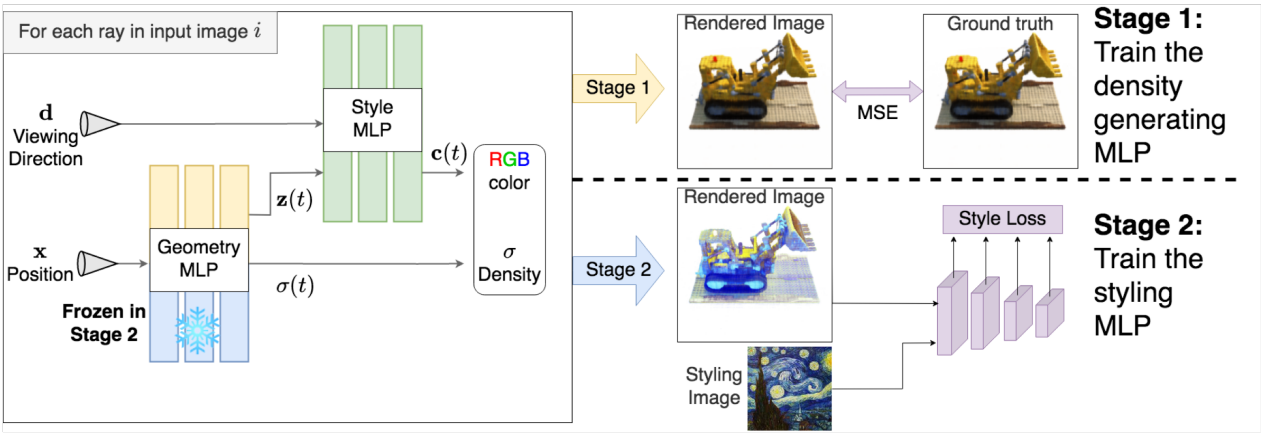

Figure 2: Proposed architecture for 3D style transfer consisting of two stages. In *Stage 1* the scene geometry is learned. In *Stage 2*, geometry MLP is frozen and style transfer is done by guiding the style MLP with the style loss.

The continuous integral along each ray is approximated by taking N discrete samples along the ray (using stratified sampling [1]). The estimated color is found by computing:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i \left(1 - e^{-\sigma_i \delta_i}\right) \mathbf{c}_i \tag{1}$$

Where $T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right)$ and δ_i is the distance between samples. Since $\hat{C}(\mathbf{r})$ is differentiable, a MSE loss can be used to compare rendered pixel values to ground truth pixels and propagate errors back through the MLPs. For a more detailed description, refer to Appendix A.

2.2. Style Transfer

Using deep neural networks for the creation of artistic images was introduced by Gatys et al. [3]. In their implementation an image is directly manipulated to reflect the styling of some artistic work while retaining the content of the original image. This was achieved by balancing a style loss with a content loss, both generated using a pre-trained convolutional neural network (CNN). The representation of an image, as it is passed through the layers of a CNN, increasingly captures the content of the image. Simultaneously, individual pixel values become less important. Higher layers in the network are more focused on objects and their arrangement than they are on the precise appearance of those objects. Therefore, given an input image $\hat{y}$ and a reference content image y, the content loss L_c at a high layer j in the network can be computed as: $L_c = \frac{1}{C_j H_j W_j} ||\phi_j(\hat{y}) - \phi_j(y)||_2^2$ where $\phi_j(y)$ is the representation of an image at layer j, and C_j, H_j, W_j are the number of feature maps (channels), height and width of that representation respectively.

The style of the image is extracted from lower layers. It is quantified using Gram matrices which contain the correlations between feature maps at layer j. The style loss at layer j is calculated as the difference between the Gram matrices of the input image and the style image $L_{s,j} = ||G_j^\phi(\hat{y}) - G_j^\phi(s)||_F^2$ where G's values are:

$$G_j^\phi(x)_{c-c'} = \frac{1}{C_j H_j W_j} \sum_{h=1}^{H_j} \sum_{w=1}^{W_j} \phi_j(x)_{h,w,c} \phi_j(x)_{h,w,c'} \tag{2}$$

3. Methodology

Our aim is to learn a 3D scene representation from a set of 2D images and to change the appearance of the scene using a styling image, while meeting the following criteria:

1. The 3D object resembles the styling image at a *high level*. The transfer of style therefore includes both color and texturing from the styling image.
2. Style is applied consistently from all viewpoints.
3. Structure and detail of the 3D object is retained.

To that end, we propose the two-stage approach in Figure 2 to transfer style onto a scene. In *Stage 1*, our aim is to learn the geometry of the scene using the standard NeRF training procedure. In this stage, the scene geometry is captured in the geometry MLP. The style MLP acts as an auxiliary network to facilitate this process.

Having learned a detailed representation of the scene geometry, the geometry MLP is frozen during *Stage 2*. This ensures structural integrity is not lost. The weights of the style MLP are reinitialised randomly to remove any bias towards the scenes original appearance. Crucially, the MSE loss is replaced by the style loss proposed by Gatys et al. (see Section 2.2. This loss compares the styling of each rendered image to the styling image. In contrast to traditional style transfer, no content loss was included. Structure

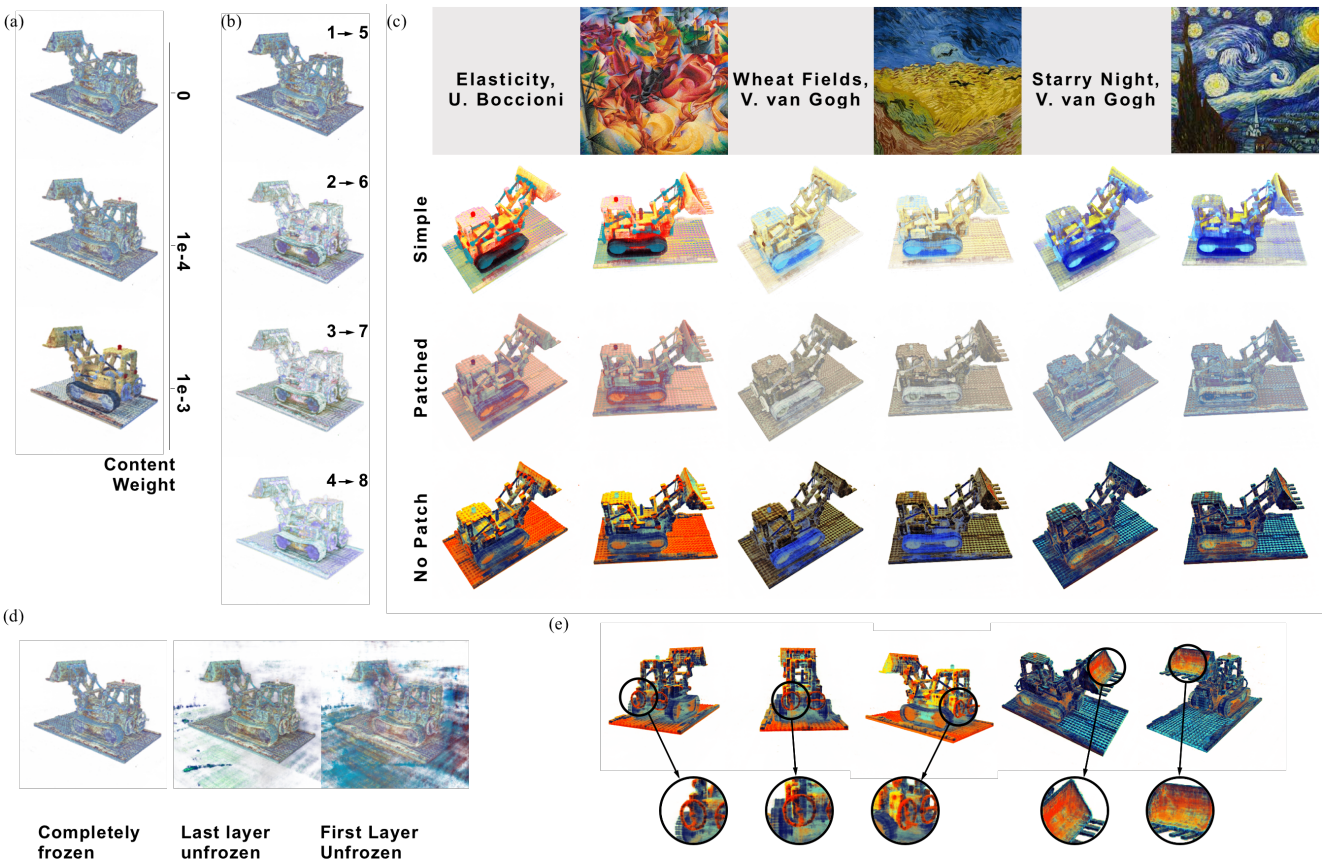

Figure 3: A summary of the experimental results. If not specified, "Starry Night" is used as the styling image. **(a)** Experiments with weighting content losses showed that content loss works against the styling process. **(b)** Working with first five layers of the style loss network provided the most style-consistent results. **(c)** Comparison of *simple*, *patched* and *no patch* methods for 3 styling images and 2 camera poses. The *no patch* method captures details and is styled in a globally coherent manner without sacrificing image quality. **(d)** Unfreezing the last layer of geometry MLP yields undesirable results. **(e)** Proposed architecture is capable of rendering the style with high viewpoint consistency for all styles (left set is styled by "Elasticity").

is preserved by freezing the geometry MLP. In this case, a content loss works against the style transfer process.

Unlike *Stage 1*, in *Stage 2* one cannot process pixels individually due to the nature of the style loss. The intermediate activations for potentially millions of sampled points must be stored for the entire rendered image. The resulting demand for extremely large GPU memory prevents the styling of high resolution images. With this practical limitation in mind, we propose three viable implementations.

Simple Method: The simplest approach uses low resolution images to avoid exceeding our memory capacity. This approach imposes a limit of 50×50 pixels per image during style transfer (on one GPU with 8GB memory).

Patched Method: To use higher resolution images, we propose a strategy where patches, rather than the entire image, are rendered and used to compute the style loss. This captures finer detail, but prevents globally consistent styling across the scene since patches are processed individually.

No Patch Method: Motivated by the downsides of the previous methods, we developed a method for globally consistent styling without sacrificing detail. The *no patch* method is identical to the *simple* method, but it is not restricted to using small images. This is achieved by splitting the backpropagation of gradients into two substages. In *Substage 1*, an image is rendered in inference mode and the gradients with respect to the style loss are stored for each pixel. Memory saving is realised in *Substage 2*, where pixels can be rendered individually and the corresponding gradients stored in the previous substage are propagated back through the style MLP (see Appendix C for details). The MLP weights are updated after processing all the pixels in the rendered image. This alleviates the memory constraint allowing styling on high resolution images. For a comparison between approaches see Figure 3c.

4. Experimental Results and Analysis

Several variations of the proposed architecture are implemented. The results are evaluated based on how well they meet the requirements laid out at the start of Section 3. Our model uses an 8-layer geometry MLP and 6-layer style MLP. Other than the density output, which uses Softplus, ReLU is used throughout the network. The following experiments were conducted on a single NVIDIA RTX 2070 GPU with 8GB of memory. In both stages the MLPs are trained for a single epoch, with learning rates of 5e-4 for *Stage 1* and 1e-3 for *Stage 2*.

Including Content Loss: As shown in Figure 3a, including a content loss (as described in Section 2.2) works against the style transfer process. Since freezing the geometry MLP guarantees structural integrity content loss provided no benefit.

Tuning Style Loss Layers: It is possible to tune the style network by choosing which layers in the VGG contribute to the style loss. After testing several sets of layers (see Figure 3b), it was determined that optimal color transfer is achieved using layers 1 to 5 of the style network.

Unfreezing Density Layers: To allow more flexibility, the final layers of the geometry MLP are unfrozen during Stage 2. This generates undesirable floating artefacts as seen in Figure 3d. This is likely due to the model having no incentive to enforce physical coherence during Stage 2.

Based on these experiments, all three practical implementations omit content loss, use layers 1 to 5 as style layers, and freeze the entire geometry MLP during styling.

4.1. Qualitative Evaluation

To evaluate the three implementations, we chose styling images with very recognisable artistic styles ranging from impressionism to futurism. All three variations successfully transfer elements from these styling images onto the 3D object without perturbing the structure.

Despite its speed, and ability to capture colors from the styling image, the *simple* method fails to capture higher level details in the styling image. This is caused by having to render low resolution images in the style transfer process.

The *patched* method is slower by ≈30 minutes, and provides higher levels of detail since no downsampling is required. This method produces locally plausible results. However, a careful qualitative examination reveals that a similar color pattern is repeated across the entire scene, which is a direct consequence of working with patches and not desirable for a successful style transfer.

Finally, the *no patch* method solves the problems of both of its predecessors. It is memory efficient due to the two-stage backpropagation scheme, and achieves a highly detailed result. Styling is performed across the entire image, preventing the monotonous patterning generated by the *patched* method. Colors, and their distribution across the object's surface correlate closely with the styling image. Despite taking ≈50 minutes, this method consistently produces the most appealing results.

Style Image	Simple	Patched	No-patch
Elasticity	**1244**	1323	1251
Starry Night	**1401**	1440	1500
Wheatfields	2403	**2304**	2495

Table 1: Comparison of different style transfer implementations using "E-scores" [6]. Lower scores represent higher effectiveness of style transfer.

4.2. Quantitative Evaluation

Neural style transfer is usually evaluated qualitatively or by using user preference studies [4]. A method exists to evaluate transfer of styling between 3D objects [5]. However, to the authors knowledge, there is no well established metric to evaluate style transfer from a 2D image onto a 3D object. We use a method, recently proposed by Mao-Chuang et al. [6], to quantify the effectiveness of style transfer between two images. This metric is calculated for 200 styled images, rendered at different poses, for each implementation (see details in Appendix B). Mean scores are presented in Table 1. This metric is not capable of reliably differentiating between the implementations. Therefore, it did not prove helpful in determining which performed best.

5. Discussion and Future Directions

By taking advantage of the disentangled color and density of volumetric scene representations in a NeRF, we are able to successfully transfer arbitrary styling onto 3D scenes. We implemented three approaches to style transfer and found that the *no patch* method provides the most pleasing results while being memory efficient.

One benefit of our proposed two-stage training procedure is its modularity. Once the geometry MLP has learned a structural representation of the 3D object, the second stage can be repeated for any styling image. This provides the ability to rapidly change the appearance of a 3D model which could be useful in product design or artistic works. However, a superior realisation would allow styling at inference time. We believe that this would be an exciting next step in generating 3D objects with arbitrary appearances.

The proposed method is not capable of transferring geometrical features from the styling image. Attempting to provide this flexibility by unfreezing layers of the geometry MLP generated artefacts. However another future direction could combine this approach with a content loss or regularization scheme to enforce physical coherence.

References

[1] 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 *European Conference on Computer Vision*, pp. 405–421, Springer, 2020. 1, 2, i

[2] R. Martin-Brualla, N. Radwan, M. S. Sajjadi, J. T. Barron, A. Dosovitskiy, and D. Duckworth, "Nerf in the wild: Neural radiance fields for unconstrained photo collections," *arXiv preprint arXiv:2008.02268*, 2020. 1

[3] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," 2015. 2

[4] Y. Jing, Y. Yang, Z. Feng, J. Ye, Y. Yu, and M. Song, "Neural style transfer: A review," *IEEE transactions on visualization and computer graphics*, vol. 26, no. 11, pp. 3365–3385, 2019. 4

[5] M. Segu, M. Grinvald, R. Siegwart, and F. Tombari, "3dsnet: Unsupervised shape-to-shape 3d style transfer," 2021. 4

[6] M.-C. Yeh, S. Tang, A. Bhattad, C. Zou, and D. Forsyth, "Improving style transfer with calibrated metrics," 2020. 4, i

[7] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *Proc. 8th Int'l Conf. Computer Vision*, vol. 2, pp. 416–423, July 2001. ii

APPENDIX

A. NeRF Details

NeRF [1] encodes a volumetric representation of a scene into two MLPs, and then uses volume rendering to render novel views of the underlying scene. The MLPs approximate learn a mapping:

$$F_\Theta : (\mathbf{x}, \mathbf{d}) \rightarrow (\mathbf{c}, \sigma)$$

where the inputs are the spatial coordinate $\mathbf{x} = (x, y, z)$ and viewing direction $\mathbf{d} = (\theta, \phi)$. The outputs are a volume density σ at coordinate $\mathbf{x}$ and a view-dependent color $\mathbf{c} = (r, g, b)$.

To ensure that the representation is consistent from multiple views, the volume density σ is predicted as a function of the spatial location $\mathbf{x}$ only. The model therefore processes $\mathbf{x}$ with 8 fully connected layers (256 nodes, ReLU activations) which output σ and a feature vector $\mathbf{z}$. The view dependent color is a function of that feature vector, and the camera's viewing direction. These two vectors are concatenated and passed to one final layer (128 nodes, ReLU activations) which outputs $\mathbf{c}$.

The MLPS can be thought of as two separate components. The first 8 layers of the model predict a density $\sigma(\mathbf{x})$ and a feature vector $z((x)$. Then the viewing direction and the feature vector from the previous layers are used to predict the view dependent color. NeRF uses these density and color values to determine the value of each pixel in the raster image using ray tracing [1].

A position along a ray is defined as $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ where $\mathbf{o}$ is the position of the camera, and $\mathbf{d}$ is the viewing direction and t is a position along the ray. The ray passes through a single pixel in the raster image. Within a range defined by near and far bounds (t_n, t_f), the pixel color corresponding to a ray $C(\mathbf{r})$ is found using:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\ \sigma(\mathbf{r}(t))\ \mathbf{c}(\mathbf{r}(t), \mathbf{d}) dt \tag{3}$$

$$T(t) = \exp\left(- \int_{t_n}^{t} \sigma(\mathbf{r}(\mathbf{s})) ds \right) \tag{4}$$

Where $T(t)$ denotes the probability that the ray travels from t_n to t without intersecting another object. An entire image is rendered by tracing a ray through every pixel of the raster image and calculating $C(\mathbf{r})$.

The continuous integral along each ray is approximated by taking N discrete samples along the ray (using stratified sampling [1]). The estimated color is found by computing:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i\ (1 - e^{-\sigma_i \delta_i})\ \mathbf{c}_i \tag{5}$$

Where $T_i = \exp\left(- \sum_{j=1}^{i-1} \sigma_j \delta_j \right)$ and δ_i is the distance between samples. Since $\hat{C}(\mathbf{r})$ is differentiable, a MSE loss can be used to compare rendered pixel values to ground truth pixels and propagate errors back through the MLPs.

B. Quantitative Evaluation Metric

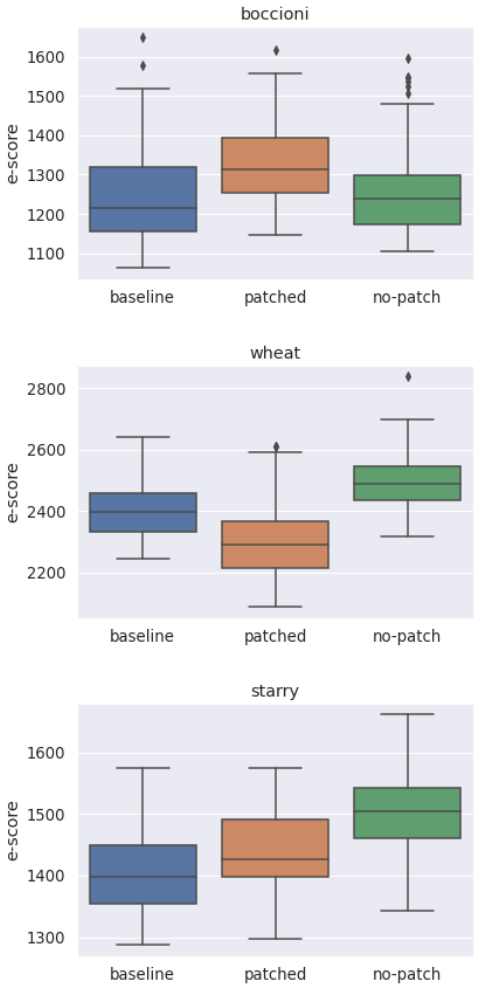

Figure B.1: Results of the E-statistic metric for 200 images per implementation method (each image is generated from a different pose). There is no consistency as to which method achieves the lowest E-statistic score across different styling images. This implies that the metric is not a useful quantitative measure for this use case.

This metric was proposed by [6] and measures the difference between two distributions. The distributions are generated from the styling image and the styled image respectively. According to this metric, a smaller difference represents more effective style transfer from the style image onto the styled image.

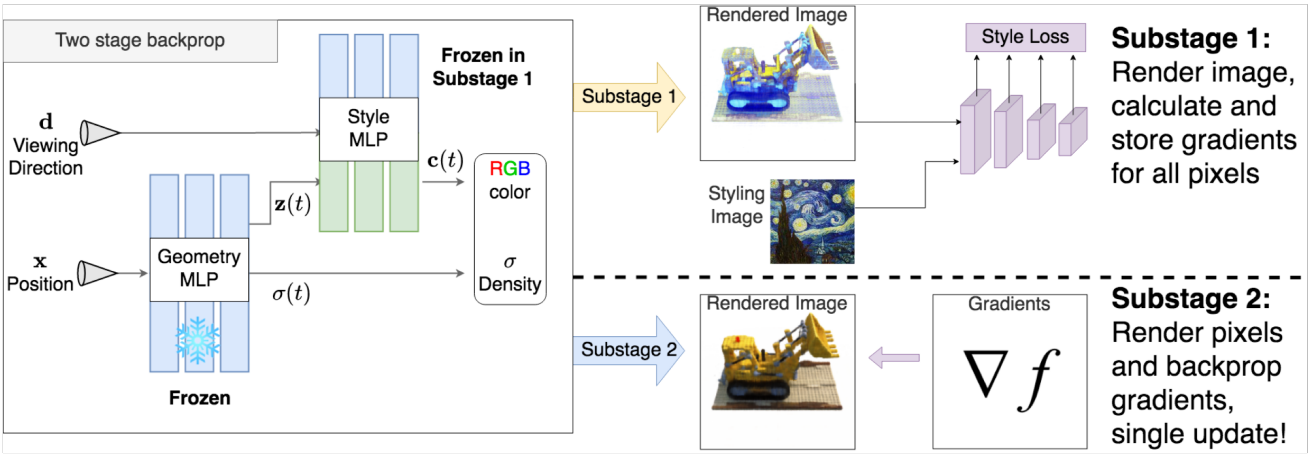

Figure C.1: Proposed architecture for 3D style transfer consisting of two stages. In *Stage 1* the scene geometry is learned. In *Stage 2*, geometry MLP is frozen and style transfer is done by guiding the style MLP with the style loss. The memory efficient approach, (*No Patch Method*), modifies *Stage 2* so that backpropagation is completed in two substages. This mitigates the memory issues without lowering the image resolution.

At every style transfer layer the feature map is projected into a low dimensional space. Then it is assumed that this representation represents the parameters of a Gaussian distribution. This distribution is compared between the style and styled images using KL divergence.

The projection matrix is generated using a set of "canonical" images. These images are not related to the content which is being styled but should represent a wide variety of images. In the paper, and in our reproduction, the BSD500 dataset is used [7]. For each image in this projection set $I_N = I_1, ..., I_N$ a covariance matrix similar to a Gram matrix is computed. At layer l, between feature maps i and j the covariance is:

$$Cov^l_{ij}(I_n) = \sum_p \left[f^l_{i,p}(I_n) - \bar{f}^l_i(I_n)\right]\left[f^l_{j,p}(I_n) - \bar{f}^l_j(I_n)\right] \tag{6}$$

Where $f^l(I_n)$ is the feature map generated by a CNN at layer l, and $\bar{f}^l(I_n)$ is the mean value of that feature map. Therefore $f^l_i(I_n)$ is the i'th element of the channel wise mean of the feature map. The sum is taken over all 'pixel values' in the feature map.

Then, singular value decomposition is applied on the average covariance matrix (the average is found across the entire projection dataset) Cov^l_{avg}. The authors choose t singular vectors from the decomposition and use this as a projection basis P^l.

Now, given some input image $I \notin I_N$ (for example the styling image, or the content image), the projection matrix is used to transform I's feature map into a space where the representation can be interpreted as a mean and variance of a t-dimensional Gaussian distribution $\mathcal{N}(\mu, \Sigma)$.

$$\mu^l_{proj}(I) = f^l(I)P^l \tag{7}$$

$$\Sigma^l_{proj}(I) = (P^l)^T Cov^l(I)P^l \tag{8}$$

At a given layer, the E-score is computed as the KL divergence between the style images distribution $\mathcal{N}_0$ and the styled image's distribution $\mathcal{N}_1$.

$$\begin{aligned} E_l = D_{KL}(\mathcal{N}_0||\mathcal{N}_1) = \frac{1}{2}(&\text{tr}\left(\Sigma_1^{-1}\Sigma_0\right) \\ &+ (\mu_1 - \mu + 0)^T\Sigma_1^{-1}(\mu_1 - \mu_0) - t \\ &+ \ln\left(\frac{\det\Sigma_1}{\det\Sigma_0}\right)) \end{aligned} \tag{9}$$

Finally the E-scores are added for each styling layer and this is reported as the total E-score for a given image I and some styling image I_s.

Note that in our implementation we use projection matrices dimensions 18, 100, 128, 128, 256 (set using the t parameter).

C. Memory Saving Implementation Details

The *no-patch* implementation which is used to ensure that images with high resolution can be styled is displayed in Figure C.1. Backpropogation during stage 2 is split into two sub stages which reduces the memory required to process an entire image.