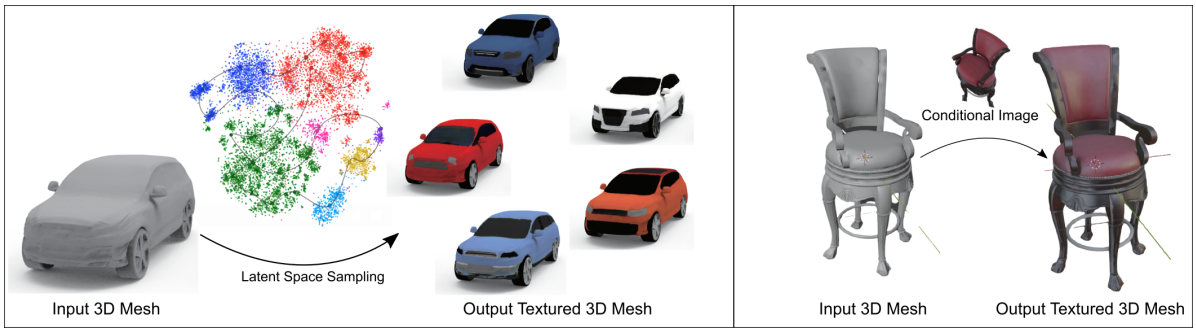
VJAM Mesh Texture: Learning continuous texture representation for conditional and unconditional texture synthesis of 3D meshes

Vikas Thamizharasan ***J***oshua Pierce ***A***ngelina Grosso ***M***oyi Tian



Abstract

We endeavor to design an architecture that can learn detailed textures across a latent space of images and shapes, thus capable of probabilistically generating realistic, novel textures for previously unseen meshes. Taking inspiration from the recent work TextureFields by Oechsle et al. [10] and others, we explore varying methods of shape-representations, alternative loss metrics, and high-frequency learning techniques. We demonstrate that applying Fourier feature transformations through a positional encoding is an effective means for learning more detailed textures.

1. Introduction

Generative models over the past few years have exploded in popularity and become a hallmark of research problems within the vision community since the seminal work of Goodfellow et al. [4] and Jürgen Schmidhuber [13]. Deep learning models have become a cornerstone of image-based tasks such as image-to-image translation. However adapting these models to 3D data is not trivial as naive implementations can grow in space and time complexity. Unlike images which are regular, 3D data have numerous representations, each with their pros and cons. Fortunately over the past couple of years, there have been numerous works designing learning-based techniques for 3D geometry representations and that take into consideration the trade-offs across fidelity, efficiency and generalizability. Some of these works include PointNet[12], AtlasNet[5], DeepSDF[11], Occupancy Net[8] and MeshCNN[7]. They have shown promising results in various tasks such as single image 3D reconstruction, surface reconstruction from partial and noisy 3D data, interpolation, generative modeling and learning high quality shape representation.

Learning texture representations has been a less explored problem. Like in the case of geometry, how we represent our input textures is not obvious. This is important, not just for deciding what kind of data should be collected for training, but also informing the design of the architecture and the representations learned by the neural network. If we attempt to leverage the regular nature of voxels data, we run into issues such as: the discretization of voxels presents inherent limits in expressing high frequency, sharp details; furthermore, voxels are very memory inefficient, increasing by a cubic factor. Some of the above problems can be resolved by using techniques such as texture atlas, which provides a mapping between 3D data and UV space (2D), or using Geometry Images [6]. While these representations alleviate some of the concerns of using voxels, they come with substantial distortion associated with orientation as well as discontinuities at boundaries. Furthermore, convolutions on such maps would make learning canonical features difficult due to having irregular and randomly distorted receptive fields.

With this motivation, our Deep Learning Project builds from a recent work called TextureFields by Oechsle et al. [10]. TextureFields seeks to solve the aforemen-

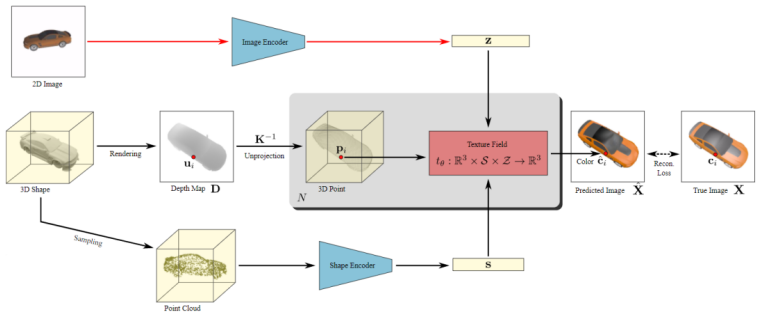


Figure 1. TextureFields's Conditional Model. Source : [10]

tioned problems by using neural networks to approximate a parameterized continuous function for representing texture information in 3D space. In this report, we first review the framework proposed in TextureFields and subsequently explore some ideas to improve certain aspects of this paradigm. Ultimately, we seek to train a network that produces more meaningful latent representations for appearances and shapes, and accordingly generating more realistic textures.

In our project, we attempt to incorporate:

- More state-of-the-art techniques for learning shape representations (like DeepSDF, OccupancyNets, PointNet++ [12]) by either replacing the TextureFields shape encoder entirely, or by initializing the shape encoder weights with a PointNet encoder pre-trained on a geometry-specific task with stronger priors.

- Methods for approximating high-frequency data, such as mapping inputs to Fourier features via a 'positional encoding' [15]. Ideally, this would afford TextureFields a more detailed expression of textures from the latent space.

- Alternative loss functions such as the earned Perceptual Image Patch Similarity [16]. The goal would be to clarify blurry textures generated with L1 loss, which is more sensitive to large perceptual changes

- Additional experiments with ShapeNet [2] and the 3D Future dataset [3], with corresponding pre-processing scripts.

2. Background

2.1. TextureFields

The authors define a function t parameterized by a neural network t_θ, with learnable parameters θ, which maps an input 3D point p conditioned on shape embedding $s \in S$ (calculated from the input geometry) and latent variable $z \in Z$ (derived from either a conditioned image or sampled from a learned distribution), to produce an output colour c:

$$t_\theta : \mathbb{R}^3 \times S \times Z \longrightarrow \mathbb{R}^3 \tag{1}$$

As the idea of texturing a mesh is ill-posed, the authors choose to condition TextureFields with a shape embedding s: generated from a shape encoder that ideally captures contextual geometric information about the input shape; and also a viewpoint invariant global-feature representation z: encoded from an image that constraints this task by providing information about the mesh appearance.

TextureFields, built around a shape encoder and the actual texture fields architecture, implements a total of three different models: a conditional model, a generative adversarial network and a variational auto-encoder to capture ambiguity in a random latent code z. In this work we focus on the conditional model which is illustrated in figure [1]. This model takes as input both a mesh (with points and surface normals uniformly sampled) and a conditional 2D image, and outputs a textured mesh modelled after the conditioned image. The conditional model is composed of a shape encoder that generates the shape embedding, an image encoder that generates the latent variable and the TextureField model itself.

2.2. Occupany Nets

Extracting shape-information is critical to the task of TextureField. Occupany Networks implicitly represent the 3D surface of an object as the continuous decision boundary of a deep neural network classifier. In later sections we use a pretrained occupancy net to initialize the shape embedding of texturefields.

2.3. PointNet++

Pointnet is another method for extracting shape information: specifically, offering a permutation-invariant approach to analyzing point cloud data and thus allowing one to directly extract geometric information with deep learning

Figure 2. Samples from the chair class of 3D FUTURE.

Figure 3. Samples from the car class of ShapeNet.

techniques. Pointnet++ offers notable improvements over its predecessor by introducing a method for applying convolutions to the point cloud data. While Pointnet relied on aggregating individual features of points, Pointnet++ also incorporates the local structures, facilitating markedly better shape representations.

2.4. Losses

TextureFields conditional model uses ℓ_1-loss between the predicted and rendered images to train the model. Such setting is commonly used in network training, but it cannot efficiently capture the perceptual similarity in image features. In recent years, a network-trained loss function, perceptual losses, is proposed as a measure more-aligned with human perceptual judgement by better capturing low-level perceptual similarity across the latent features of images. Studies have shown that perceptual losses excel in further fine-tuning the learned features of a pre-trained network.

2.5. Positional Encoding

Recent research has made notable strides in enhancing neural networks' ability to approximate high-frequency data, such as Sitzmann's Sinusoidal Representation Networks[14] and Tancik's Fourier Feature Networks[15]. One notable application of the latter method was NeRF's[9] learning of photo-realistic, 3D scene representations. With this as inspiration, we explore whether these techniques could afford TextureFields a more granular, detailed expression of textures from its learned latent space.

3. Methodology

3.1. Data

We use data from the ShapeNet dataset [2] and the 3D FUTURE Dataset [3]. ShapeNet is a popular large scale 3D repository covering various object classes. In this work, we use the car category along with the textures. 3D FUTURE was developed by professional designers and contains high quality 3D instances of furniture with high resolution textures. This dataset contains both synthetic scenes and 3D instances of furniture, but for our purposes we will only be dealing with 3D instances of furniture, of which there are 9,992 instances across a total of 34 different categories. These 34 categories can be divided into broad categories, such as "bed", "cabinet" or "chair". Looking at figure [2] and [3], it can be seen that 3D FUTURE contains objects with more complex topology and higher resolution textures containing fine details.

TextureFields requires as input: uniformly sampled points of the 3d mesh and the corresponding surface normals at each sampled point, N depth maps and albedo images rendered from various camera angles along the viewing hemisphere, the camera intrinsic and camera extrinsic for each of the N camera poses in order to unproject the depth pixel to a 3d point, and optionally a set of conditional images of objects similar to the input mesh. TextureFields provides us with the pre-processed data for ShapeNet, but not the corresponding script that accomplishes this. We thus write our own script to be able to generate data for 3D FUTURE that adheres to the requirments of TextureFields.

3.2. Adjusting the Shape Encoding Architecture

As 2D image encoders are a well-studied field, we attempted to improve the performance of the conditional model by modifying the shape encoder.

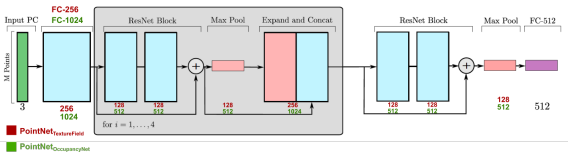


Figure 4. Shape encoder used by TextureFields, Source : [10]

We hypothesized that the shape embedding outputted by the shape encoder should ideally contain pertinent topological information about the mesh, such as local connectivity, symmetry, and part decomposition. We conjecture that having a latent vector which is more topologically aware would benefit the overall output of the model. The shape encoder originally implemented by the authors was inspired by Pointnet and uses ResNet blocks; it takes point clouds as

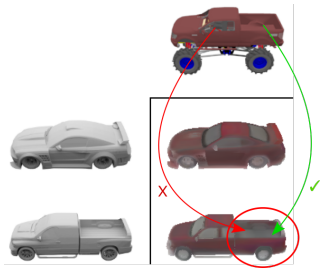


Figure 5. An example of incorrect texture prediction. While the example of the input car looks correct, the second example of the pickup truck has an instance of incorrect texture inferred from the conditioned image. Ideally, the shape embedding coupled with the encoded image representation should inform TextureFields of semantic-level information such as windows, chassis, trunk etc. But here we see texture for the trunk being drawn from the windows as opposed to the trunk visible in the conditioned image.

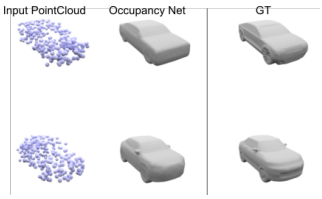


Figure 6. Occupany net trained on surface reconstruction.

inputs and outputs a 512 dimensional vector. This PointNet encoder was trained from scratch within the pipeline shown in figure [1], where all the losses are only computed on the output of the TextureFields model. This leads to some uncertainty as to what degree the shape encoder encodes meaningful information as described above into the low dimensional shape embedding. Some of the results shown in figure [5], makes us believe there is room for improvement.

We propose some of the following changes:

- Initialize the PointNet encoder with pre-trained weights. The idea behind this is to use weights from a model that was learned for a 3D-geometry task such as reconstructing surfaces from pointclouds. With such a task, the network is able to better utilize priors and is penalized for not capturing pertinent topological information.

  This goal is never fully-realized in TextureFields and thus motivates us to employ pre-trained weights, and then subsequently fine tune the PointNet encoder within TextureFields, thus giving the network a head start and hopefully helping it converge to an optimal minima.

- We replaced this shape encoder by a Pointnet++ encoder.

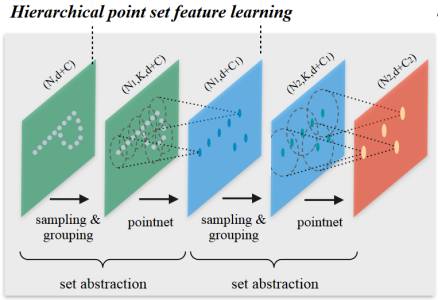


Figure 7.

Training and Evaluation

Ideally, the changes we made to the TextureFields architecture would be evaluated by training the model multiple times on a large dataset. One training run would use the original TextureFields architecture, while the other training runs would incorporate our various modifications. This experimental setup would allow for results that could clearly evaluate the impact of our modifications on performance. After reaching out to the authors of TextureFields, we learned that training their model on the ShapeNet car dataset took approximately one week. As this was an infeasible task for our project timeline, we modified our original training and evaluation plan.

Instead of training on a large dataset, we decided to overfit on single examples. Using a single example as the input into the model, we trained the model until the loss value appeared to converge and the output did not appear to change between epochs. We then used a mixture of qualitative and quantitative criteria to evaluate the performances of the different architecture. We essentially test the representation power of the network.

Similarly, to compare the performances of the standard ℓ_1-loss and the perceptual losses, we perform the representation power experiments on both ℓ_1-loss and perceptual loss [16]. Specifically, the perceptual loss function we use is based on the VGG network, initialized from a pre-trained classification model.

3.3. Positional Encoding

Recent research has proposed effective techniques for enabling neural networks to approximate high-frequency data, such as SIREN[14] and Fourier feature mapping[15]. The latter uses a positional encoding technique: specifically, mapping the inputs of a neural network to higher-dimensional Fourier features. This technique has presented demonstrable improvements in learning high-frequency inputs such as in the case of NeRF's photorealistic, three-dimensional scenes[9]. Given the conceptual similarity between TextureFields and NeRF's learned mapping of colors

in a 3D space, we chose to implement NeRF's positional encoding method. Furthermore, this approach is limited to mapping the input coordinates to higher dimensions, thus allowing us to avoid substantially modifying the TextureFields architecture.

Recall that TextureFields takes as input a shape-vector, image-vector, and a 3D coordinate on the provided mesh, and outputs a corresponding color. Following the implementation in NeRF, our positional encoding technique maps each dimension of the 3D coordinate into higher dimensional space with a Fourier features transformation.

$$\begin{aligned}\gamma(v) = (&a_0 sin(\pi b_0^T v), a_0 cos(\pi b_0^T v),\\ &a_1 sin(\pi b_1^T v), a_1 cos(\pi b_1^T v), ...,\\ &a_L sin(\pi b_L^T v), a_L cos(\pi b_L^T v))\end{aligned} \tag{2}$$

Note that each input coordinate is mapped to multiple pairs of sin and cos transformations- the number of such pairs is represented by L in the above equation. Therefore, the resulting transformation maps the 3D input coordinates to $3 * L * 2$ dimensions. L is a hyperparameter, and from the applications outlined in NeRF[9] and Fourier Features Networks[**?**] we chose $L = 8$, thus mapping TextureFields' 3D input to a 48D vector.

Training and Evaluating

As noted above, fully training an updated TextureFields model requires a prohibitive amount of training time. Furthermore, in this case it's not sufficient to overfit to a single texture because we want to evaluate both the positional encoding's effect on learning detailed textures from the training set as well as the impact on synthesizing novel textures from new shape- and image-vectors. In balancing these goals we chose to train a TextureFields model modified with a positional encoding for 30k epochs on five car models. For comparison, we also trained the original TextureFields model for 29.5k epochs on the same five cars. This required 10.5 hrs and 7.5 hrs of training on a single NVIDIA Tesla V100 GPU for the normal and positional encoding models, respectively. Comparison of the two models consists of a qualitative assessment of their generated images as well as an evaluation of image similarity metrics calculated from predicted views and their corresponding ground truths. Similar to TextureFields, we use the structure similarity image metric (SSIM) as a measure of local image-properties and their Feature-ℓ_1-metric as a more global perceptual measure. Generated images and corresponding image similarity metrics are calculated at every 500 epochs.

4. Results

4.1. Shape Encoders

In order to evaluate the performance of our modifications to the shape encoder architecture, we overfit the model on a single example from the shape net dataset. We ran the model a total of three time: once as a baseline, once with Pointnet++ used as the shape encoder, and once with the original shape encoder pre-trained with weights from Occupancy Network. Quantitatively, we observe the same two model evaluation metrics use by Texture Fields: structure similarity image metric (SSIM) and feature ℓ_l-metric.

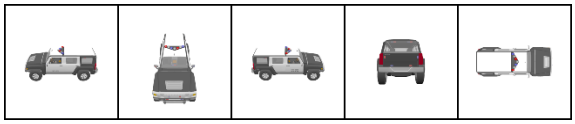


Figure 8. Ground-truth images for the training

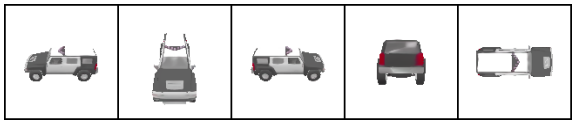


Figure 9. Baseline predicted images

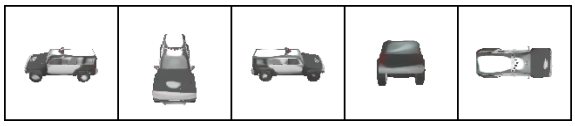


Figure 10. Pointnet++ predicted images

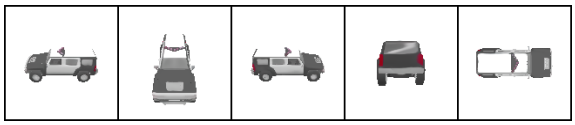


Figure 11. Pretrained Weights predicted images

For the baseline, we used the original architecture provided by Texture Fields and ran for a total of approximately 70,000 epochs. Qualitatively, the final result is very close to the ground truth which is to be expected when overfitting to a single model. Some small details remain unlearned, such as the ground truth fact that the seats can be observed through the rear window of the car.

We then ran the model again, this time for approximately 85,000 epochs, with Pointnet++ as the shape encoder. We saw that qualitatively, the final result was further from the ground truth than the baseline. Many details remained

blurred or smudged, particularly on the roof of the car. As we ran this model for more epochs we received worse results and concluded that changing the architecture to Pointnet++ hurt the performance of the model.

The third model used the original architecture provided by TextureFields with the shape encoder initialized with pre-trained weights from the Occupancy Network. We originally tried to initialize all weights with pre-trained weights. This yielded a strange output of monochrome texture. At first we believed there was a bug within the code we used to load the pre-trained weights. Extensive debugging showed that when different combinations of weights were initialized, some would stay frozen at certain values while others would change throughout training. We came to the conclusion that there was nothing wrong with our implementation. Our theory for this strange behavior is that the pre-trained weights from Occupany Network cause the conditional model to remain stuck in a very non-optimal minima.

To avoid getting stuck in such a minima, we decided to only initialize some of the weights within the geometry encoder. Specifically, all the weights within the first 2 ResNet blocks. Qualitatively, the results were very similar to those from the baseline. The hypothesis here is that, the first few layers learn features that are common between the two tasks: texture synthesis and surface reconstruction.

Comparing the quantitative metrics, there is no significant difference between the (SSIM) and (FID) of the three runs.

Altogether, these results led us to tentatively conclude that changing the geometry encoder to Pointnet++ harmed the model's performance while initializing some weights within the geometry encoder had no effect.

4.2. ℓ_1-loss vs. Perceptual Loss

We trained on one car model from ShapeNet dataset once with ℓ_1-loss proposed by TextureFields and once with perceptual loss function. Each of the trainings ran over approximately 29k epochs. Figure [25] shows the real images from the actual car model. Figure [13] shows the predicted images with respect to ℓ_1-loss and Figure [14] shows the predicted images with respect to the perceptual loss.

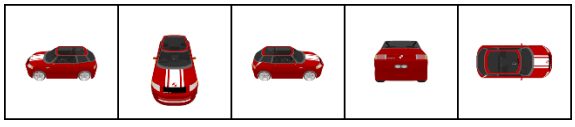


Figure 12. Ground-truth images for the training

From the generated fake images, we can qualitatively conclude that the TextureFields network is unlikely to improve performance by implementing the perceptual loss. Specifically, we observe that some types of patterns in appearance, the white stripes on the hood for example, can

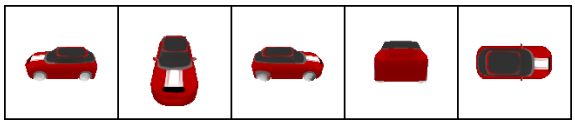


Figure 13. Predicted images after training 29k epochs with ℓ_1-loss

Figure 14. Predicted images after training 29k epochs with perceptual loss

no longer be effectively captured after taking the perceptual judgement part into the network. We also found that the predicted color tends to be brighter in the perceptual network. One thing to notice is that the tail lights of the car, which have long thin horizontal shapes, are better depicted in the perceptual case than the ℓ_1-loss case. We conjecture that the perceptual loss is better at learning textures with patterns that are thin and long horizontally than the ones that are more like vertical strips.

Quantitatively, we can look at the graphs of average loss per epoch over training as depicted by Figure [15] and compare the behaviors of the two networks. We observe that perceptual losses result in larger variations and also are higher than the ℓ_1-loss on average. Hence, we can also conclude from the data that ℓ_1-loss has better performance than the perceptual losses in the task of learning appearance.

In general, the behavior of the network implemented with the perceptual loss function does not match our prediction that the network will be able to learn the texture information better if we simply improve perceptual similarities by capturing deep features on the image level. Since the network with perceptual losses takes approximately double the time as the ℓ_1-loss to train for the same number of epochs, in practice applying perceptual losses for capturing 3D texture is not a practical choice. However, the differences in effectively captured features suggests a new potential direction to investigate. It is worth studying which features are more advantageously learned by these networks, and how they may correspond with specific loss functions. This may provide a strong basis for strategically combining multiple losses to improve performance.

4.3. Positional Encoding

When qualitatively comparing the abilities of both models' to learn detailed representations of textures in the training set, the TextureFields model with the positional encoding transformation (*PE* learned more granular textures as compared to the original TextureFields model (*ORIG*). Although the models improved their texture representations

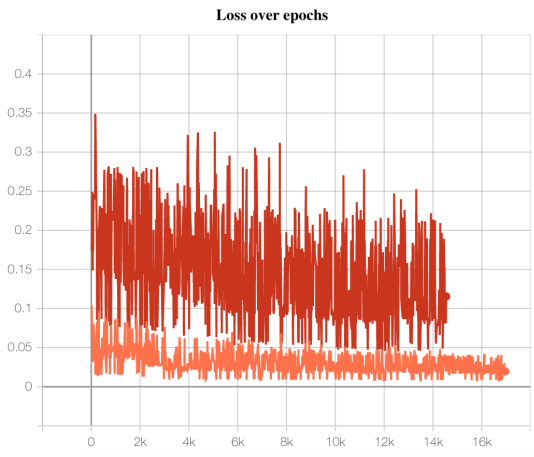


Figure 15. Average loss per epoch (collected from the initial part of the training). Perceptual loss is in red and ℓ_1-loss is in orange

over the course of training, both models occasionally generate abruptly distorted textures for one or more car models before adjusting in the subsequent epochs. (For examples see figure 24 in the appendix: epochs 24.5k and 29.5k for *PE* and *ORIG*, respectively.) The last epoch for *ORIG* generated substantially distorted textures for three of the car models. To avoid exaggerating the degree of *PE*'s improved accuracy relative to *ORIG*, in figure 16 we compare the textures generated from *PE*'s last epoch with the qualitatively 'best' images selected from across *ORIG*'s training history. Even with these *ORIG*-favorable terms, *PE* generates comparatively more detailed textures. For further context, figure 22 in the appendix compares the generated textures from the last epoch of both models for all five cars in the training set.

Additionally figures 23-25 in the appendix show the progression of generated images at an interval of 2.5k epochs. (These generated images are the ones from which *ORIG*'s 'best' results were selected in figure 16.) Figures 23-25 also highlight a distinction between the learning processes of *ORIG* and *PE*. In broad terms, details appeared to be learned more rapidly for the *PE* model. Specifically, note that even the initial texture representations of *PE* include some variability in allocating color, while initial *ORIG* representations are largely monochrome, reflecting the most prevalent color of each car. This comports with the functional purpose of positional encoding- ie, *PE* more readily learns to allocate high-frequency changes in color-values across relatively short distances.

The comparative speed with which *PE* approximates textures in the training set is evident when evaluating the training loss. See figure 17. The *PE* model also shows better SSIM and Feature-ℓ_1-metric values over the course of the 500-epoch reporting intervals, indicating higher-fidelity

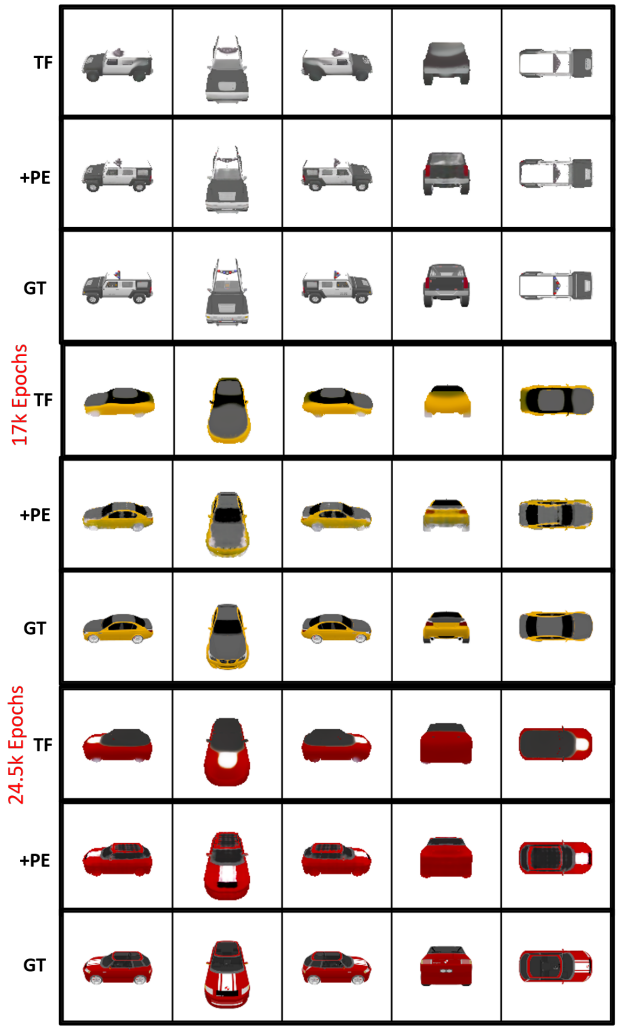


Figure 16. **Comparison of textures generated from three of the five cars in the training set.** The model with positional encoding (*+PE*) learned more detailed representations of the training textures. For example, note the red border around the roof of the bottom car; and on extremely close inspection, the white stripe on the hood can be seen to be segmented into three sections similar to the ground truth (*GT*). This model was trained for 29.5k epochs. The normal TextureFields model (*TF*) happened to generate substantially distorted textures for the red and yellow cars on the final 29.5k epoch. (This is a normal periodic occurrence for both models over the course of training- for reference the generated textures across training is visible in figures 23-25 in the appendix.) So as to not over-emphasize the difference between the two models, the above figure selects the best texture over the course of training for the TextureFields model, with the corresponding epoch indicated in red text. (Unless otherwise indicated, the 29.5k epoch is used in the above figure.) A comparison of generated textures from the 29.5k epoch for both models is visible in figure 22 of the appendix

representations of the training textures. See figures 18 and 19.

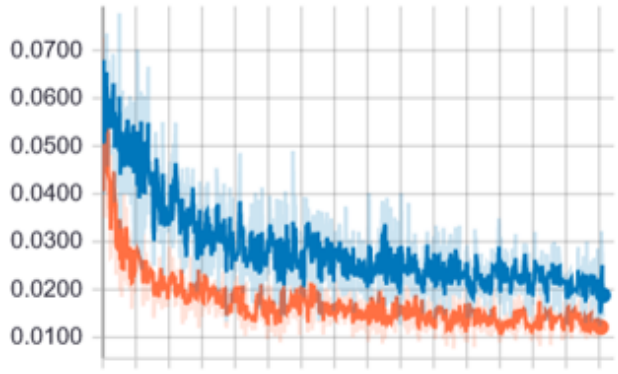


Figure 17. **Training loss over the course of 29.5k epochs**. The blue line represents the normal TextureFields model (*ORIG*) while orange indicates the model with positional encoding (*PE*). Note the steeper initial drop in loss for the *PE* model, as well as the diminished variation across epochs. The steeper slope in the loss curve comports with the comparatively earlier appearance of texture details during *PE*'s training (see figures 23-25 in the appendix)

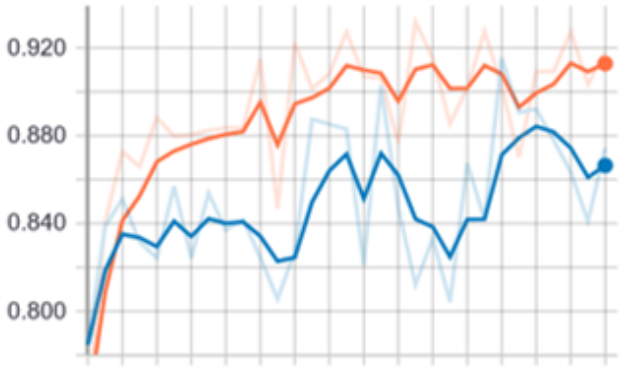


Figure 18. **Structure similarity image metric (SSIM) calculated at 500-epoch intervals over the course of 29.5k epochs.** The blue line represents the normal TextureFields model (*ORIG*) while orange indicates the model with positional encoding (*PE*). Higher values indicate more image similarity between between the model's predicted images and the ground truth for cars in the training set. As compared to the metric reported in figure 19, SSIM is a measure of local image-properties.

While *PE* performs better at learning textures from the training set, that does not necessarily equate to improved performance when synthesizing novel textures. Figures 20 and 21 show two examples where *PE* and *ORIG* generate novel textures after being given a new mesh and corresponding shape and image-vectors as inputs. Given the limited training data and training time this analysis is only useful for the sake of comparison between the two models; in other words, we wouldn't expect either to perform well with this task. In the first example, *ORIG* does a notably better job

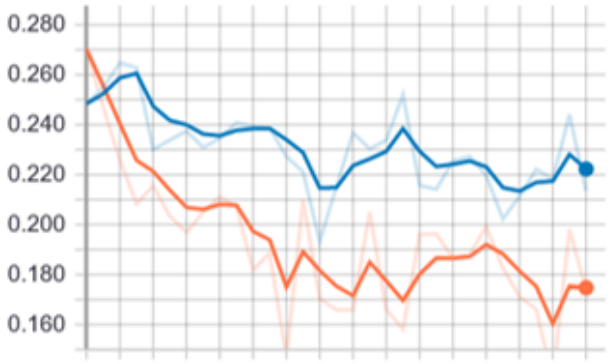


Figure 19. **Feature-ℓ_1-metric calculated at 500-epoch intervals over the course of 29.5k epochs.** The blue line represents the normal TextureFields model (*ORIG*) while orange indicates the model with positional encoding (*PE*). This is a metric that was introduced by TextureFields, where higher values indicate *less* image similarity between between the model's predicted images and ground truth for cars in the training set. This metric was designed to capture more global features of image similarity.

of incorporating the image-vector into the texture, however *ORIG* completely misses the mark with the second example. While *PE* displays more texture-variance with its mottled allocation of colors, this does not clearly correspond to generating more realistic textures.

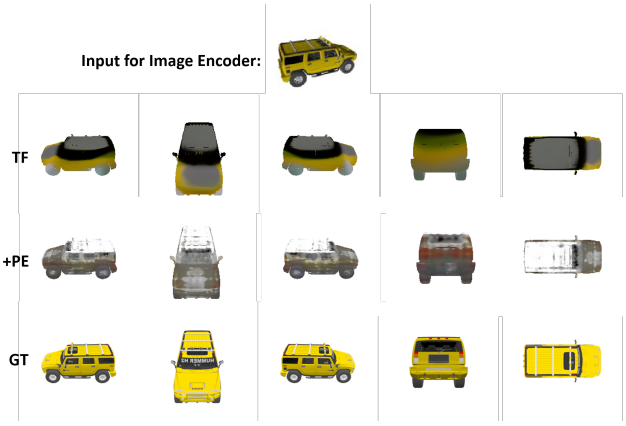


Figure 20. **Comparison of novel textures generated after training for 29.5k epochs on five car models.** The models are given as input the shape- and image-encodings from a previously unseen car, and then predict a corresponding texture. The image-encoding is calculated from a single view of the car, which is displayed at the top of the figure. As expected, neither model did particularly well at this task due to limited training; however, *ORIG* better approximated the color of the target texture, seeming to appropriately lift the coloring-scheme of the yellow-car from the training set. On extremely close inspection, it's apparent both models made slight attempts at the lettering on the windshield of the truck, with the *PE* model's version slightly more pronounced.

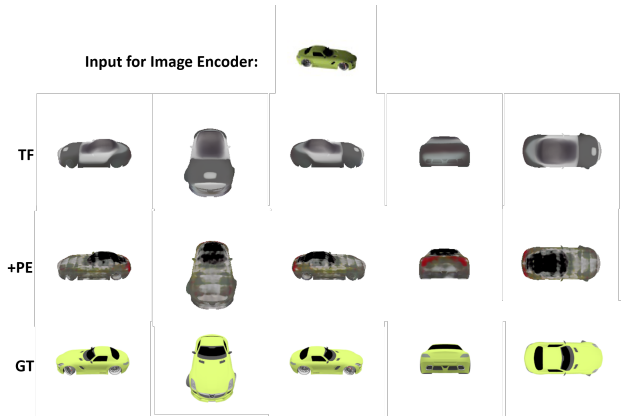


Figure 21. **Comparison of novel textures generated after training for 29.5k epochs on five car models.** In this instance *ORIG* entirely missed the mark of the target coloring scheme. However, *PE* is not much better, though arguably closer to the slightly darker shade of green in the input-image. Similar to the previous figure, *PE*'s textures display higher-frequency variation in color, while *ORIG*'s textures are consistently more smooth. This fits with the design of the positional encoding, however at least in these examples this does not result in discernibly more realistic textures. Altogether, these two results fail to suggest that the improved granularity of the positional encoding in learning the training set will naturally translate to improved synthesis of novel textures.

5. Discussion

5.1. Shape Encoding

With regards to the shape encoder modifications, we had hoped to see loss converging faster and to a smaller value than the overfitting performance yielded by the baseline. Quantitatively we see this is not the case; the differences in loss convergence and metrics is not significant. Qualitatively we see that pretrained weights make no difference and that Pointnet++ seems to yield inferior results. We will need to put more thought into what sort of geometric details are captured by the shape embedding.

5.2. ℓ_1-loss vs. Perceptual Loss

With regards to changing the loss, quantitative results in different metrics yielded no significant differences when compared to the baseline, but the variances and loss values are in general higher than the original network. Through qualitative observations, we note that perceptual losses can capture some minor horizontal texture features slightly better. In contrast to our expectation, the experiment indicates that capturing deep features or perceptual similarities at image level does not contribute to learning essential texture information in 3D. One future direction is to investigate which features are better learned by which networks/loss functions, and how we can combine these advantages to achieve better performance.

5.3. Positional Encoding

In our experiment, the positional encoding transformation corresponded with a clear improvement in TextureField's ability to learn detailed textures from the training set. However, this did not result in a corresponding improvement in performance when synthesizing novel textures. Altogether, there are a couple of caveats to keep in mind with this experiment:

- The limitations in data and training prohibit us from conclusively evaluating the impact of the positional encoding on synthesizing novel textures. For example, it's conceivable that the improved performance of *PE* in learning textures in the training set may come at the cost of generalizing to new textures. In other words, it's possible the positional encoding as it's currently structured may present the inherent trade-off of overfitting: learning high-frequency details in the training set versus generalizing to novel textures. On the other hand, the failure in both these examples could very reasonably be the result of the minuscule training data.

- Given the limitations in training time, we cannot speak conclusively about the relative potential of these models in less-restrictive scenarios. For example, it's quite possible that *ORIG* would always eventually learn a similar level of granular textures as the positional encoding model given enough training time.

- The larger input vector of *PE* (48 dimensions instead of three) inherently provides considerably more trainable parameters in the first fully connected layer of TextureFields. It's conceivable that some amount of improved performance with regards to learning training textures is the result of simply having more trainable parameters. In subsequent studies this could be controlled for by increasing the size of *ORIG*'s initial, fully-connected layer.

None-the-less, the clear improvement in the positional encoding's ability to learn detailed textures suggests that this is a worthy avenue of further research. Future studies could include:

- Expanding training time. This will help investigate whether or not the original model is in fact capable of learning the same level of detail, albeit at a slower pace.

- Expanding the training data. This can help inform whether the positional encoding's approximation of high-frequency details could be applied to the task of novel texture synthesis after learning a more comprehensive latent space.

- Exploring other techniques for learning high-frequency data. NeRF's positional encoding was partly selected for its simplicity in assimilating with TextureFields. Subsequent research with SIREN[14] and Fourier transformations[15] have demonstrated improved efficacy in learning high-frequency data. For example, recent research by Chan et al. [1] has applied neural radiance fields in the context of a generative adversarial network while incorporating the SIREN method for approximating high-frequency data. This research may provide relevant insight for how to better incorporate high-frequency techniques into TextureFields as it bridges the conceptual similarity of neural radiance fields' implicit 3D scene-representation with TextureFields' operation over a latent space. An interesting avenue of research would be studying the varying benefits and drawbacks of these methods for learning and synthesizing detailed textures.

- Expanding the size and impact of the latent space for the shape and image encodings. One can imagine that the improved expressiveness of these high-frequency learning techniques should correspond with a richer latent space to draw from. Incorporating these techniques into the generation of the shape and image encodings may thus be a worthwhile endeavor.

6. Challenges

We encountered the biggest challenges of this project when we learned from the authors of TextureFields that training on the shapenet cars data took approximately a week. Our original plan was to train on the Shapenet cars for comparison, and then train on the 3DFuture data. Realizing that this plan was infeasible due to time constraints, we had to develop an entirely new training and evaluation plan. Unfortunately this led to us not using the 3D Future data, meaning the time we spent preprocessing the data became a sunk cost.

Another challenge came when we attempted to implement DeepSDF as the shape encoder within the TextureFields architecture. DeepSDF, where a neural networks approximates the signed distance field, belongs to the same class of implicit representation networks as Occupany Net. Although DeepSDF takes a point cloud as input, it requires an essential preprocessing before passing the input into the model. After spending a substantial amount of time attempting to get the DeepSDF preprocessing code operational, we decided that our efforts would be better focused elsewhere. Another bottleneck was that DeepSDF uses an auto-decoder model as opposed to auto-encoder. This made it less clear how best to integrate it with TextureFields.

7. Reflection

A limitation of our experimental process is that overfitting on a single example in not the ideal way to evaluate model performance. A better task would be to train on the entire dataset and evaluate generalization. Additionally, we also seek to incorporate an architecture like MeshCNN that can operate directly on meshes as opposed to requiring point cloud sampling with the rendering and unprojection pipeline.

Although we were not able to train on the entire dataset as initially hoped, we did manage to implement different shape encoders, perceptual losses and a positional encoding. Even within these limitations, the positional encoding implementation offered promising improvements over the base model. Although are other various experiments did not yield results better than the existing works, some experiments produced results with similar quality and most importantly, we saw pros and cons for different methods and processed the reasoning about why certain issues would occur. We accomplished the base and target goals that we modified the architecture, tried applying more state-of-the-art techniques and observed the qualitative and quantitative results.

This is a novel and challenging problem and thus despite not achieving the ultimate goal, we still learned some valuable lessons both on the engineering and research aspect and got an opportunity to ask important questions in the field of 3D deep learning.

References

[1] Eric Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis, 2020. 10

[2] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository, 2015. 2, 3

[3] Huan Fu, Rongfei Jia, Lin Gao, Mingming Gong, Binqiang Zhao, Steve Maybank, and Dacheng Tao. 3d-future: 3d furniture shape with texture. *arXiv preprint arXiv:2009.09633*, 2020. 2, 3

[4] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014. 1

[5] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan C. Russell, and Mathieu Aubry. Atlasnet: A papier-mâché approach to learning 3d surface generation, 2018. 1

[6] Xianfeng Gu, Steven J. Gortler, and Hugues Hoppe. Geometry images. 2002. 1

[7] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. Meshcnn. *ACM Transactions on Graphics*, 38(4):1–12, Jul 2019. 1

[8] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space, 2019. 1

[9] 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, 2020. 3, 4, 5

[10] Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. Texture fields: Learning texture representations in function space, 2019. 1, 2, 3

[11] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation, 2019. 1

[12] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space, 2017. 1, 2

[13] J. Schmidhuber. Learning factorial codes by predictability minimization. *Neural Computation*, 4(6):863–879, 1992. 1

[14] Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions, 2020. 3, 4, 10

[15] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng. Fourier features let networks learn high frequency functions in low dimensional domains, 2020. 2, 3, 4, 10

[16] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 2, 4

8. APPENDIX: Supplementary Material

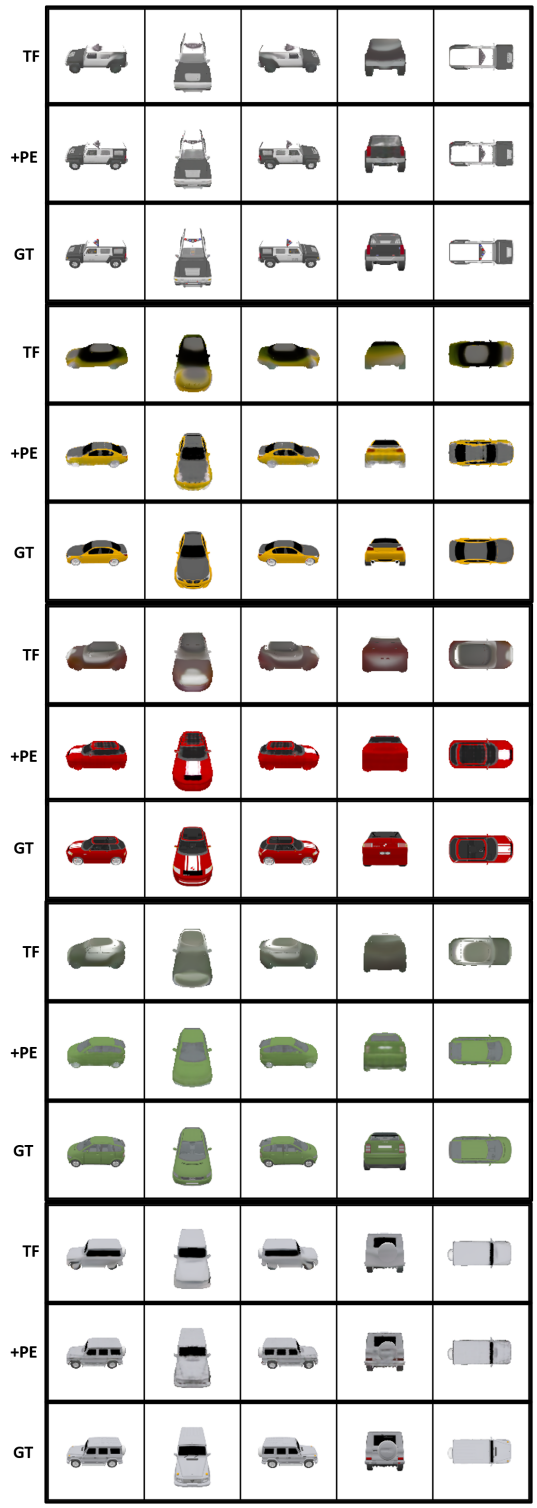


Figure 22. **Comparison of textures generated by *ORIG* and *PE* for all five cars in the training set after 29.5k epochs of training.** Note that *ORIG* happened to generate substantially distorted textures for the red, yellow and green cars on the final 29.5k epoch. This is a normal periodic occurrence for both models over the initial course of training- see the textures generated across training in figures 23-25. One example of the detail *PE* is able to represent is evident in the license plate captured on the green sedan.

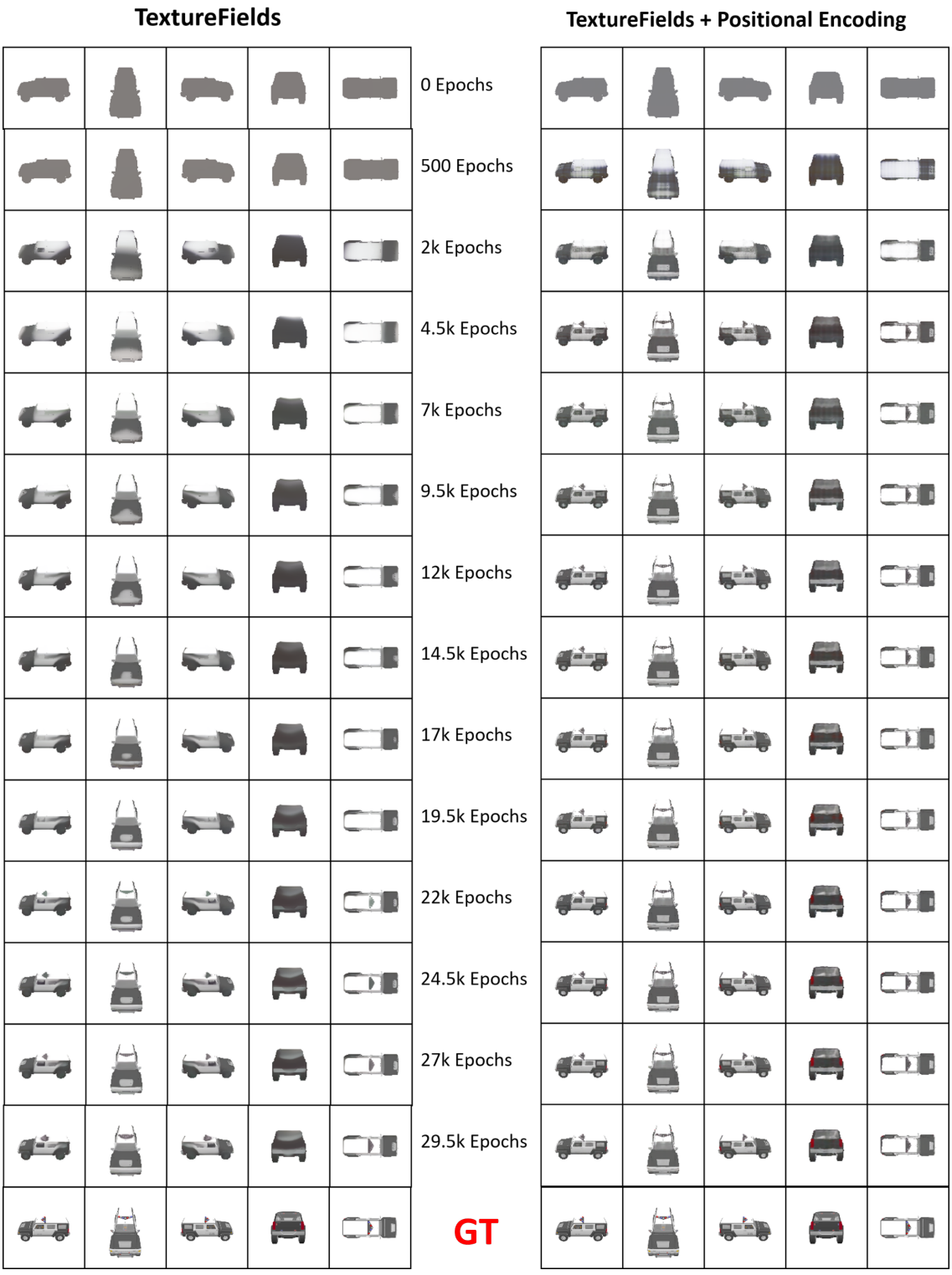


Figure 23. **Comparison of textures generated by *ORIG* and *PE* for a car in the training set at 2.5k epoch intervals across the course 29.5k epochs of training.** The bottom row are the ground truth images. Note the early approximation of details for the *PE* model.

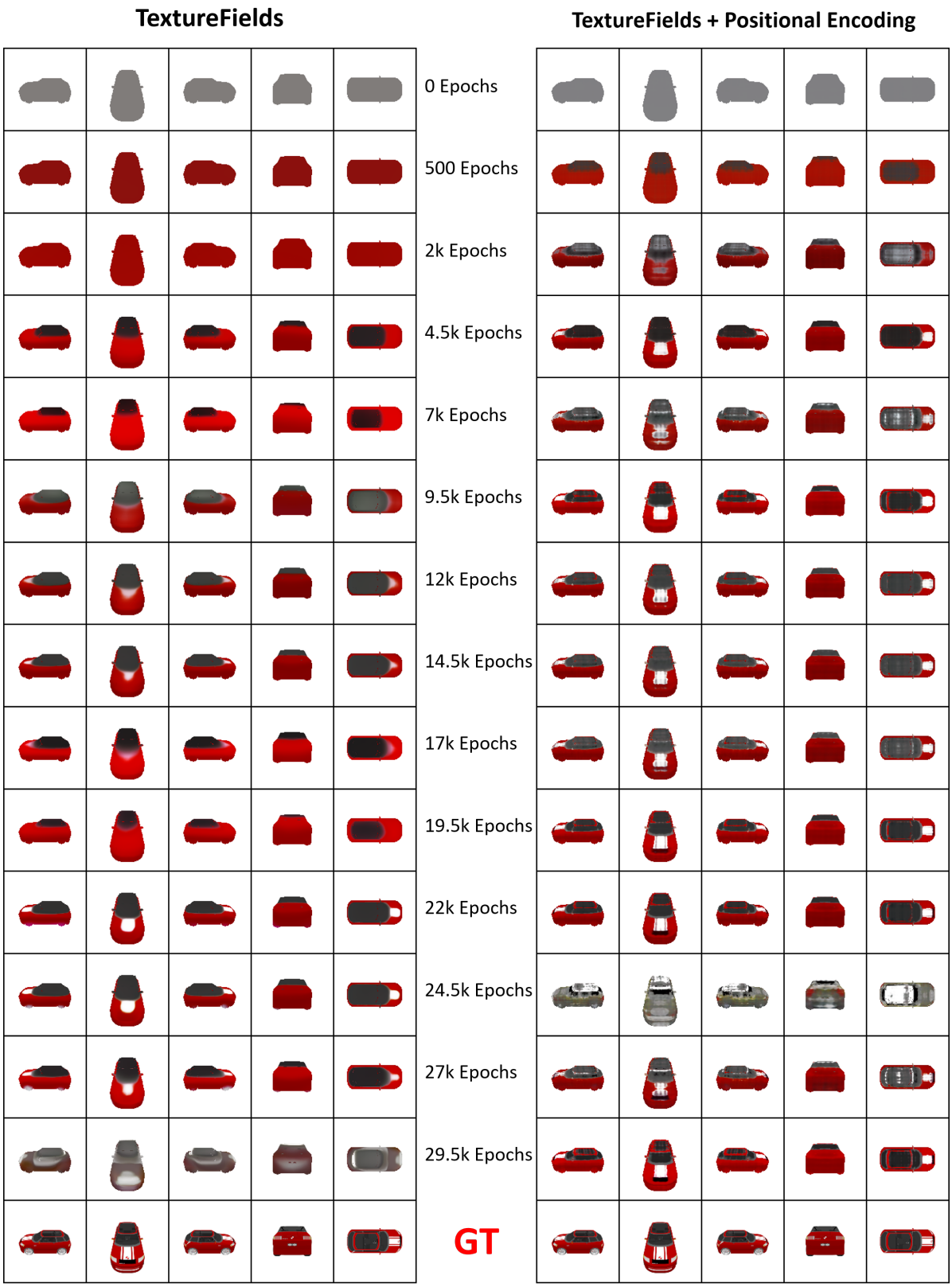


Figure 24. **Comparison of textures generated by *ORIG* and *PE* for a car in the training set at 2.5k epoch intervals across the course 29.5k epochs of training.** The bottom row are the ground truth images. Note the early approximation of details for the *PE* model. There are abrupt distortions in the generated textures for both *ORIG* and *PE* at epochs 29.5k and 24.5k, respectively

Figure 25. **Comparison of textures generated by *ORIG* and *PE* for a car in the training set at 2.5k epoch intervals across the course 29.5k epochs of training.** The bottom row are the ground truth images. Note the early approximation of details for the *PE* model.