Towards 3D Neural Style Transfer

Jo Mazeika, Jim Whitehead
Computer Science Department
UC Santa Cruz
Santa Cruz, CA 95064 USA
{jmazeika, ejw}@soe.ucsc.edu

Abstract

Neural Style Transfer was first unveiled by (Gatys, Ecker, and Bethge 2015), and since then has produced fantastic results working with 2D images. One logical extension of this field would be to move from 2D images into 3D models, and be able to transfer a notion of 3D style from one model to another. Here, we provide steps towards both understanding what style transfer in a 3D setting would look like, as well as demonstrating our own attempts towards one possible implementation.

Introduction

(Gatys, Ecker, and Bethge 2015) introduced a technique for transferring the style of one image onto another, exploiting properties of convolutional networks to extract the information required. Since then, the technique has been refined and expanded upon, with impressive results. A survey paper of the field (Jing et al. 2017), containing references up through March 2018, has over one hundred different papers listed. Because of this interest, it is natural to wonder where this technique could be applied to next. Here, we describe our attempts to apply the techniques underlying style transfer to the domain of 3D point cloud models.

At its core, 2D style transfer works by optimizing a noise vector to minimize a function describing its distance from both the style and content images at different layers within a neural network. In the original paper, the authors utilize VGG-19, a network trained to classify images in the ImageNet data set, and by comparing the

Once the network is chosen, the system's designer picks a particular set of layers of the network, and gets the values of the input images as well as the output image at each of the different layers. From there, the distances between those values are computed and turned into a single loss value, which is used to compute the gradient by which we transform the noise image, and we continue the process. Once a specified number of iterations is completed, the noise vector (which conveniently is chosen to be a N x M x 3 vector) can be interpreted as a N x M bitmap and rendered into a image using standard image libraries.

To implement this for the 3D case, we first explored what style transfer in this domain would mean, given that 3D models carry a lot of information in their positional data. We chose an existing classification network for point clouds—an analog of the network chosen for the 2D case—and explored the different layers to identify which would be the best to use for style transfer. We additionally show the results of our experimentation with different 3D models.

3D style transfer would be useful as a design and ideation tool — by creating models that embody various styles, and a generic model, a designer could use a style transfer system to create versions of the original object in the different styles. These, while not perfect, would give said designer a springboard to create finalized versions of these 3D models. Additionally, having a system like this would also for interesting manipulation of scanned 3D spaces — since laser scanners produce point cloud models, being able to perform style transfer on the results could allow for interesting bespoke spaces.

In this paper, we provide the following contributions:

- An analysis of how to represent style in a 3D space
- An implementation of a style transfer system using a network designed for 3D models
- An analysis of our results and what future work will be required to make a system like this a reality.

Related Work

Deep Learning and Point Clouds

Deep learning systems operate natively over large arrays of floating point numbers, which makes point clouds a natural way of encoding 3D models for neural processing — unlike polygonal models, point clouds exist as a simple list of 3D points in space. Additionally, several common techniques for analyzing real world 3D spaces map those spaces into point clouds, making them an important target to understand and process.

In our work here, we focused on examining PointNet (Qi et al. 2016), a network designed primarily to classify point clouds into several different object classes. Since its publication, it has seen a number of follow-up works, including one extension by the original authors (Qi et al. 2017) as well as others that extended it beyond preselected 3D models (Zhou and Tuzel 2017).

PointNet is not the only system family for neural analysis of point clouds, however. It utilizes and benchmarks itself against the ModelNet dataset (Engelmann et al. 2017) of 40 different classes of point clouds for systems to distinguish between. ModelNet's website[1] lists over thirty different systems and their relative accuracy rankings on their benchmark dataset.

3D style systems

While neural style transfer has not been implemented in this sense, there are a number of other systems that attempt to transfer the style of one 3D object onto another. For instance, (Zheng, Cohen-Or, and Mitra 2013; Lun et al. 2016) focus on handling style by breaking each object apart into individual components and reassembling them based on their structural similarity. In contrast, (Ribeiro et al. 2003) looks at style as a conceptual blend, using an external knowledge base instead of looking purely at the structural similarity to build the comparisons. (Hu et al. 2017) focuses on identifying decorative elements that convey stylistic features across an array of different objects. (Ma et al. 2014) takes an analogy approach to style, starting from a set of example models (one initial model, with one structural variation and one stylistic variation) and then extrapolating those variations into a new model. In contrast, (Kalogerakis et al. 2012) learns a probabilistic model of an object class, allowing it to generate different models that exist within that space, and in that way defines a style of model.

3D styles

When we look at the results from (Gatys, Ecker, and Bethge 2015), we can quickly understand what aspects of an image the algorithm understands as being its style. These aspects include the color palette, the length and shape of the strokes used in painting, as well as some parts of the content - an image generated from van Gogh's Starry Night retains its bright stars in various portions of the sky. The other image - the content image - provides most of the underlying structure and direction of the image itself; the geometry of that image is preserved and we see that image as ”stylized” in the style of the other.

However, for 3D models, the question of how to modify one model to match the style of another is a non-trivial question, especially given that it's unclear what a 3D model is comprised of in general. A 3D model's geometry could be constructed of a number of triangles or a point cloud, to pick two examples. Then, some models feature textures while others don't. With all of these differences in how 3D models are encoded, it is unclear how we would transfer style between two sufficiently different models.

Furthermore, even if we have two similarly structured models, we need to figure out how to transfer styles between the two models. In this paper, our system was implemented to follow (Gatys, Ecker, and Bethge 2015) fairly directly, however this is only one way that the style of a 3D model could be interpreted. Given the nature of 3D models, making adjustments to their structure - moving a few things here or there, adjusting the size of a small part of the model, etc. - can have a large impact in how the model is interpreted by an observer. Additionally, 3D models often have critical, functional parts that impact how they are perceived—an airplane without wings could be nearly impossible to identify as an airplane without some other strong 'airplane-like' features.

Given that our models were point clouds with no texture information, the only changes we could attempt to make were modifying the positions of the different points in 3D space. This limited our possible options for stylistic features to consider during the transfer process, however we came up with several different visions of what 3D style transfer could look like.

First, we have the conceptually simplest version which we call *exemplar style transfer*. In this, the target object is modified to look like the target object, but purely on a cosmetic level without interfering with its functionality. For instance, an umbrella could be blended with a sword by transforming the handle of the umbrella into the hilt of the sword, or a glass mug could be molded to look like any number of existing object. In this way, the target object resembles the style, but maintains all of its own properties.

Secondly, we have the converse of the above, which we call *functional part transfer*. Here, instead of transferring the cosmetic features of the style object, we instead transfer the functional parts of said object. For instance, we could take an skateboard model and add strings and frets from a guitar along its body as our way of blending the two objects. Again, the target object still maintains most of its own identify, but takes on features of the style object to produce the blend.

Next, we have *mix-in style transfer* where the two models are joined together, creating a model that features pieces of both attached together. For instance, we could have the blend of an airplane and an apple where the airplane has a stem coming out of it, or the apple has wings and the tail of the airplane.

Another possibility is the *part-blend style transfer* where the parts of one model are made out of similar parts of the other. In this way, we focus on transferring the local aspects of style, while keeping the broader model's structure fundamentally the same. One example of this would be Lego models — while the boarder structure of the model is kept, the local features of the model now must conform to Lego bricks.

Our final concept for style transfer would be to create an abstract style definition and apply that directly. This is probably the most complex approach to implement, as it requires the system to be able to translate the abstract style information into the transformations that the model should undergo; however, this does allow for the most control over what elements of style are actually transfered onto the new model. This is most similar to the work on Lego models done in (Mazeika and Whitehead 2017).

These are not meant to be conclusive; rather these form a possibility space of how 3D style transfer could occur. For the purposes of aligning with the 2D case, we chose to focus our system on the mix-in style, using the neural networks to find the relationships between the models and their parts.

[1]http://modelnet.cs.princeton.edu

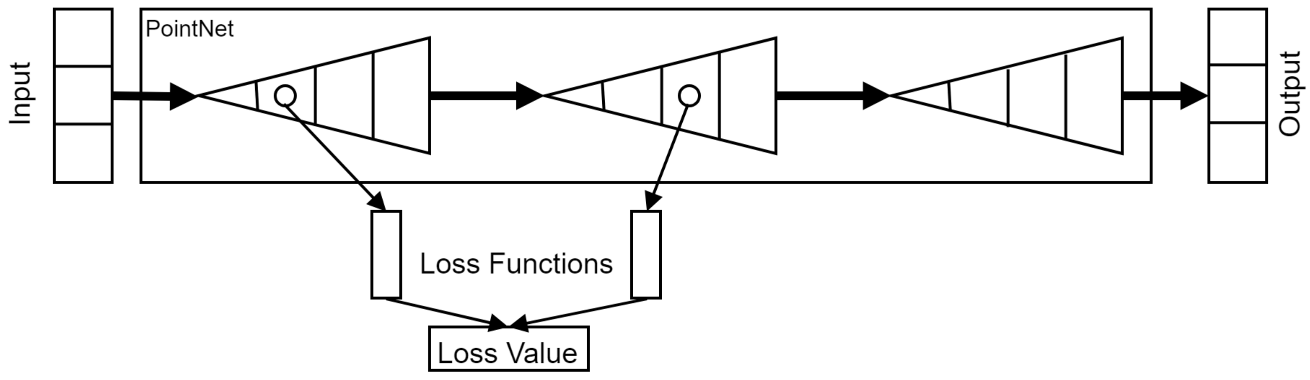

Figure 1: A style transfer system diagram. The center box represents the point net network, which is comprised of several intermediate stages (a more detailed diagram can be found in (Qi et al. 2016)). We select particular layers from the network use to compute our loss function.

Style Transfer Implementation

For our system, we utilized PointNet, an existing network designed to classify 3D models into different categories. In the original style transfer paper, the authors chose VGG-19, a network for classifying 2D images, and so we chose an analogous network for our system. PointNet operates in two different modes — one that classifies point clouds into different classes, and one that learns to segment point cloud, labeling each point of a cloud with a domain specific label (ie the wings versus the body of an airplane, or the wheels versus the handlebars of a motorcycle). PointNet's classification network is structurally similar to VGG-19, comprised of multiple convolutional layers, but it also includes two particular layers: one that learns a three-by-three matrix product (intended to account for rotations in the model) and one that is intended to learn a permutation function (as point cloud data is invariant to the input order).

The segmentation network in PointNet uses the classification network as its basis, and adds on an extra four layers for producing the output labels. This version of PointNet has a major drawback for our purposes: it must be trained on each individual class of model, rather than on all classes at once. While that means that we can't use a trained segmentation network for general style transfer, it makes sense that we could use it as a tool for modeling individual classes of objects.

Once we have a fully trained model, we then need to pick the set of layers to consider for computing the loss function. When a model is evaluated by the network, it considers more abstract representations of the model at each of the subsequent layers. In the original style transfer system, the authors considered earlier layers for the structure and later layers for the style of the various images. Here, we use a similar metric for picking the layers to use for our system, and show the results of exploring this space later.

Loss Functions

Once the layers have been chosen, it then falls to pick a loss function to evaluate how far the generated image is from the inputs on those particular layers. To do this, the system uses the square-mean distance between the layers for the content image and the sum of the difference between the Gram Matrices of the style image and the output.

For our 3D version, we investigated using these loss functions, but also other ones found in our exploration of metrics designed for Point Clouds. To this end, we included the Hausdorff distance and the Chamfer distance as metrics to consider for our loss functions. The Hausdorff distance is defined in our system as

$$d_H(X,Y) = \max\{\max_{x\in X}\min_{y\in Y} d(x,y), \max_{y\in Y}\min_{x\in X} d(x,y)\}$$

where $d(x,y)$ is the euclidean distance between the vectors x and y. Similar, we define the Chamfer distance in our system as

$$d_C(X,Y) = \sum_{x\in X}\min_{y\in Y} d(x,y) + \sum_{y\in Y}\min_{x\in X} d(x,y)$$

using the notation from above. In English, the Hausdorff distance looks at the minimum distance from each vector to any vector in the other set and returns the overall maximum of these values, while the Chamfer distance givens the sum of these minimal distances instead.

Both of these metrics look for outliers within the space — the Hausdorff distance is maximized when a single vector is far away from ones in the other set, while Chamfer looks more at the average distance for all of the vectors in both sets. Importantly, they are also both differentiable, which is a strict requirement for our loss functions.

Finally, one of the key components of any system that hopes to take multiple components into a single value is weighting. Since different functions over different layers can produce values on wildly different orders of magnitude, normalizing the values to both balance the different components against each other, and to provide some bias towards either the structure or the style.

Implementation

While PointNet is publicly published in Tensorflow[2], we chose to implement our system in Keras instead, due to

[2]https://github.com/charlesq34/pointnet

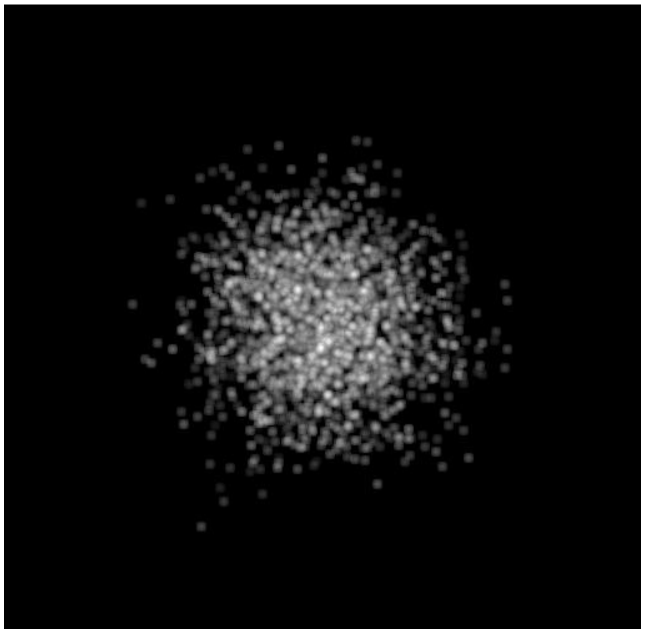

Figure 2: The common result of optimizing on arbitrary layer-function pairs

familiarity. We used an existing Keras implementation[3] as a reference for our implementation, and we used the style transfer code provided in (Chollet 2017) as the starting point and reference for our implementation. Our reimplementation required us to train PointNet ourselves, and we did so using the ModelNet data set provided by (Wu et al. 2015). For the labeled segmentation data, we used the data set provided by (Yi et al. 2016). Our overall accuracy results for training were comparable with the original paper.

Results

Exploration of Layers and Loss Functions

As we began our exploration of style transfer for 3D models, we first began by optimizing our input against a single layer to see how the different layers respond to different loss functions, hoping to identify layers that correspond to stylistic or structural features to optimize for. Our intuition comes from the 2D example, where we can extrapolate what the individual layers have learned by optimizing for them directly, as shown in (Rupprecht 2017).

We considered all of the convolutional layers of the classification model of PointNet and used a fixed random noise input with the Squared Sum, Gram Matrix, Hausdorff and Chamfer loss functions.

However, most of our layer-function pairs lead to fuzzy noise-spheres, as seen in Figure 2. On the other hand, we did have a few pairs that led to interesting results. Some of the lower layers simply produced a twisted version of the original model (see Figure 3). Here, the model is vastly deformed and rotated upside down, which we took as a possible candidate to consider. And, finally, a few layer function pairs simply reproduced the initial model instead, with some small variations due to the fuzziness of the optimization process.

[3]https://github.com/garyloveavocado/pointnet-keras

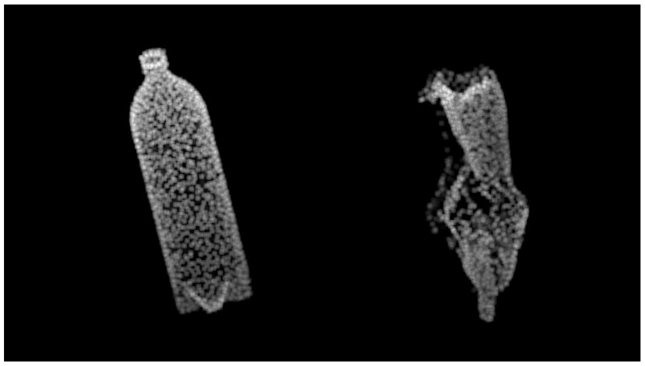

Figure 3: A model and its twisted counterpart
(3rd convolutional layer using the Gram loss function)

This was disappointing to see; while we could reproduce the initial model in the very early layers, we had hoped to see the noise clouds showing abstract features of the particular model class. This may have resulted from our choice of network — since PointNet attempts to learn how models are structured, regardless of how the points are arranged and permuted, it could be the case that the abstract features are being represented in a way that in imperceptible to humans. In the 2D case, we're able to see patterns and variations between different images, as those appear as variations in color, but the 3D case solely considers the positions of the different points, meaning that there might be relations that are not actually the ones we want to express being learned.

Classification Model Results

With our results from the previous exploratory work, we attempted to blend models with one of the layer-loss pairs that simply reproduced the initial model; specifically the Chamfer loss on the third convolutional layer of the classification model. One of the key features of style transfer is balancing the different loss values against each other — most of the 2D systems feature a weighing system in which the different sides of the function (the similarity to the style and the similarity to the initial content) are balanced against each other to get the desired blend.

To this end, we took two models, and blended them at different ratios between the different sides — we fixed the content weight at 1, and scaled the style weights through different powers of 2. We used the third convolutional layer with Chamfer loss for the content and Gram loss for the style as our basis for our loss function. While the Chamfer loss reproduced the original model at that layer, the Gram loss created a twisted version of the model, as seen in Figure 3. Here, we hypothesized that the twistedness was due to having a slightly more abstract understanding of the original, and that we would see this conveyed in the style transfer process.

The results of blending an airplane model and a model of

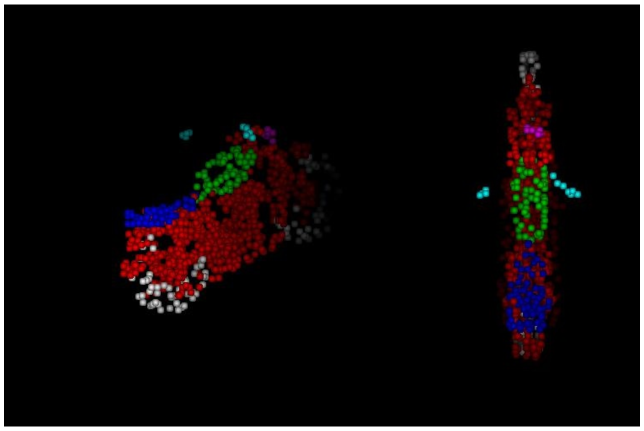

Figure 4: A ground-truth segmentation for a motorcycle model

a woman in a dress are shown in Figure 6. At the extreme values (2^{15} and 2^{-4}), our system effectively optimizes for one model or the other, as the loss value is dominated by the output's distance from that model. In the middle, we see blends in of the two models; however, this occurs merely on the level of the points' actual distances from each other — no abstract qualities are carried between the two models.

One of the clear qualities here is that the blends are contained in, effectively, the intersection in space between the two models. This suggests that one of the big issues here is orientation, since the airplane lays flat while the person is standing upright. Additionally, using other layers turns the output into a fuzzball, so this proved to a dead end.

Segmentation Model Results

Finally, we attempted to utilize the segmentation version of PointNet to transfer the underlying model class's style onto an unrelated model. To do so, we first trained the network to perform segmentation on airplanes. We then took a motorcycle model (seen in Figure 4) and assigned each one of its labels a particular label from the airplane label set (i.e., the body of the motorcycle corresponded with the body of the airplane; the wheels corresponded with the engines, etc.). From here, we built our loss function to optimize for the original structure of the motorcycle against each point receiving its correct label under the classification system. The intent here was to create a new motorcycle such that each one of its components was interpreted as part of an airplane.

Unfortunately, again, the results were suboptimal. We again tested various weights of style and content, as seen in Figure 5, but optimizing for style merely leads to the model expanding and not, as hoped, being reshaped.

Discussion

Since our experiments produced negative results, the question then becomes "why?" — what led to our results, and what can we learn from these experiments? Fundamentally, there are two high-level cases for these failures: either the

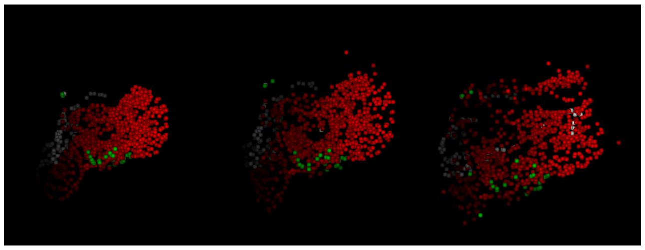

Figure 5: The results of segmentation style transfer on the motorcycle model with different weights

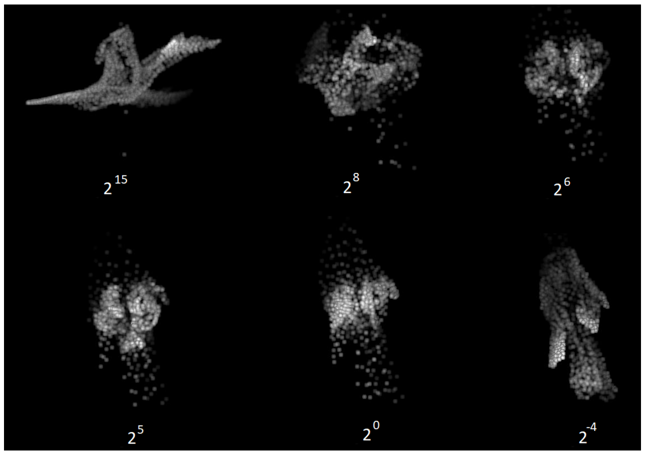

Figure 6: Two models (airplane and woman in a dress) blended at different ratios of style weight and content weight. Labeled values are the style weight compared to a content weight of 1.

system had some errors of design, or there are theoretic factors that prevent this approach from working at all.

Fundamentally, style transfer in the 2D domain relies on being able to detect the edges (for the content image) and the color palette (for the style image) of an image in a way that can be optimized for. Doing so requires the system to examine relationships between spatially related pixels, and then shift a noise vector until it grows closer and closer to these features. This works well for images, since the images are represented as a 2D spatial matrix (with a third dimension representing color values). On the other hand, the point clouds are merely a list of different points in space. While PointNet itself attempts to compensate for the input order (by learning a reordering function about halfway through the system), this may cause the issues with the loss function.

Additionally, one of the common stylistic features of the 2D style transfer images (independent of the style image itself) is a certain amount of fuzziness in the output—edges often have a certain amount of fuzziness to them, a result of the optimization process. But, because of the overall shape of the output and the color patches included, humans are still able to recognize the contents of the image without much issue. However, in our 3D case there is no color channel that we can rely on for context. As such, the only information we have to work with in the system are the locations of the different points. Because of this, the results are highly sensitive to points being moved around, which means that the fuzziness that results from the style transfer can lead to results that are impossible for humans to interpret correctly.

For neural style transfer to be possible for 3D point clouds, several adjustments would need to be made. First of all, a different neural network would likely need to be considered. One of the key aspects of PointNet is that the network tries to learn a permutation function that allows it to detect the models correctly regardless of the order their points are in. This factor may be part of the issue we have in producing visible results, and other networks on point clouds may feature clearer results.

Secondly, a different set of loss functions would need to be considered. While we tried to include relevant loss functions to point clouds in general, it might be the case that others exist that would work better with the particular network or point clouds in general, and other functions would have produced intelligible results. We chose our functions based on metrics that were used previously in style transfer and known functions for examining point clouds, and ran an exhaustive search over.

Finally, in this work, we only considered one layer at a time — most of the existing work on style transfer looks at multiple layers at once and averages the loss between all of them. This provides more consistent results and the generated images benefit from these multilayer views. However, in the 2D case, it is clear what sorts of features are captured by the different layers of VGG-19; our attempts to visualize the PointNet network were inconclusive. As we ran an exhaustive set of experiments over the layer-loss pairs, it is unlikely that we missed any layer that would drastically change our results, and reduplicating layers is the equivalent of doubling the value of the weight.

However, the deeper issue remains of what style even means for point clouds. When we look at style transfer for images, we see clearly what aspects of style are picked up by the system—colors, line curviness, patterns, etc—and how those are applied to the content image. With 3D style, it is unclear what features a neural network would pick up on—would it key into the shapes of the different parts, the different relative positions of things, the orientation of the model itself, or the patterns of points within the model itself?

Conclusions and Future Work

In this paper, we attempted to apply the neural style transfer techniques that have seen so much success in the domain of 2D images to the domain of 3D point clouds. Despite this, our system was unable to perform style transfer as is seen in the 2D style transfer systems. While negative results do not necessarily provide conclusive evidence, our exhaustive exploration provides a strong argument against this being possible.

However, there is room to explore style transfer within point cloud models. One idea would be to use a segmentation model (such as PointNet) to design a parts-based style transfer system, similar to (Lun et al. 2016). Additionally, exploring other neural networks and other ways of encoding style in a 3D space could provide interesting results as well.

References

Chollet, F. 2017. *Deep learning with python*. Manning Publications Co.

Engelmann, F.; Kontogianni, T.; Hermans, A.; and Leibe, B. 2017. Exploring spatial context for 3d semantic segmentation of point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 716–724.

Gatys, L. A.; Ecker, A. S.; and Bethge, M. 2015. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*.

Hu, R.; Li, W.; Kaick, O. V.; Huang, H.; Averkiou, M.; Cohen-Or, D.; and Zhang, H. 2017. Co-locating style-defining elements on 3d shapes. *ACM Transactions on Graphics (TOG)* 36(3):33.

Jing, Y.; Yang, Y.; Feng, Z.; Ye, J.; Yu, Y.; and Song, M. 2017. Neural style transfer: A review. *arXiv preprint arXiv:1705.04058*.

Kalogerakis, E.; Chaudhuri, S.; Koller, D.; and Koltun, V. 2012. A probabilistic model for component-based shape synthesis. *ACM Transactions on Graphics (TOG)* 31(4):55.

Lun, Z.; Kalogerakis, E.; Wang, R.; and Sheffer, A. 2016. Functionality preserving shape style transfer. *ACM Transactions on Graphics (TOG)* 35(6):209.

Ma, C.; Huang, H.; Sheffer, A.; Kalogerakis, E.; and Wang, R. 2014. Analogy-driven 3d style transfer. In *Computer Graphics Forum*, volume 33, 175–184. Wiley Online Library.

Mazeika, J., and Whitehead, J. 2017. Solving for bespoke game assets: Applying style to 3d generative artifacts.

Qi, C. R.; Su, H.; Mo, K.; and Guibas, L. J. 2016. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593*.

Qi, C. R.; Yi, L.; Su, H.; and Guibas, L. J. 2017. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*.

Ribeiro, P.; Pereira, F. C.; Marques, B. F.; Leitão, B.; Cardoso, A.; Polo, I.; and de Marrocos, P. 2003. A model for creativity in creature generation. In *GAME-ON*, 175.

Rupprecht, P. 2017. Understanding style transfer. `https://ptrrupprecht.wordpress.com/2017/12/05/understanding-style-transfer/`.

Wu, Z.; Song, S.; Khosla, A.; Yu, F.; Zhang, L.; Tang, X.; and Xiao, J. 2015. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1912–1920.

Yi, L.; Kim, V. G.; Ceylan, D.; Shen, I.-C.; Yan, M.; Su, H.; Lu, C.; Huang, Q.; Sheffer, A.; and Guibas, L. 2016. A scalable active framework for region annotation in 3d shape collections. *SIGGRAPH Asia*.

Zheng, Y.; Cohen-Or, D.; and Mitra, N. J. 2013. Smart variations: Functional substructures for part compatibility. In *Computer Graphics Forum*, volume 32, 195–204. Wiley Online Library.

Zhou, Y., and Tuzel, O. 2017. Voxelnet: End-to-end learning for point cloud based 3d object detection. *arXiv preprint arXiv:1711.06396*.