# AUV-Net: Learning Aligned UV Maps for Texture Transfer and Synthesis

Zhiqin Chen[1,2] Kangxue Yin[1] Sanja Fidler[1,3,4]

NVIDIA[1] Simon Fraser University[2] University of Toronto[3] Vector Institute[4]

## Abstract

*In this paper, we address the problem of texture representation for 3D shapes for the challenging and under-explored tasks of texture transfer and synthesis. Previous works either apply spherical texture maps which may lead to large distortions, or use continuous texture fields that yield smooth outputs lacking details. We argue that the traditional way of representing textures with images and linking them to a 3D mesh via UV mapping is more desirable, since synthesizing 2D images is a well-studied problem. We propose* AUV-Net *which learns to embed 3D surfaces into a 2D aligned UV space, by mapping the corresponding semantic parts of different 3D shapes to the same location in the UV space. As a result, textures are aligned across objects, and can thus be easily synthesized by generative models of images. Texture alignment is learned in an unsupervised manner by a simple yet effective texture alignment module, taking inspiration from traditional works on linear subspace learning. The learned UV mapping and aligned texture representations enable a variety of applications including texture transfer, texture synthesis, and textured single view 3D reconstruction. We conduct experiments on multiple datasets to demonstrate the effectiveness of our method. Project page: https://nv-tlabs.github.io/AUV-NET.*

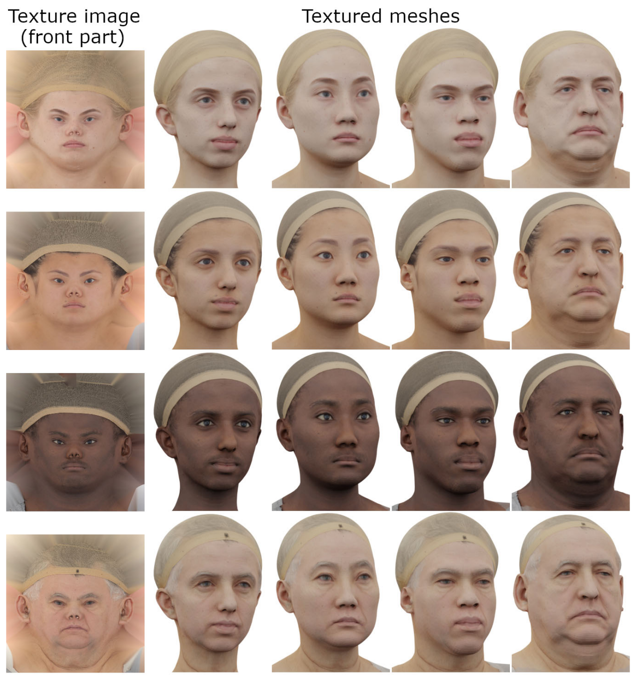

Figure 1. AUV-Net learns aligned UV maps for a set of 3D shapes, enabling us to easily transfer textures between shapes.

## 1. Introduction

The field of 3D shape reconstruction and synthesis has witnessed significant advancements in the past few years. By utilizing the power of deep learning, several works reconstruct 3D shapes from voxels, point clouds, single and multi-view images, with a variety of output shape representations [12, 14, 15, 20, 24, 31, 52]. 3D generative models have also been proposed to synthesize new shapes [4, 11, 19, 35, 40], with the aim of democratizing 3D content creation. However, despite the importance of textures in bringing 3D shapes to life, very few methods have tackled semantic-aware texture transfer or synthesis for 3D shapes [5, 10, 18, 22, 36, 37, 51].

Previous work on texture generation mostly relies on warping a spherical mesh template to the target shape [5, 10, 22, 36], therefore obtaining a texture map defined on the sphere's surface, which can be re-projected into a square image for the goal of texture synthesis. NeuTex [48] generates 3D shapes with a neural implicit representation for arbitrary surface topology, yet embeds the surface of the shape onto a sphere, which also results in a spherical texture map. Spherical texture maps can only support limited topology, and may introduce severe distortions for thin parts such as animal limbs [27, 44]. Another line of work uses implicit texture fields for texture synthesis [33], without relying on explicit texture mapping. Although texture fields were successfully applied to multi-view image reconstruction [32], they have primarily been used for fitting a single object or scene. Generative models usually suffer from overly smoothed synthesized textures [38, 49].

In contrast, the traditional UV mapping in computer graphics handles arbitrary shape topology and avoids heavy distortions by cutting the surface into pieces and mapping different pieces to different regions on the 2D UV plane. It further preserves texture details by storing the texture in a high-resolution texture image. However, the UV mappings are usually created by 3D artists, and thus are inconsistent across different shapes. Therefore, using such representation for texture synthesis and transfer would require dense shape correspondences.

In this paper, we propose to train a neural network to predict the UV mapping and the texture image jointly, aiming at high-quality texture transfer and synthesis without needing to conform to a pre-defined shape topology. Specifically, our network learns to embed 3D coordinates on mesh surfaces into a 2D aligned UV space, where corresponding parts of different 3D shapes are mapped to the same locations in the texture image, as shown in Figure 1. Such alignment is enabled by a simple yet effective texture alignment module inspired by traditional linear subspace learning methods such as Principal Component Analysis (PCA), as shown in Figure 3. The network generates a basis shared by all shape textures, and predicts input-specific coefficients to construct the texture image for each shape as a linear combination of the basis images. This forces the texture images to be aligned so that they can be effectively decomposed into combinations of basis images, as visualized in Figure 2. Afterwards, the network reconstructs the colors of the input shape by learning a UV mapping to index the aligned texture image. To unwrap 3D shapes of complex structure or topology, we further introduce a masking network that cuts the shape into multiple pieces to reduce the distortion in the UV mapping.

Our method effectively aligns textures across all shapes, allowing us to swap textures between different objects, by simply replacing the texture image from one object with another. The aligned high-quality texture images produced by our method make it significantly easier to train generative models of textures, since they are aligned and disentangled from geometry. They also enable textured 3D shape reconstruction from single images. We perform extensive experiments on multiple categories including human heads, human bodies, mammals, cars, and chairs, to demonstrate the efficacy of our approach.

## 2. Related work

We discuss previous work that is most relevant to ours in the fields of texture transfer and synthesis for 3D shapes.

**Template-based methods** assume that all target shapes can be represented by deforming a template mesh, usually a sphere [5, 10, 22, 27, 36, 44] or a plane [34, 47]. The UV mapping of the template mesh is given and transferred to the target shape after deformation. However, by imposing a mesh template, these methods often cannot capture details, especially when the topology or the structure of the target shape is complex. For example, when deforming a sphere into a human body, it is hard to accurately reconstruct the fingers. Even if the deformation is successful, the texture of the fingers, when projected from the human body to a sphere and then to its texture image, is typically heavily distorted.

**UV map from artists.** Another line of work [8, 51] does not assume template meshes are given, but instead assumes that the UV maps are provided with the 3D shapes. The UV maps and textures are typically modeled by artists and can be in arbitrary layouts. To address this issue, these methods usually require ground-truth semantic segmentation of the texture image or the 3D shape for semantic-aware texture synthesis. In our work, we aim to perform texture synthesis without such supervision. Automatic UV mapping has also been extensively studied in computer graphics, though for single shapes. It includes mesh parameterization with certain constraints, and surface cutting to generate charts with disk topology. We refer to [39] for a survey of related techniques. Different from these traditional methods, we learn aligned UV maps for a set of shapes.

**Discretization and colorization.** Instead of adopting UV maps to reduce the dimensionality of textures from 3D to 2D, some methods discretize the 3D shapes into "atoms" and then colorize each "atom". When a shape is represented as a voxel grid, the shape can be textured by predicting the color of each voxel [9, 42]. For triangle meshes, the color of each vertex can be predicted [17]. However, since discretization is in 3D rather than in 2D (pixels), these approaches either cannot scale up, or cannot predict the color efficiently due to the irregularity of the representation.

**Texture fields** [33] predict the color for each 3D point in a continuous 3D space. The NeRF family [32] also adopts this approach, by using the viewing direction as an additional condition for predicting the color of each point. Since the NeRF family does not directly generate textures for 3D shapes, we mainly discuss and compare with Texture Fields in this paper. One major issue of Texture Fields is that it is unable to represent high-frequency details, which is a property of the MLPs that it uses. Positional encoding [32] and SIREN [41] are proposed to alleviate this issue, which works well on overfitting of single shapes. However, performance degrades significantly in generative tasks. The results of implicit methods tend to be smooth and lack high-frequency details [6, 41].

**Shape correspondences.** There is a large body of work that finds dense correspondences among shapes [16, 28], which can also enable texture transfer. However, these methods do not take color into account when finding correspondences, which may hinder their performance.

## 3. Our Approach

In this section, we first introduce our core alignment module in 2D and verify it with a 2D-to-2D image alignment experiment in Sec 3.1. We then explain our network architecture for learning aligned UV maps for 3D shapes in Sec 3.2. Finally, in Sec 3.3 we show applications enabled by

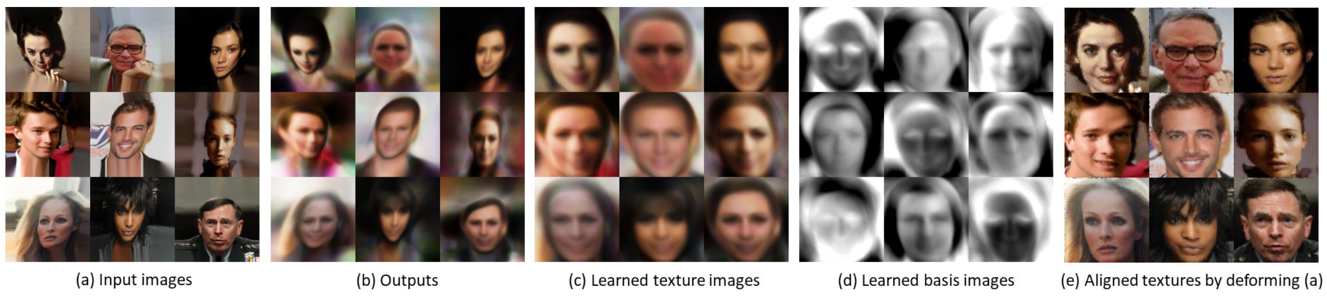
(a) Input images
(b) Outputs
(c) Learned texture images
(d) Learned basis images
(e) Aligned textures by deforming (a)

Figure 2. Results of the 2D toy experiment on the face dataset. Our network reconstructs input images (a) by learning a set of basis images (d) and linearly combining them into aligned texture images (c), and then deforming the texture images (c) into the outputs (b) via learned UV mapping. The learned UV mapping can be used to deform the input images (a) into aligned high-quality texture images (e).

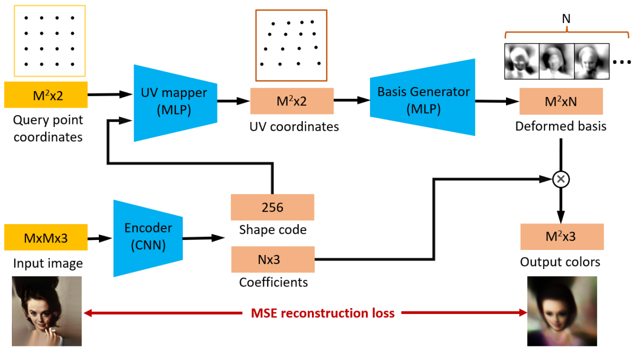

Figure 3. Network architecture of the 2D toy experiment on the face dataset, to demonstrate the concept of our alignment module.

our approach, including texture transfer, texture synthesis, and texture prediction from single images.

### 3.1. Texture alignment module

Learning aligned textures for a set of shapes is a complex task. However, we show that a simple alignment module performs surprisingly well. Before we introduce our network for 3D shapes, we will use a toy 2D experiment to demonstrate how the alignment module works.

**Task.** Given a set of face images in random poses, as shown in Fig. 2(a), we aim to align them into a canonical pose, as shown in Fig. 2(e). We take 1,000 face images from CelebA-HQ [25, 29] and perform random perspective transformations to obtain 128x128 training images, such as those in Fig. 2(a). To link this task with texture mapping, one could consider training images in Fig. 2(a) as square shapes in 2D, and Fig. 2(e) are their aligned texture images. The color of each pixel in the square shape must be retrieved from the shape's texture image, by mapping the pixel's coordinates into the UV space to get the UV coordinates, and then indexing the texture image with those UV coordinates.

**Insight.** We take inspiration from classic linear subspace learning methods such as eigenfaces [45], where a basis is computed via PCA for a set of face images, so that each face is decomposed into a weighted sum of the eigenfaces. Note that PCA works best when the images are aligned. Therefore, if a network is designed to decompose the input images into weighted sums of basis images, and is allowed to deform the input images before the decomposition, the network should learn to align the input images into a canonical pose, and decompose the aligned images so as to minimize the reconstruction error.

**Framework.** Fig. 3 illustrates our alignment module operating for 2D images. It is composed of three neural networks: a *basis generator* to predict a set of basis images; an *encoder* to predict the coefficients to weigh the basis images; and a *UV mapper* to predict the UV coordinates for each query point. The encoder also predicts a shape code to condition the UV mapper.

**Basis generator.** Our basis generator is a Multilayer Perceptron (MLP) that takes a 2D point $(x, y)$ as input and outputs the color of this point. For $N$ basis images, the network outputs $N$ values ($N$ gray-scale colors). In this toy 2D experiment, we use $N = 128$. We adopt an MLP for generating the basis because it is fully differentiable with respect to both the colors of the basis and the input point coordinates. In contrast, if a Convolutional Neural Network (CNN) or a grid of learnable weights is applied to generate the basis, one would need to index the output grid with query points, limiting the gradients from the output color to the basis and the query point coordinates to small neighborhoods.

**UV mapper.** Once trained, we can input a regular grid of query points to the basis generator to obtain the aligned basis images such as those in Fig. 2(d), and the aligned texture images shown in Fig. 2(c) by multiplying the basis images with the coefficients. However, at training time, we need to "deform" the texture images to reconstruct the input images in Fig. 2(a). This is achieved by the UV mapper, which maps the query points sampled from the square shape into UV coordinates to index the texture image, as shown in Fig. 3. The final deformed outputs are shown in Fig. 2(b). The UV mapper is an MLP conditioned on a shape latent code. It takes 2D point coordinates concatenated with the shape code as input, and outputs UV coordinates.

**Encoder and loss function.** Since the inputs are images, we use a 2D CNN as our encoder to predict shape codes

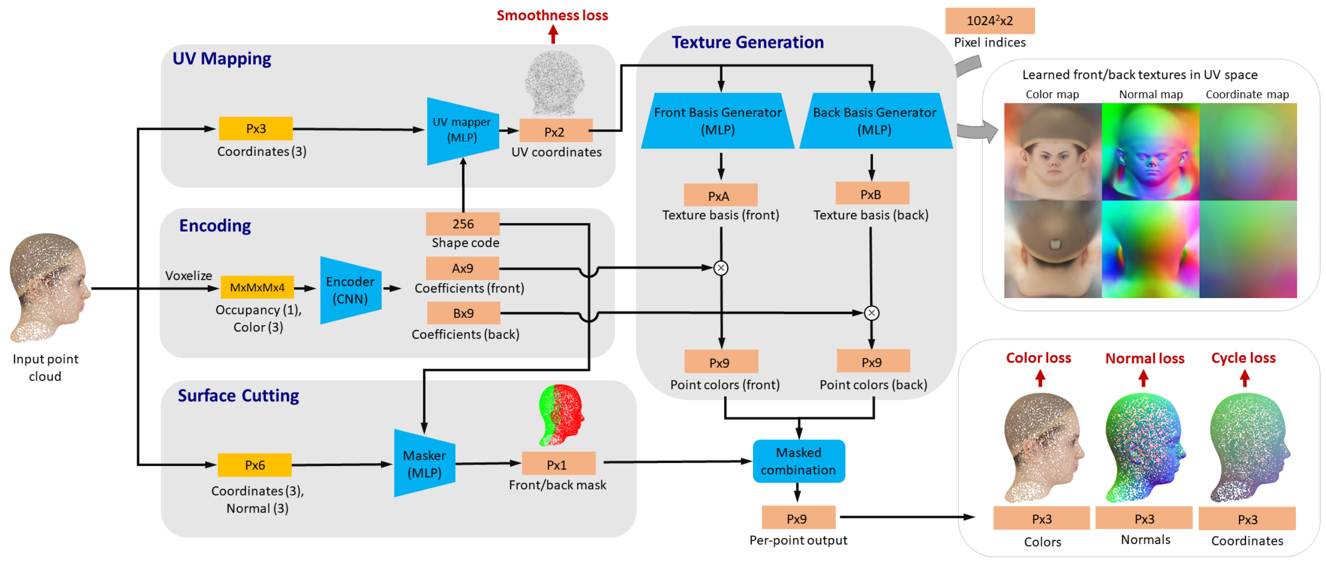

Figure 4. Network architecture of our AUV-Net. The encoder predicts the shape code and the coefficients from the voxelized input point cloud. The UV mapper and the masker take as input the shape code and the query points from the input point cloud, and output the UV coordinates and the segmentation mask, respectively. The UV coordinates are fed into the two basis generators to obtain the basis colors for each query point, and the basis colors are multiplied by the predicted coefficients to generate the actual colors for each query point. Those colors from the two basis generators are selected by the predicted segmentation mask to produce the final colors.

and coefficients. The deformed basis produced by the basis generator is multiplied with the coefficients to produce the final output, as shown in Fig 3. We use Mean Squared Error (MSE) between the output and the input image as the reconstruction loss. During early stage of training (first few epochs), we apply a prior loss, i.e., MSE between the query point coordinates and their corresponding UV coordinates, to encourage the UV mapper to perform an identity mapping, so that the basis is initialized with appropriate orientation, scale, and position.

**Obtaining high-quality texture images.** After training, our network produces aligned texture images as shown in Fig. 2(c). However, the images are in low quality since they are constructed by a limited number of over-smoothed basis images. To obtain a high-quality texture image with details, we can deform the input image into the UV space. We sample points from the input image, feed those points to the UV mapper to obtain UV coordinates, use the UV coordinates and colors of the sampled points to fill a blank image, and finally inpaint the missing regions. We use [43] for inpainting. The results are shown in Fig. 2(e).

## 3.2. Learning aligned UV maps for 3D shapes

Our network for 3D shapes, dubbed AUV-Net, is built upon the alignment module in Fig. 3. It can be thought of as a 3D-to-2D version of the 2D-to-2D alignment module, with modifications to address issues caused by the properties of 3D shapes. The architecture of AUV-Net is shown in Fig. 4. In the following, we will first describe the notable changes in AUV-Net compared to the 2D alignment module, and then the loss functions and training details.

**Predicting color, normal, and coordinate maps.** 3D shapes tend to have textures with large areas of pure color or similar colors. Those featureless regions make it hard for our network to align the textures with only a loss function defined on colors. Therefore, in addition to the color maps, our network also produces normal maps and coordinate maps, as shown in Fig. 4 top-right. The normal maps are used for predicting the unit normals of input points. The coordinate maps are used for predicting the positions of the input 3D points, therefore forming a 3D-2D-3D cycle in our network, which encourages injective UV mapping.

**Cutting surfaces with a masking network.** Unlike images, it is usually impossible to embed a 3D shape onto a 2D plane without overlap or severe distortion. Therefore, we introduce a *masker*, to generate a segmentation mask for the input shape, as shown in Fig. 4 bottom-left. This can be considered as cutting the 3D shape into multiple pieces, so that each piece can be represented by a single texture image. The input to the masker contains point coordinates, point normals, and the shape code. The normals are essential to segmenting the shapes, since thin parts such as fingers on a human body mesh are very hard to segment with only point coordinates. The predicted segmentation mask ($M$) is used to mask the outputs of the two basis generators ($A$ and $B$), as $M \cdot A + (1 - M) \cdot B$, to produce the final output.

**Multiple basis generators.** We introduce two basis generators to represent the "front" and the "back" part of a shape, respectively. The "front" does not have to literally denote the front-facing part of a shape. It simply refers to a part of the shape so that the union of the "front" and "back" covers the entire shape. The outputs of the two basis generators are shown in Fig. 4 right, where the head is being represented with two texture maps. Note that the number of basis generators does not have to be two; we use four basis generators for chairs in our experiments.

**Shared UV mapper.** We use one shared UV mapper for both the front and the back basis generators, instead of two independent UV mappers. This is based on a careful consideration. In our experiments, one of the most prominent issues when we transfer the texture from one shape to another is that we inevitably obtain seams between the two pieces of shapes using two different texture images. A shared UV mapper alleviates this issue by forcing the two pieces to share the same boundary in the texture images. It does not fully resolve the seam issue, but it is very helpful in practice. After we inpaint the texture images, the seams are barely visible in most cases.

**Loss functions.** To train AUV-Net, the meshes are converted into point clouds with normals and colors, as input to our network. We also voxelize the point clouds to obtain colored voxel grids as input to the 3D CNN encoder. The overall loss function is composed of five terms:

$$L = w_c L_c + w_n L_n + w_x L_x + w_s L_s + w_p L_p \qquad (1)$$

where $L_c$, $L_n$, $L_x$ denote the color loss, the normal loss, and the cycle consistency loss on the 3D coordinates, respectively. They are defined as MSEs between the predictions and the ground truth.

$L_s$ is the smoothness loss. For a subset of input points, we find their neighbors within a distance $\sigma = 0.02$ , and use the distances between the points and their neighbors to regularize the corresponding distances in the UV space:

$$L_s = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} |D(p_i, p_j) - D(q_i, q_j)| \cdot T(p_i, p_j) \quad (2)$$

where $N$ is the number of input points, $M$ is the size of the subset, $p_i$ is the $i$-th input 3D point, $q_i$ is the 2D UV point predicted for $p_i$ by the UV mapper. $D(a, b)$ is the Euclidean distance between point $a$ and $b$. $T(a, b)$ is defined as 1 if $D(a, b) < \sigma$, and 0 otherwise. In each mini-batch, we process one shape, with $N = 16,384$ and $M = 2,048$.

$L_p$ is the prior loss to initialize the UV coordinates and the masks. It may vary per category of the training shapes. For the human head dataset shown in Fig. 1 and Fig. 4, where all the heads are facing z direction, we have:

$$L_p = \frac{1}{N} \sum_{i=1}^{N} (p_i^x - q_i^x)^2 + (p_i^y - q_i^y)^2 + (m_i - n_i)^2 \quad (3)$$

where $p_i^x$ is the x coordinate of $p_i$, $q_i^x$ is the x coordinate of $q_i$, $m_i$ is the masking value predicted by the masker for $p_i$. $n_i$ is defined as 1 if the unit normal of $p_i$ in the z direction is greater than $-0.5$, and 0 otherwise. This prior loss initializes the UV mapping by projecting the 3D points onto the xy-plane. To cut the shape into two pieces, this prior loss follows our prior that: if the angle between a point's normal and z axis is less than 120 degrees, the point belongs to the "front" part. Similar to Sec. 3.1, prior loss is only used in the first few epochs of training to initialize the mask and the UV coordinates. We provide the definitions of prior losses for other categories in the supplementary.

**Assumptions on the training set.** Note that the above loss terms assume certain properties of the training dataset. First, the shapes need to have part-level correspondences, as the network actually assigns dense correspondences between shapes when it maps all shapes into the same UV space. Therefore, we only train our model on 3D shapes of the same category. Second, the shapes need to be pose aligned, e.g., heads should all face z direction in the aforementioned human head dataset. We also normalize all shapes to unit boxes before training, to avoid interference of drastically different scales.

**Multi-stage training.** We train the network in three stages, due to a trade-off between the quality of the texture alignment and the level of distortion. In some cases, aligning textures requires heavy distortion in the texture images, e.g., when aligning a sedan with a van (Fig. 5). However, less distortion is a desirable feature that reduces aliasing effect when rendering the textures, and makes post-processing easier, e.g., when being edited by an artist. We find that if the network is trained with fixed weighting of the loss terms, we cannot get both the alignment and minimal distortion. Therefore, we first initialize the network with prior loss $L_p$ and a set of weights focused on minimal distortion. In the second stage, we remove $L_p$, and use weights focused on alignment. In the final stage, we use weights focused on minimal distortion. For the human head dataset, the first stage has 10 epochs, with $\{w_c, w_n, w_x, w_s, w_p\} = \{1, 0.5, 100, 100, 1\}$; second stage has 2,000 epochs, with $\{1, 0.5, 1, 1, 0\}$; third stage has 2,000 epochs, with $\{1, 0.5, 100, 100, 0\}$. Training takes 2 days on one NVIDIA RTX 3080 Ti GPU. Other training details are in the supplementary.

## 3.3. Applications

**Texture transfer.** After training AUV-Net, we obtain aligned high-quality texture images ($1024^2$ in our experiments) for all training shapes, as shown in Fig. 1. The fact that these texture images are aligned allows us to transfer textures between two training shapes by simply swapping their texture images, as shown in Fig. 1 and 5. We denote this application as **Tsf (transfer)**. Given a new shape that is not in the training set, we can also texture it by mapping its vertices into the aligned UV space. This is done via a posttraining optimization stage, in which we add the new shape into the training set, and continue training the network for a few epochs. During the optimization, we fix the weights of the basis generators to reuse the well-learned texture basis.

**Texture synthesis.** A great advantage of having aligned texture images is that it allows us to utilize existing 2D generative models to synthesize new textures for 3D shapes. We train StyleGAN2 [26] in experiments and show results in Fig. 9. We denote this application as **Gen (generation)**.

**Single-image 3D reconstruction.** We can condition texture synthesis on a variety of inputs, for example, reconstructing textured 3D shapes from single images, as shown in Fig. 10. To this end, we add a 2D ResNet [21] image encoder to predict the texture latent code and the shape code from an input image, a CNN decoder to predict the aligned texture images from the texture latent code, and an IM-Net decoder [14] to predict the geometry of the shape conditioned on the shape code. We denote this application as **SVR (single view reconstruction)**. Implementation details are provided in the supplementary.

## 4. Experiments

**Datasets.** We use six datasets in our experiments, as listed in Table 1. Information about dataset licenses is in the supplementary. We mainly perform generative tasks (Gen, SVR) on ShapeNet [7] categories since other datasets have too few training shapes. Note that the original shapes in ShapeNet usually have complex geometry but simple textures. We create a new version of ShapeNet Cars and Chairs better suited for the texture transfer/synthesis task, by simplifying the meshes to reduce geometric details and baking the geometric details into textures.

### 4.1. Texture Transfer

We show texture transfer results in Fig. 1 and 5. Only non-ShapeNet categories are shown due to page limit. More results can be found in the supplementary. Our method uses cues such as colors, normals, and positions when learning aligned UV mapping, and therefore performs well on aligning facial orifices, car windows and wheels, fingers, and animal limbs. Previous methods find dense correspondences among shapes by deforming geometry [16, 28]. However, they do not utilize color information, and may thus misalign regions with fewer geometric cues, as shown in Fig. 6. We show results on transferring textures to new shapes that are clearly different from the training shapes in Fig. 5. Our method is able to correctly texture an over-simplified texture-less car model, and transfer textures to a cartoon character model.

**Quantitative evaluation.** To evaluate the alignment quality, we label one texture image with a different color per semantic part, as shown in Fig. 7 (b). Since the texture image is aligned across shapes, we get semantic segmentation of

| Dataset name | Number of Shapes | Applications |
|---|---|---|
| ShapeNet [7] cars | 7,497 | Tsf, Gen, SVR |
| ShapeNet [7] chairs | 6,778 | Tsf, Gen, SVR |
| Turbosquid [3] cars | 436 | Tsf |
| RenderPeople [1] human bodies | 500 | Tsf |
| Triplegangers [2] heads | 515 | Tsf, Gen |
| Turbosquid [3] animals | 442 | Tsf |

Table 1. Datasets used in our experiments. Tsf, Gen, and SVR refer to the applications listed in Sec. 3.3.

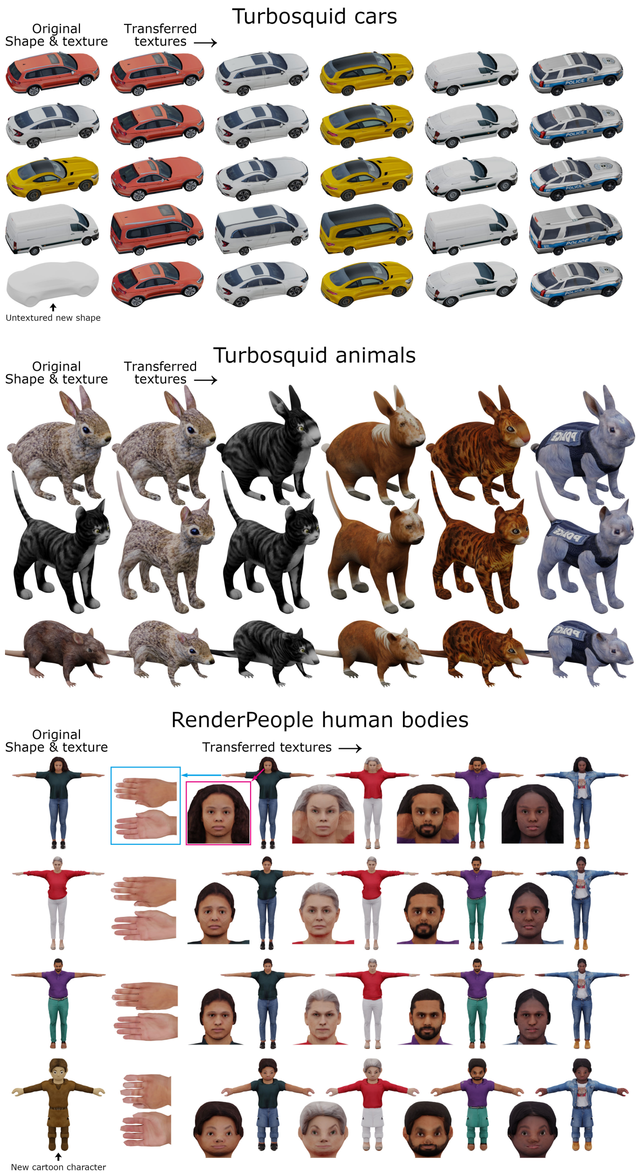

Figure 5. Texture transfer results. We show three categories in this figure: Turbosquid cars (top), Turbosquid animals (middle), and RenderPeople human bodies (bottom). Triplegangers heads can be found in Figure 1. For RenderPeople, we show a zoom-in of the head on the lower left of each shape; we also show zoom-ins of hands for the second column of shapes.

3D shapes with a single labeled example. We evaluate our part segmentation of shapes with ground truth segmentation

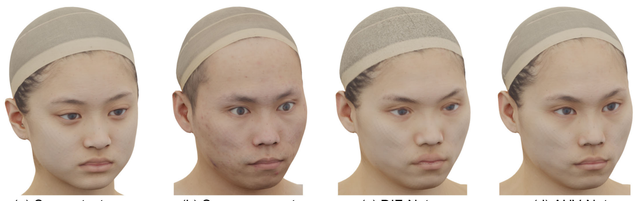

Figure 6. Comparison with DIF-Net [16] on texture transfer. The texture is transferred from (a) to (b). In (c), the eyes' shapes are not changed with respect to (a), the lips are misaligned, and the hat is lower than it should be compared to (b). Those details are mostly represented in colors rather than geometry.

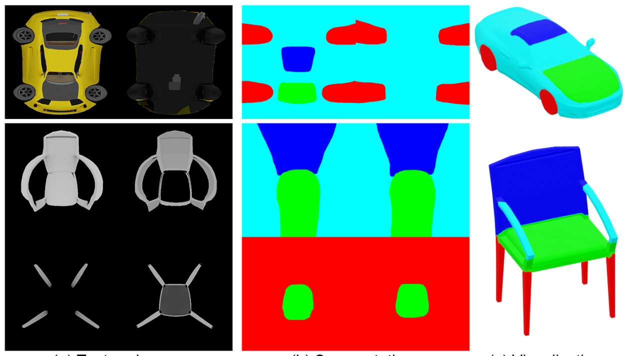

Figure 7. Sample texture images and segmentation on ShapeNet cars and chairs. (a) shows texture images before inpainting. Note that there are 2 texture images for each car and 4 for each chair. In (b), we show the segmentation we used to produce Table 2. A visualization on 3D shapes is shown in (c).

| Dataset (#parts) | ShapeNet cars (4) | ShapeNet chairs (4) |
|---|---|---|
| Segmented parts | Wheel, body, hood, roof | Back, seat, leg, arm |
| BAE-Net | 59.3 | 85.2 |
| DIF-Net | 69.0 | 80.3 |
| AUV-Net | **72.7** | **85.8** |

Table 2. Semantic segmentation results in IOU, comparing with BAE-Net [13] and DIF-Net [16].

provided in the ShapeNet part dataset [50]. We compare with BAE-Net [13] that performs one-shot shape segmentation and DIF-Net [16] that learns dense correspondences, and report Intersection Over Union (IOU) in Table 2. Our method outperforms alternatives.

**Ablation study.** We provide the ablation study in Table 3 and Fig. 8, where we remove one of the five loss terms in Eq. 1, or the masker module. We use the same evaluation setting described above. The results are consistent with our design choices of the individual modules. The color loss $L_c$, the normal loss $L_n$, and the cycle loss $L_x$ are designed to help find correspondences, therefore removing them often causes the performance to drop (Table 3). The smoothness loss $L_s$ is designed to regularize the UV coordinates; it may hurt the correspondence, but removing it can cause certain parts to be squished (Fig. 8 column 4), thus making texture synthesis difficult. The cycle loss $L_x$ also helps regularize

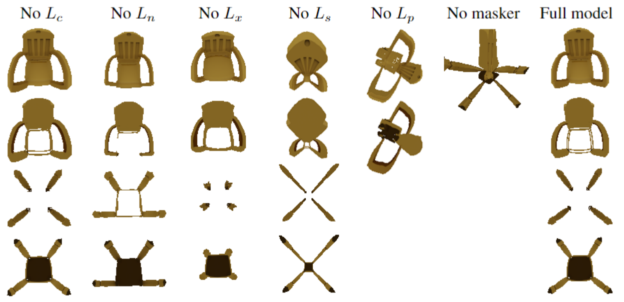

Figure 8. Ablation study: learned texture images with different settings. Each chair has four texture images (shown vertically).

| | No $L_c$ | No $L_n$ | No $L_x$ | No $L_s$ | No $L_p$ | No masker | Full model |
|---|---|---|---|---|---|---|---|
| Cars | 68.5 | 71.7 | **73.0** | 72.8 | 70.6 | 72.0 | 72.7 |
| Chairs | 85.2 | 84.6 | 83.7 | **87.1** | 85.7 | 71.1 | 85.8 |

Table 3. Ablation study: semantic segmentation results in IOU.

| | Triplegangers | ShapeNet cars | ShapeNet chairs |
|---|---|---|---|
| Tex. Fields | 24.59 | 53.09 | 7.03 |
| AUV-Net | **5.69** | **12.11** | **5.33** |

Table 4. Quantitative results of generative models in FID.

the UV coordinates since it encourages one-to-one mapping between surfaces of 3D shapes and the 2D textures; removing it causes overlap in the texture images (Fig. 8 column 3). The prior loss $L_p$ and the masker are critical to our model, as removing them causes severe segmentation and overlap issues (Fig. 8 column 5&6). Also note that different categories have different sensitivity to the loss terms. As shown in Table 3, the car category relies heavily on the colors: removing $L_c$ leads to significant performance drop. In contrast, chair category is less sensitive: removing $L_c$ has no visible effect on the learned texture maps in Fig. 8.

## 4.2. Texture Synthesis

We show texture synthesis results in Fig. 9, and compare them with Texture Fields [33] (TF). Our method generates more details and gets correct alignments, while TF outputs smooth color chunks that are sometimes misaligned. This is because TF uses an MLP to map continuous 3D coordinates into colors, and does not properly disentangle texture and geometry. In contrast, Our method uses a sophisticated 2D generative model trained on aligned texture images to generate the outputs. The texture images aligned by AUV-NET are mostly independent from the actual mesh geometry.

To evaluate the methods quantitatively, we use Fréchet Inception Distance (FID) [23]. We test on 1,000 shapes for each ShapeNet category and 100 shapes for Triplegangers heads. For each test shape, we generate 5 textures, and render each textured shape into 8 views. Results are presented in Table 4, where our method clearly outperforms baseline.

## 4.3. Textured Single-View 3D Reconstruction

We show results in Fig. 10, and compare with TF. For a fair comparison on texture prediction, we use the mesh generated by our method as the output mesh of TF, so that

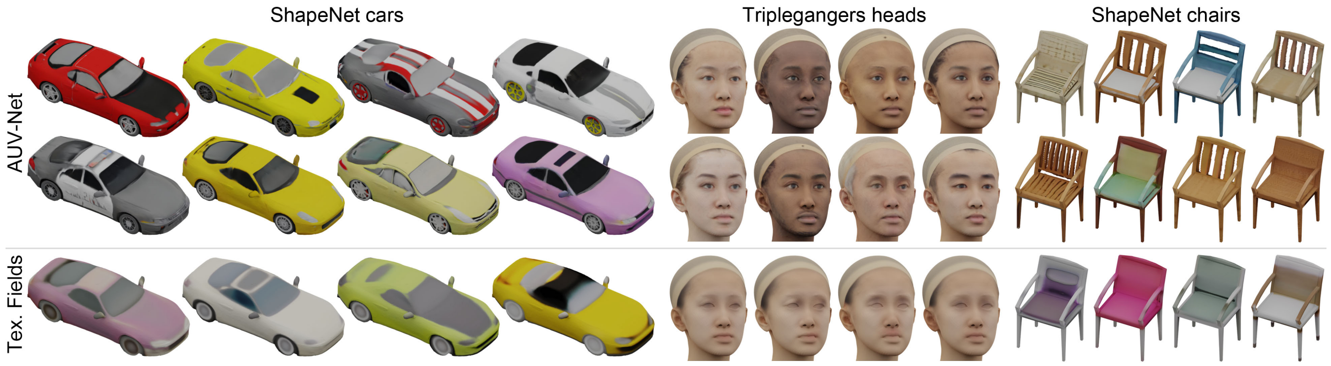

Figure 9. Texture synthesis results. The holes on chairs are hallucinated via texture transparency (alpha channels in the texture images).

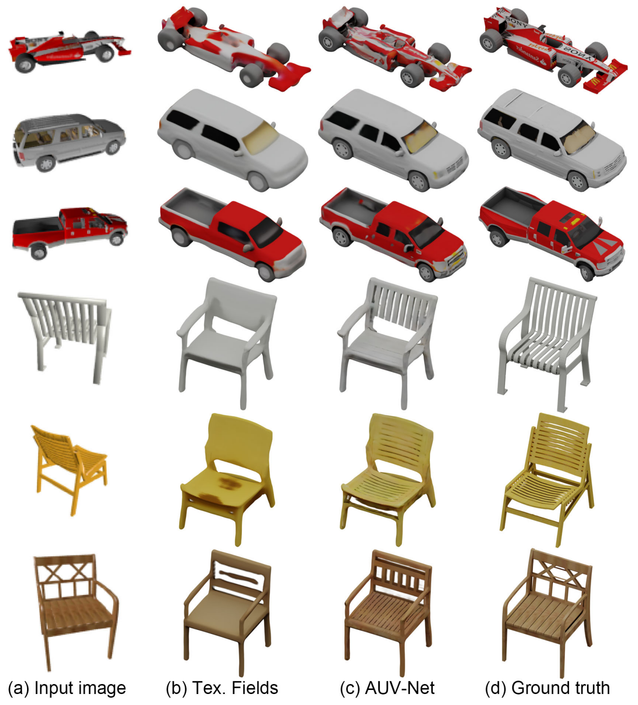

Figure 10. Textured single view reconstruction results. Zoom in to see the details, e.g., wheels of the cars.

| | ShapeNet cars | | | ShapeNet chairs | | |
|---|---|---|---|---|---|---|
| | FID | SSIM | feature-$l_1$ | FID | SSIM | feature-$l_1$ |
| Tex. Fields | 92.89 | **0.897** | 0.219 | 36.89 | **0.855** | 0.193 |
| AUV-Net | **40.85** | 0.894 | **0.186** | **33.26** | 0.853 | **0.189** |

Table 5. Results of textured single view reconstruction.

TF only needs to predict textures of the shapes. The results show that our method produces sharper boundaries and more details in the textures.

In addition to FID evaluated on a set of shapes, we use Structural Similarity Index Measure (SSIM) [46] and feature-$l_1$ [33] to evaluate the quality of each rendered view and then average them, as proposed in TF. Table 5 reports the results, with AUV-Net outperforming the baseline. We observe that both methods overfitted on ShapeNet chairs, and are unable to recover the correct textures of complex test shapes. This is likely due to the fact that the dataset has significant variation for chairs, but has insufficient training examples. In addition, we find that SSIM, which is not semantics-aware, may not be a good evaluation metric when the results are not very close to the ground truth. As shown in Table 5, the SSIM of ours and the baseline are very close, though other metrics show clear differences.

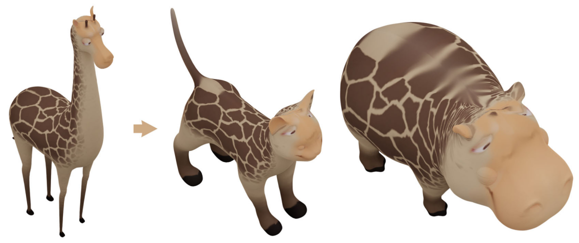
Figure 11. When transferring texture of a cartoon giraffe into other animals, the positions of the eyes are wrong. The cut seam on the hippo is clearly visible, although inpainted.

## 5. Conclusion, Limitations, and Future Work

We introduce the first method to learn aligned texture maps for a set of shapes in an unsupervised manner. We show that alignment can be done with a simple alignment module inspired by PCA. The resulting texture images of our method are well aligned and disentangled from geometry. They have enabled several applications including texture transfer, texture synthesis, and textured single view 3D reconstruction, which we showcase in our experiments.

There are three main limitations of our approach. First, our method does not handle seams that arise when texturing the shape, and only exploits a shared UV mapper network to alleviate the issue. Therefore, seams may become conspicuous for some shapes after transferring textures, as shown in Fig. 11. Second, our method does not always find correct correspondences, especially when the textures are messy, e.g., the animal dataset - one can observe that the eyes are not properly aligned in Fig. 5. Adding weak supervision could help, e.g., annotating two points for the eyes in all training shapes. Third, our method does not handle shapes with complex topology well. In fact, ShapeNet chairs pose a significant difficulty for our method with two basis generators, and we had to use four to avoid overlapping textures. We leave these challenges to future work.

**Acknowledgements.** We thank the reviewers for their valuable comments. This work was completed when the first author was carrying out an internship at NVIDIA.

## References

[1] Renderpeople. https://renderpeople.com/. 6, 3, 4

[2] Triplegangers. https://triplegangers.com/. 6, 2, 4

[3] Turbosquid. https://www.turbosquid.com/. 6, 3, 4

[4] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas J Guibas. Learning representations and generative models for 3D point clouds. In *ICML*, 2018. 1

[5] Anand Bhattad, Aysegul Dundar, Guilin Liu, Andrew Tao, and Bryan Catanzaro. View generalization for single image textured 3d models. In *CVPR*, 2021. 1, 2

[6] Eric R Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis. In *CVPR*, 2021. 2

[7] 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. *arXiv preprint arXiv:1512.03012*, 2015. 6, 3, 4

[8] Bindita Chaudhuri, Nikolaos Sarafianos, Linda Shapiro, and Tony Tung. Semi-supervised synthesis of high-resolution editable textures for 3d humans. In *CVPR*, 2021. 2

[9] Kevin Chen, Christopher B Choy, Manolis Savva, Angel X Chang, Thomas Funkhouser, and Silvio Savarese. Text2shape: Generating shapes from natural language by learning joint embeddings. *ACCV*, 2018. 2

[10] Wenzheng Chen, Jun Gao, Huan Ling, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. In *Advances In Neural Information Processing Systems*, 2019. 1, 2

[11] Zhiqin Chen, Vladimir G. Kim, Matthew Fisher, Noam Aigerman, Hao Zhang, and Siddhartha Chaudhuri. Decor-gan: 3d shape detailization by conditional refinement. *CVPR*, 2021. 1

[12] Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. Bsp-net: Generating compact meshes via binary space partitioning. *CVPR*, 2020. 1

[13] Zhiqin Chen, Kangxue Yin, Matthew Fisher, Siddhartha Chaudhuri, and Hao Zhang. BAE-NET: Branched autoencoder for shape co-segmentation. *ICCV*, 2019. 7

[14] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. *CVPR*, 2019. 1, 6, 2

[15] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3D-R2N2: A unified approach for single and multi-view 3D object reconstruction. In *ECCV*, 2016. 1

[16] Yu Deng, Jiaolong Yang, and Xin Tong. Deformed implicit field: Modeling 3d shapes with learned dense correspondence. In *CVPR*, 2021. 2, 6, 7

[17] Jun Gao, Wenzheng Chen, Tommy Xiang, Clement Fuji Tsang, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Learning deformable tetrahedral meshes for 3d reconstruction. *NeurIPS*, 2020. 2

[18] Lin Gao, Tong Wu, Yu-Jie Yuan, Ming-Xian Lin, Yu-Kun Lai, and Hao Zhang. Tm-net: Deep generative networks for textured meshes. In *Siggraph Asia*, 2021. 1

[19] Lin Gao, Jie Yang, Tong Wu, Yu-Jie Yuan, Hongbo Fu, Yu-Kun Lai, and Hao Zhang. Sdm-net: Deep generative network for structured deformable mesh. *ACM Transactions on Graphics (TOG)*, 38(6):1–15, 2019. 1

[20] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan Russell, and Mathieu Aubry. AtlasNet: A papier-mâché approach to learning 3D surface generation. In *CVPR*, 2018. 1

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 6, 2

[22] Paul Henderson, Vagia Tsiminaki, and Christoph Lampert. Leveraging 2D data to learn textured 3D mesh generation. In *CVPR*, 2020. 1, 2

[23] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In *NeurIPS*, 2017. 7

[24] Krishna Murthy J., Edward Smith, Jean-Francois Lafleche, Clement Fuji Tsang, Artem Rozantsev, Wenzheng Chen, Tommy Xiang, Rev Lebaredian, and Sanja Fidler. Kaolin: A pytorch library for accelerating 3d deep learning research. *arXiv:1911.05063*, 2019. 1

[25] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *ICLR*, 2018. 3

[26] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of StyleGAN. In *CVPR*, 2020. 6, 2

[27] Xueting Li, Sifei Liu, Shalini De Mello, Kihwan Kim, Xiaolong Wang, Ming-Hsuan Yang, and Jan Kautz. Online adaptation for consistent mesh reconstruction in the wild. In *NeurIPS*, 2020. 1, 2

[28] Feng Liu and Xiaoming Liu. Learning implicit functions for topology-varying dense 3d shape correspondence. In *NeurIPS*, 2020. 2, 6

[29] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *ICCV*, 2015. 3

[30] William E. Lorensen and Harvey E. Cline. Marching cubes: a high resolution 3d surface construction algorithm. In *SIGGRAPH*, 1987. 2

[31] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3D reconstruction in function space. In *CVPR*, 2019. 1

[32] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 1, 2

[33] Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. Texture fields: Learning texture representations in function space. In *ICCV*, 2019. 1, 2, 7, 8

[34] Xingang Pan, Bo Dai, Ziwei Liu, Chen Change Loy, and Ping Luo. Do 2d gans know 3d shape? unsupervised 3d shape reconstruction from 2d image gans. In *ICLR*, 2021. 2

[35] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *CVPR*, 2019. 1

[36] Dario Pavllo, Graham Spinks, Thomas Hofmann, Marie-Francine Moens, and Aurelien Lucchi. Convolutional generation of textured 3d meshes. *NeurIPS*, 2020. 1, 2

[37] Amit Raj, Cusuh Ham, Connelly Barnes, Vladimir Kim, Jingwan Lu, and James Hays. Learning to generate textures on 3d meshes. In *CVPR Workshops*, 2019. 1

[38] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. Graf: Generative radiance fields for 3d-aware image synthesis. In *NeurIPS*, 2020. 1

[39] Alla Sheffer, Emil Praun, and Kenneth Rose. Mesh parameterization methods and their applications. *Foundations and Trends® in Computer Graphics and Vision*, 2(2):105–171, 2007. 2

[40] Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. In *NeurIPS*, 2021. 1

[41] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *NeurIPS*, 2020. 2

[42] Yongbin Sun, Ziwei Liu, Yue Wang, and Sanjay E Sarma. Im2avatar: Colorful 3d reconstruction from a single image. *arXiv preprint arXiv:1804.06375*, 2018. 2

[43] Alexandru Telea. An image inpainting technique based on the fast marching method. *Journal of graphics tools*, 9(1):23–34, 2004. 4

[44] Shubham Tulsiani, Nilesh Kulkarni, and Abhinav Gupta. Implicit mesh reconstruction from unannotated image collections, 2020. 1, 2

[45] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *J. Cognitive Neuroscience*, 3(1):71–86, 1991. 3

[46] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004. 8

[47] Shangzhe Wu, Christian Rupprecht, and Andrea Vedaldi. Unsupervised learning of probably symmetric deformable 3d objects from images in the wild. In *CVPR*, pages 1–10, 2020. 2

[48] Fanbo Xiang, Zexiang Xu, Milos Hasan, Yannick Hold-Geoffroy, Kalyan Sunkavalli, and Hao Su. Neutex: Neural texture mapping for volumetric neural rendering. In *CVPR*, 2021. 1

[49] Tarun Yenamandra, Ayush Tewari, Florian Bernard, Hans-Peter Seidel, Mohamed Elgharib, Daniel Cremers, and Christian Theobalt. i3dmm: Deep implicit 3d morphable model of human heads. In *CVPR*, 2021. 1

[50] Li Yi, Vladimir G Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3D shape collections. *Siggraph Asia*, 35(6), 2016. 7

[51] Kangxue Yin, Jun Gao, Maria Shugrina, Sameh Khamis, and Sanja Fidler. 3dstylenet: Creating 3d shapes with geometric and texture style variations. In *ICCV*, 2021. 1, 2

[52] Yuxuan Zhang, Wenzheng Chen, Huan Ling, Jun Gao, Yinan Zhang, Antonio Torralba, and Sanja Fidler. Image gans meet differentiable rendering for inverse graphics and interpretable 3d neural rendering. In *ICLR*, 2021. 1

# AUV-Net: Learning Aligned UV Maps for Texture Transfer and Synthesis

Supplementary Material

This document provides supplementary material for AUV-Net. It contains details regarding network architectures, loss function definitions, training protocols, dataset licenses, and ethics discussions. More qualitative results can be found in https://youtu.be/UTzH8WB-Xl0.

## 1. Implementation details

### 1.1. Detailed network architectures

The detailed architectures of sub-modules in our network are shown in Fig. 12.

### 1.2. Texture transfer for unseen shapes

Given a new shape not in the original training set, we put 100 duplicates of that shape in the training dataset, and then continue training stage 3 for 200 epochs. During training, we fix the weights of the basis generators to avoid affecting the already well-trained texture basis. If the new shape does not have textures, we remove the color loss. It takes about 3 hours to finish training for the new shape on one NVIDIA RTX 3080 Ti GPU.

### 1.3. Image generative model for texture synthesis

We train a StyleGAN2 [26] on the aligned texture images, after stitch multiple texture images of the same object into one single image. For categories with 2 texture images per object, we concatenate the two images horizontally and then resize them to $1024^2$. For chair category with 4 texture images, we put them at the four quadrants of an $1024^2$ image. We further apply Gaussian blur with $\sigma = 1$ on the training images to remove small noise, making the edges slightly smoother, and boosting the performance of generation. Training StyleGAN2 takes around 48 hours on 8 Nvidia Tesla v100 GPUs. During inference, we split each generated image back to multiple ones.

### 1.4. Textured single view reconstruction

The network architecture for this application is shown in Fig. 13. The details of the image decoder is shown in Fig. 12. We follow the implementation of ResNet-18 [21] and IM-Net [14] for the image encoder and the implicit shape decoder, respectively. Note that we generate $512^2$ texture images to reduce training time, and the number of output channels (RGB+alpha) is $C = 8$ for car category with 2 texture images and $C = 16$ for chair category with 4 texture images.

**Training.** We first train our AUV-Net on the 3D training set to obtain aligned texture images and the shape codes. Then we use those as ground truth to train the SVR network in Fig. 13 with MSE losses. Note that the ResNet encoder directly regresses the provided shape codes, therefore the pre-trained UV mapper and Masker are not used during SVR training. The IM-Net decoder also reconstructs shapes directly from the provided shape codes, therefore it can be trained separately without other sub-networks. Pre-training AUV-Net on each ShapeNet category (200 epochs) takes around 36 hours on one NVIDIA RTX 3080 Ti GPU. Training the SVR network (2000 epochs) takes around 36 hours on 8 Nvidia Tesla v100 GPUs.

**Inference.** For each input image, the ResNet encoder predicts a shape code and a texture code. We use the IM-Net decoder to obtain the implicit field of the shape from the predicted shape code, and then use Marching Cubes [30] to obtain the triangle mesh of the shape. Then we feed the vertices and the triangles of the shape into the UV mapper and the masker to obtain the UV maps. The texture images are predicted by the image decoder from the texture code.

## 2. Configurations for different datasets

The main differences between the configurations of different datasets are the number of basis generators, basis generator hyper-parameters (the hidden layer size and the number of basis images), the definition of the prior loss, and the weights of the five loss terms. The basis generator hyper-parameters are different because we reduce the sizes of the networks in certain categories to reduce training time. The other hyper-parameters are set differently according to the properties of the datasets.

### 2.1. Triplegangers [2] heads

We use 2 basis generators. The first basis generator has hidden layer size $G_{dim} = 1024$ (see Fig. 12), and outputs 64 channels (=64 basis images). The second basis generator has hidden layer size $G_{dim} = 128$, and outputs 16 channels. We use different hidden layer sizes and output channels for those basis generators to reduce training time, considering that the second basis generator has significantly less details to represent (see Fig. 4 in the main paper).

The prior loss is defined as

$$L_p = \frac{1}{N}\sum_{i=1}^{N}(m_i - n_i)^2 + (p_i^x - q_i^x)^2 + (p_i^y - q_i^y)^2. \quad (4)$$

where $m_i$ is the masking value predicted by the masker for the $i$-th input point $p_i$. $n_i$ is defined as 1 if the normal of $p_i$ in the z direction is greater than $-0.5$, and 0 otherwise. (The heads are facing z direction, and the top is y direction.) $p_i^x$ is the x coordinate of $p_i$, $q_i^x$ is the x coordinate of $q_i$.

We train the first stage for 10 epochs, with $\{w_c, w_n, w_x, w_s, w_p\} = \{1, 0.5, 100, 100, 1\}$; second stage for 2,000 epochs, with $\{1, 0.5, 1, 1, 0\}$; third stage for 2,000 epochs, with $\{1, 0.5, 100, 100, 0\}$.

### 2.2. RenderPeople [1] human bodies

We use 2 basis generators. Both basis generators have hidden layer size $G_{dim} = 1024$, and output 64 channels.

The prior loss is defined as

$$L_p = \frac{1}{N}\sum_{i=1}^{N}(m_i - n_i)^2 + (p_i^x - q_i^x)^2 + (p_i^y - q_i^y)^2. \quad (5)$$

where $n_i$ is defined as 1 if the normal of $p_i$ in the yz direction is greater than $-0.5$, and 0 otherwise. (The bodies are facing z direction, and the top is y direction.)

We train the first stage for 10 epochs, with $\{1, 0.5, 1, 1, 1\}$; second stage for 2,000 epochs, with $\{1, 0.1, 1, 1, 0\}$; third stage for 2,000 epochs, with $\{1, 0.5, 100, 100, 0\}$.

### 2.3. Turbosquid [3] animals

We use 2 basis generators. Both basis generators have hidden layer size $G_{dim} = 1024$, and output 64 channels.

The prior loss is defined as

$$L_p = \frac{1}{N}\sum_{i=1}^{N}(m_i - n_i)^2 + (p_i^y - q_i^x)^2 + (p_i^z - q_i^y)^2. \quad (6)$$

where $n_i$ is defined as 1 if the normal of $p_i$ in the x direction is greater than 0, and 0 otherwise. (The animals are facing z direction, and the top is y direction.)

We train the first stage for 10 epochs, with $\{0.1, 1, 100, 100, 100\}$; second stage for 2,000 epochs, with $\{0.1, 1, 100, 100, 0\}$; third stage for 2,000 epochs, with $\{0.1, 1, 100, 10, 0\}$.

### 2.4. Turbosquid [3] cars

We use 2 basis generators. The first basis generator has hidden layer size $G_{dim} = 1024$, and outputs 64 channels. The second basis generator has hidden layer size $G_{dim} = 128$, and outputs 16 channels.

The prior loss is defined as

$$L_p = \frac{1}{N}\sum_{i=1}^{N}(m_i - n_i)^2 + (p_i^x - q_i^x)^2 + (p_i^z - q_i^y)^2. \quad (7)$$

where $n_i$ is defined as 1 if the normal of $p_i$ in the y direction is greater than $-0.5$, and 0 otherwise. (The cars are facing inverse z direction, and the top is y direction.)

We train the first stage for 200 epochs, with $\{1, 0.1, 100, 10, 1\}$; second stage for 1,800 epochs, with $\{1, 0.1, 1, 10, 0\}$; third stage for 2,000 epochs, with $\{1, 0.1, 1000, 100, 0\}$. We set $w_p$ to 0 after 10 epochs in the first stage.

### 2.5. ShapeNet [7] cars

We use 2 basis generators. The first basis generator has hidden layer size $G_{dim} = 1024$, and outputs 64 channels. The second basis generator has hidden layer size $G_{dim} = 128$, and outputs 16 channels.

The prior loss is defined as

$$L_p = \frac{1}{N}\sum_{i=1}^{N}(m_i - n_i)^2 + (p_i^x - q_i^x)^2 + (p_i^z - q_i^y)^2. \quad (8)$$

where $n_i$ is defined as 1 if $p_i^y > 0$ or the normal of $p_i$ in the y direction is greater than $-0.5$, and 0 otherwise. (The cars are facing x direction, and the top is y direction.)

We train the first stage for 20 epochs, with $\{1, 0.1, 10, 10, 1\}$; second stage for 40 epochs, with $\{1, 0.1, 1, 10, 0\}$; third stage for 140 epochs, with $\{1, 1, 100, 100, 0\}$. We set $w_p$ to 0 after 5 epochs in the first stage.

### 2.6. ShapeNet [7] chairs

We use 4 basis generators. All basis generators have hidden layer size $G_{dim} = 512$, and output 64 channels.

Since there are four texture images, we modify the masker to output four channels. We only input point coordinates and the shape code to the masker, and let it predict the point normal $n_i^{pred}$ (not necessarily unit normal) and the mask $m_i^{pred}$. We then compute the dot product between the predicted normal and the ground truth normal to get a second mask $m_i^n = sigmoid(n_i^{pred} \cdot n_i^{gt})$. The final output masks are

$$\begin{bmatrix} m_i^a & m_i^c \\ m_i^b & m_i^d \end{bmatrix} = \begin{bmatrix} m_i^n \\ 1 - m_i^n \end{bmatrix} \cdot [m_i^{pred} \quad 1 - m_i^{pred}] \quad (9)$$

The prior loss is defined as

$$L_p = \frac{1}{N}\sum_{i=1}^{N}(m_i^a - s_i^a)^2 + (m_i^b - s_i^b)^2 + (m_i^c - s_i^c)^2 + (m_i^d - s_i^d)^2 + (t_i^x - q_i^x)^2 + (t_i^y - q_i^y)^2. \quad (10)$$

where $m_i^a$,$m_i^b$,$m_i^c$,$m_i^d$ are the masking values predicted by the masker. $s_i^a$,$t_i^x$,etc. are defined as follows. The chairs are facing x direction, and the top is y direction. The chairs are normalized in a unit cube centered at origin. $n_i$ is the unit normal of $p_i$.

$$\begin{aligned}
p^y_{max} &= \max_i p^y_i, \\
p^{y\prime}_i &= p^y_i - p^y_{max} - 0.05, \\
m_i{}' &= \begin{cases} 1 & \text{if } p^x_i n^x_i + p^{y\prime}_i n^y_i + p^z_i n^z_i < 0, \\ 0 & \text{otherwise.} \end{cases} \\
p^y_{seat} &= \max_{i \in \{i \mid p^x_i > 0 \wedge p^x_i < 0.1 \wedge p^z_i > -0.05 \wedge p^z_i < 0.05\}} p^y_i \\
m_i{}'' &= \begin{cases} 1 & \text{if } p^y_i > p^y_{seat} - 0.2, \\ 0 & \text{otherwise.} \end{cases} \\
\begin{bmatrix} s^a_i & s^c_i \\ s^b_i & s^d_i \end{bmatrix} &= \begin{bmatrix} m_i{}' \\ 1 - m_i{}' \end{bmatrix} \cdot [m_i{}'' \quad 1 - m_i{}''] \\
d_i &= \sqrt{p^{x\,2}_i + p^{z\,2}_i + 4(p^y_i - p^y_{seat})^2} \Big/ \sqrt{p^{x\,2}_i + p^{z\,2}_i} \\
t^x_i, t^y_i &= p^x_i d_i, p^z_i d_i
\end{aligned} \tag{11}$$

We train the first stage for 50 epochs, with $\{1, 1, 10, 100, 100\}$; second stage for 50 epochs, with $\{1, 1, 10, 10, 0\}$; third stage for 100 epochs, with $\{1, 1, 100, 100, 0\}$. We set $w_p$ to 0 after 5 epochs in the first stage.

## 3. Dataset licenses

In the paper, we use 3D assets of cars and chairs from ShapeNet [7], animals and cars from Turbosquid [3], human faces from Triplegangers [2], human bodies from Renderpeople [1]. The licenses for using Renderpeople, Triplegangers and TurboSquid datasets were obtained through commercial agreements. The license information of ShapeNet is provided on its website [1]. We carefully inspected the datasets we use and did not find identifiable information or offensive content. To run the comparison with baseline methods, we use the source code provided by the authors on GitHub.

## 4. Ethics discussions

### 4.1. Potential negative societal impacts

We present a generic method for synthesizing textures for a variety of classes of 3D objects. It does also handle 3D human face/body shapes and textures. Our method can create 3D avatars for users by transferring textures to 3D meshes, or reconstructing textured meshes from input photos. Like any other machine learning models, AUV-NET is prone to biases imparted through training data which requires an abundance of caution when applied to sensitive applications. It is not recommended in off-the-shelf settings where privacy or erroneous results can lead to potential misuse or any harmful application. For purposes of real deployment, one would need to carefully inspect and de-bias the dataset to depict the target distribution of a wide range of possible lighting conditions, clothing, skin tones, or at the intersection of race and gender.

### 4.2. Personal data/Human subjects

Our paper uses 3D scans of human faces and bodies obtained from Triplegangers and Renderpeople, respectively. The data collection and ethics approvals were taken care of by the dataset providers. More information about the dataset can be found on the websites of data providers.

[1]https://shapenet.org/terms

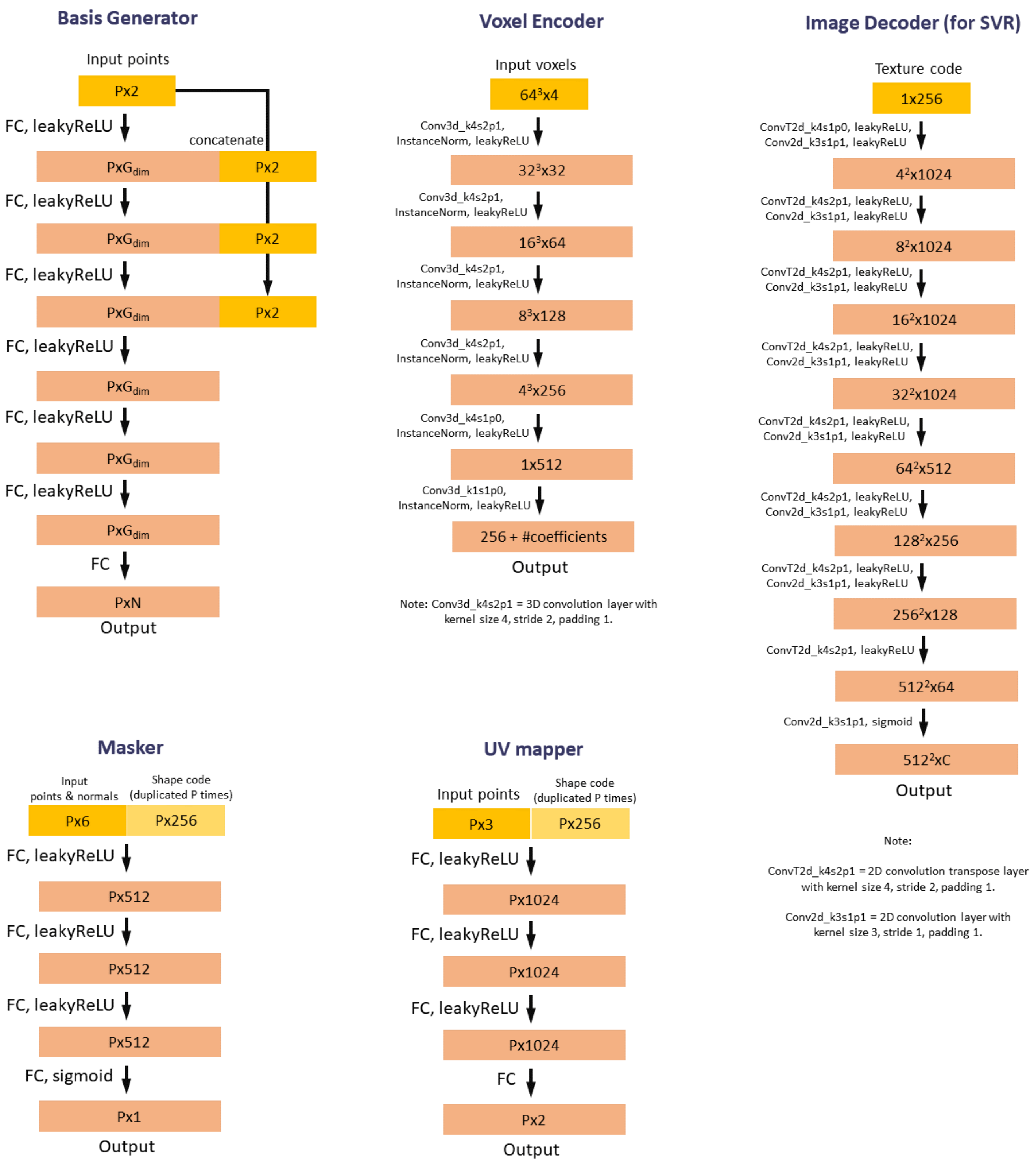
Basis Generator
Input points
Px2
FC, leakyReLU
concatenate
$\text{PxG}_{\text{dim}}$
Px2
FC, leakyReLU
$\text{PxG}_{\text{dim}}$
Px2
FC, leakyReLU
$\text{PxG}_{\text{dim}}$
Px2
FC, leakyReLU
$\text{PxG}_{\text{dim}}$
FC, leakyReLU
$\text{PxG}_{\text{dim}}$
FC, leakyReLU
$\text{PxG}_{\text{dim}}$
FC
PxN
Output
Voxel Encoder
Input voxels
$64^3$x4
Conv3d_k4s2p1, InstanceNorm, leakyReLU
$32^3$x32
Conv3d_k4s2p1, InstanceNorm, leakyReLU
$16^3$x64
Conv3d_k4s2p1, InstanceNorm, leakyReLU
$8^3$x128
Conv3d_k4s2p1, InstanceNorm, leakyReLU
$4^3$x256
Conv3d_k4s1p0, InstanceNorm, leakyReLU
1x512
Conv3d_k1s1p0, InstanceNorm, leakyReLU
256 + #coefficients
Output
Note: Conv3d_k4s2p1 = 3D convolution layer with kernel size 4, stride 2, padding 1.
Image Decoder (for SVR)
Texture code
1x256
ConvT2d_k4s1p0, leakyReLU, Conv2d_k3s1p1, leakyReLU
$4^2$x1024
ConvT2d_k4s2p1, leakyReLU, Conv2d_k3s1p1, leakyReLU
$8^2$x1024
ConvT2d_k4s2p1, leakyReLU, Conv2d_k3s1p1, leakyReLU
$16^2$x1024
ConvT2d_k4s2p1, leakyReLU, Conv2d_k3s1p1, leakyReLU
$32^2$x1024
ConvT2d_k4s2p1, leakyReLU, Conv2d_k3s1p1, leakyReLU
$64^2$x512
ConvT2d_k4s2p1, leakyReLU, Conv2d_k3s1p1, leakyReLU
$128^2$x256
ConvT2d_k4s2p1, leakyReLU, Conv2d_k3s1p1, leakyReLU
$256^2$x128
ConvT2d_k4s2p1, leakyReLU
$512^2$x64
Conv2d_k3s1p1, sigmoid
$512^2$xC
Output
Note:
ConvT2d_k4s2p1 = 2D convolution transpose layer with kernel size 4, stride 2, padding 1.
Conv2d_k3s1p1 = 2D convolution layer with kernel size 3, stride 1, padding 1.
Masker
Input points & normals
Shape code (duplicated P times)
Px6
Px256
FC, leakyReLU
Px512
FC, leakyReLU
Px512
FC, leakyReLU
Px512
FC, sigmoid
Px1
Output
UV mapper
Input points
Shape code (duplicated P times)
Px3
Px256
FC, leakyReLU
Px1024
FC, leakyReLU
Px1024
FC, leakyReLU
Px1024
FC
Px2
Output

Figure 12. Detailed network architectures.

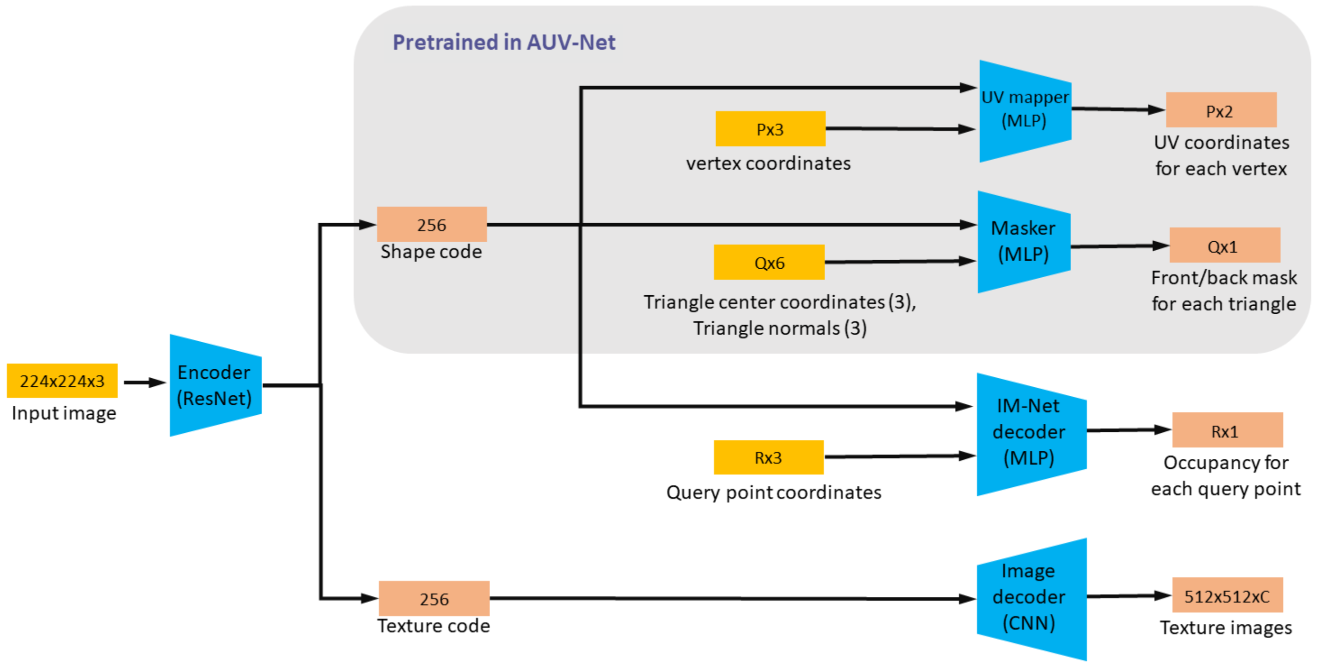

Figure 13. Architecture of network for textured single view reconstruction. Note that the UV mapper and the Masker are from a pretrained AUV-Net trained on the training shapes, and they are used to predict the UV mapping for the predicted shape by the IM-Net decoder. The shape codes and the texture images produced by the pretrained AUV-Net are used in this network as the ground truth.