HyperColor: A HyperNetwork Approach for Synthesizing Autocolored 3-D Models for Game Scenes Population

Ivan Kostiuk, Przemysław Stachura, Sławomir K. Tadeja, Tomasz Trzciński, Senior Member, IEEE, and Przemysław Spurek

Abstract—Designing a 3-D game scene is a tedious task that often requires a substantial amount of work. Typically, this task involves the synthesis and coloring of 3-D models within the scene. To lessen this workload, we can apply machine learning to automate some aspects of the game scene development. Earlier research has already tackled automated generation of the game scene background with machine learning. However, model autocoloring remains an underexplored problem. The automatic coloring of a 3-D model is a challenging task, especially when dealing with the digital representation of a colorful, multipart object. In such a case, we have to “understand” the object’s composition and coloring scheme of each part. Moreover, existing single-stage methods have their caveats. We address these limitations by proposing a two-stage training approach to synthesize autocolored 3-D models. In the first stage, we obtain a 3-D point cloud representing a 3-D object, while in the second stage, we assign colors to points within such a cloud. Next, we generate a 3-D mesh in which the surfaces are colored based on the interpolation of colored points representing vertices of a given mesh triangle. This approach allows us to develop a smooth coloring scheme.

Index Terms—3-D models, 3-D point clouds, autoencoder, colored models, deep learning, hypernetwork, machine learning.

I. INTRODUCTION

ONE of the most tedious and mundane tasks in 3-D game development is the preparation of the 3-D scene content for individual game levels [1]–[3]. Such content usually involved generation, coloring, and placement of the 3-D models populating the given scene [1].

One of the ways to reduce the burden of the level design is to automate the process of scene content generation with procedural and machine learning (ML) methods [1]–[7]. We can consider automatic content creation as a two-step process: 1) generating plausible and diversified 3-D models, and 2) coloring of these models.

Earlier work addresses terrain creation [1]–[3], [6], and 3-D model generation [8]–[17]. However, the 3-D model autocoloring [18], remains an underexplored problem [4].

To stimulate research in this area, we present in this article a hypernetwork-based [20] method for simultaneous synthesis and autocoloring of diversified 3-D models [4]. This task requires knowledge about the type of object we process, including its real-world surfaces coloring. The latter task becomes even more challenging when the object in question is composed of several parts requiring different coloring schemes [4].

Existing methods attempt to remedy this issue by segmenting the 3-D object [21] and training a neural network to single-color individual object’s parts. However, in such a case, the resulting mesh has a more complex structure than needed or requires additional mesh assembly procedures.

Contrary to this approach, we train a deep learning model to automatically color the 3-D models without the need to decompose it into individual parts. Our HyperColor1 method allows for autogeneration and autocoloring of diversified 3-D models. We use a two-stage training process to obtain colored 3-D point clouds which are later converted into 3-D meshes by leveraging a triangulation trick [20]. These 3-D objects can be easily incorporated into the 3-D background rendering plausible game scenes (see Figs. 1 and 2).

In the rest of this work, we first give an overview of the relevant literature. Next, we describe in detail our two-stage

1We made our implementation. [Online]. Available: https://github.com/KostiukIvan/HyperColor

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
Fig. 1. Hypernetwork generates and autocolors 3-D models of chairs and cars, populating game scenes. (a) and (b) The scene background is constructed out of a photogrammetric model of the Great Drawing Room at The Hallwyl Museum, while (c) and (d) backgrounds are generated based on the photogrammetric model of the Rempart Walls scenery - Fougères castle landscape (see https://sketchfab.com/noxfcna). Both backgrounds come from [19] and are presented in the editor of Unity game engine.

Fig. 2. Hypernetwork-generated 3-D plane object. The scene background was constructed out of photogrammetric model of Sackville College 3-D Mesh made by the Aircam Surveys Ltd. downloaded through [19] and presented in the editor of Unity game engine.

approach to generating plausible 3-D auto-colored meshes using hypernetworks. Lastly, we present the results of a comparison between our two-stage method against the traditional one-stage approach (see Table III) and referenced method.

II. RELATED WORK

We divide the related work into three parts. First, we describe the existing methods for generating 3-D shapes and objects. Second, we discuss models producing colored 3-D objects. Finally, to provide a broader context, we present a brief description of procedural methods used in generating game scenes and entire levels.

A. Generating 3-D Objects

Point clouds are among the most popular digital representations of various 3-D objects and shapes, widely used in light detection and ranging (LiDAR) methods and depth cameras. Complex perception tasks that use point clouds, such as localization or object recognition, typically treat these representations as fixed-dimensional matrices, which require subsampling of the cloud or other preprocessing [22]–[26]. In practice, such an approach is often limiting as the complexity of 3-D point clouds varies across object types, and some objects need more points to represent their details than the others.

To remedy this issue, PointFlow [27] proposes to learn a two-level hierarchy of distributions where the first level is the distribution of shapes, and the second is the distribution of points from the object’s surface. This formulation allows sampling an arbitrary number of points from a given shape. Moreover, PointFlow [27] applies normalizing flow [28] to model surface of the object.

Spurek et al. [17], [20] proposed to use a hypernetwork architecture to model the distributions of shapes. Instead of producing a fixed number of points, hypernetwork generates many neural networks, a single network per object. Such models take an element from the prior distribution (uniform distribution on 3-D ball) and transfer it to the object’s surface. Such an approach allows us to produce as many points as we need. We can sample an arbitrary number of points and transfer them using a target network.

Another technique models a shape by learning the gradient field of its log-density [29]. Finally, Sun et al. [25] used autoregressive architecture to model the distribution of 3-D
points, while in [30] the authors leverage a generative adversarial network (GAN) architecture for the same purpose. All the above methods work on raw, uncolored 3-D point clouds.

B. Generating Colored 3-D Objects

Generation of colored 3-D point clouds was explored to a much lesser extent than shape generation. Traditional colorization methods rely on human input that prevents automatization of the coloring process [31]–[33].

To solve this problem, previous works tried to automate the coloring task with ML. Cao and Nagao [34] and Liu et al. [35] presented methods based on the pix2pix [36] pipeline using cGAN. The generator attempts to predict the color of each point using PointNet [37], and the PointNet-based discriminator attempts to judge artificial color from the generator or real ground truth color.

Arshad and Beksi [4] presented a conditional GAN (cGAN) that creates colored, dense 3-D point clouds. A point transformer progressively grows the network. Thus, every training iteration evolves a point vector into a point cloud of increasing resolution. However, the authors evaluate the performance of their method in terms of shapes quality.

cGAN-based approach was also used in [38]. To achieve 3-D point cloud colorization, the colors are estimated by PointNet++ [39] and rendered into 2-D images. Next, the network is trained by minimizing the distance between the real color and artificial color.

All the above methods solve the problem of coloring the 3-D point cloud. In this article, we introduce a generative model that produces 3-D point clouds with colors. Furthermore, we can create meshes with colors.

C. Procedural Game Scenes Generation

The idea of using procedural tools to generate game scenes or entire levels is not new [1]–[3], [6], [7]. These methods were used to generate indoor (e.g., [2], [3]) and outdoor (e.g., [1], [6]) environments alike. As a generation of the game levels is not the main goal of our work, we offer only a small selection of such earlier research carried out by other authors.

In the case of the outdoor level-generation method, Huijser et al. [1] presented a technique for generating large-scale neutral game levels using natural systems. The substantial advantage of using this approach is the ability to edit the resulting scene after its initial generation. A brief overview of procedural terrain generation can be found in [6].

Regarding the indoor environments, Linden et al. [2] surveyed a content generation method for dungeons and catacombs systems. Another example was given by Rymon Lipinski et al. [3], where their method of procedural generation of room systems and corridors in 2-/3-D games was presented using minimum spanning trees.

Moreover, Summerville et al. [5], discussed the advantages ML-based approaches have over standard procedural methods. For instance, ML is well suited to generate various elements of the game levels (e.g., scenes, models, or game mechanics).

Furthermore, we can also use its statistical nature to analyze the generation outcomes, i.e., the resulting game level and its components. For instance, Park et al. [7] provided an example of applying deep convolutional generative adversarial networks (DCGANs) to generate educational game levels. The authors remarked that their approach offers higher solvability with a cost of a slight decrease in the novelty of game levels.

III. AUTOCOLORED 3-D OBJECT GENERATION

In this work, we introduce a two-stage training process that aims at obtaining autocolored 3-D point clouds followed by applying the triangulation trick to produce autocolored 3-D meshes.

A. General Idea

To obtain autocolored 3-D meshes of objects, we employ a two-stage approach. In the first stage, we produce a reconstruction of the object’s shape (see Part A in Fig. 3). Thanks to classical autoencoder architecture, we obtain a 3-D point cloud on the surface of an object. As a result, we are able to produce an uncolored mesh.

Next, we extend our pipeline with a coloring module (see Part B in Fig. 3). To effectively implement colorization, we have to process separately every single element from the object’s surface. Therefore, we use a continuous representation of the surface. In our case, we have used HyperCloud [20], however, our method is agnostic to a 3-D point cloud generative model.

It can work with other methods, such as PointFlow [27] or HyperFlow [17] as well. The main idea is to represent a 3-D object as a neural network, which transfers uniform distribution on a 3-D sphere into the 3-D object’s surface (see Part A in
be a given 3-D point cloud. Most of existing method works only on coordinates, therefore, we will use \( X^{[0:3]} = \{X^{[0:3]}_i\}_{i=1,...,n} = \{(x_i, y_i, z_i)\}_{i=1,...,n} \).

In general, the autoencoder transports the data through latent space \( Z \subseteq \mathbb{R}^D \) and build reconstruction. The architecture consists of an encoder \( \mathcal{E} : \mathcal{X} \rightarrow \mathcal{Z} \) and decoder \( \mathcal{D} : \mathcal{Z} \rightarrow \mathcal{X} \), which minimizes the reconstruction error between \( \hat{X}^{[0:3]} \) and its reconstructions \( D(\mathcal{E}(\mathcal{X}^{[0:3]})) \).

We use two distance measures for 3-D point clouds, namely, the Earth Mover’s (Wasserstein) Distance [40] and Chamfer pseudodistance (CD) [41].

Earth Mover’s Distance (EMD) [40] is a metric between two distributions based on the minimal cost that must be paid to transform one distribution into the other. For two equally sized subsets \( X_1 \subset \mathbb{R}^3 \) and \( X_2 \subset \mathbb{R}^3 \) their EMD is defined as

\[
EMD(X_1, X_2) = \min_{\phi: X_1 \rightarrow X_2} \sum_{x \in X_1} c(x, \phi(x))
\]

where \( \phi \) is a bijection and \( c(x, \phi(x)) \) is the cost function that can be defined as

\[
c(x, \phi(x)) = \frac{1}{2}||x - \phi(x)||^2.
\]

CD [41] measures the squared distance between each point in one set to its nearest neighbor in the other set

\[
CD(X_1, X_2) = \sum_{x \in X_1} \min_{y \in X_2} ||x - y||^2 + \sum_{y \in X_2} \min_{x \in X_1} ||x - y||^2.
\]

The task of simultaneous reconstructing of the positions and their colors is nontrivial, as there is a tradeoff between the reconstruction and colorization quality. In reconstruction loss for colored 3-D objects, we have to calculate the distance between elements from s6-D space, of which the first three coordinates describe positions and the last three express color. When using Chamfer distance as a cost function, we are simultaneously working in both 3-D space and color space. Therefore, we propose a two-stage strategy to address this issue. First, we produce reconstructions, and then we add colors on top of them. In the experimental section, we compare our model with the one-stage approach considered a baseline (see Table III).

Fig. 3. Thanks to such a solution, we can use the triangulation trick (see Part A in Fig. 5) to generate 3-D meshes without colors. Subsequently, in the second stage, we train the colorization module. To do this, we prepare an additional neural network, which transfers the sample from a uniform distribution into an red–green–blue (RGB) color space. The second stage only produces RGB colors for points obtained from the sample (see Part B in Fig. 3). Our approach allows producing colors for any point from the surface without changing the position of the reconstructed points. Furthermore, we can combine it with the triangulation trick [20] to produce autocolored 3-D meshes (see Part B in Fig. 5).

In the subsequent sections, we present our method for generating autocolored 3-D models using a two-stage approach. First, we introduce generative models trained directly on 3-D point clouds data to generate noncolored 3-D objects. Second, we extend this pipeline by introducing the module for 3-D model colorization (see Fig. 4). In consequence, our method allows the construction of autocolored 3-D meshes of chosen objects (see Fig. 7).

**B. Generative Models for 3-D Point Clouds**

Let us start with the autoencoder architecture for 3-D point cloud. Let \( X = \{X_i\}_{i=1,...,n} = \{(x_i, y_i, z_i, r_i, g_i, b_i)\}_{i=1,...,n} \)
We can modify the classical autoencoder to become a generative model. To that end, we change the cost function that forces the model to be generative, i.e., ensures that the data transported to the latent space comes from the prior distribution (typically Gaussian) [42], [43]. Thus, the generative autoencoder model uses reconstruction loss and the distance of a given sample represented in the latent space from the prior distribution.

In our work, we use variational autoencoders (VAE) [42]. Such a model uses variational inference [42] to ensure that the data transported to latent space \( Z \) are distributed according to standard normal density, we add the distance from standard multivariate normal density distribution

\[
\text{cost}(X; \mathcal{E}, D) = \text{Err}(X; D(\mathcal{E}X)) + \lambda D_{KL}(\mathcal{E}X, N(0, I))
\]

where \( D_{KL} \) is the Kullback–Leibler divergence [44] and \( \text{Err} \) is a reconstruction cost. The first part is responsible for the generative ability and the second one is for reconstruction.

C. Hypernetwork

We define hypernetworks [45] as neural models that generate weights for a separate target network trained to solve a specific task.

Since point clouds contain a number of data points, a method dedicated for such a task must be permutation invariant. In PointNet [37], the authors proposed architecture that works with different sizes of 3-D point clouds as input to the neural network. However, working with varying outputs sizes poses a challenge that PointNet is not designed to solve. One of the possible solutions is to use a hypernetwork. Instead of producing a fixed-size output, we create a small neural network--called target network--to produce an output of any size understood as a number of points.

The target network is not trained directly. Instead, we use a hypernetwork that returns weights to the corresponding target network.

D. Hypercloud

Spurek et al. [20] introduced the HyperCloud technique, which uses hypernetwork to produce a continuous representation of 3-D objects. Instead of generating objects directly with the decoder [24], the HyperCloud uses parameterization of the 3-D object’s surface as a function transferring uniform distribution on a 3-D ball into the surface of an object.

In HyperCloud, instead of producing 3-D point clouds, we generate many neural networks, i.e., we make a different neural network for each object class. Thus, in practice, we have one neural network architecture, and the model produces different weights.

In HyperCloud, we model function \( T_\theta : \mathbb{R}^3 \rightarrow \mathbb{R}^3 \) (neural network with weights \( \theta \)), which takes an element from the prior distribution \( P \) and transfers it to a point on the surface of the object.

Such an approach allows producing as many points as we need. For example, we can sample an arbitrary number of points from the uniform distribution of the unit ball and transfer them by the target network onto the object surface. Furthermore, we can produce a continuous mesh representation of an object utilizing the triangulation trick. All elements from the ball are transformed into a 3-D object. In consequence, the unit sphere is transformed into the surface of the object.

Now we can produce meshes without a secondary rendering procedure. It is obtained by simply feeding our neural network with the vertices of a sphere mesh, as shown in Part A in Fig. 5. As a result, we obtain high-quality 3-D meshes.

The potentially limiting factor related to the triangulation trick is related to using mesh on a sphere. The object’s mesh is topologically coherent with the sphere and does not model detailed geometry like holes in the arms of a chair or the triangle-like connection between the plane’s wings and the main body.

We do not train target network directly. We use a hypernetwork \( H_\phi : \mathbb{R}^3 \rightarrow \theta \), which takes a point cloud \( X_{[0:3]} \subset \mathbb{R}^3 \) and return weights \( \theta \) to the target network \( T_\theta \). In such a framework, a point cloud \( X_{[0:3]} \) is parametrized by a function

\[
T((x, y, z); \theta) = T((x, y, z); H_\phi(X)).
\]

Our goal is to train the weights \( \phi \) of the hypernetwork. For this purpose, we minimize the distance between point clouds Chamfer distance [41]. We take an input point cloud \( X_{[0:3]} \subset \mathbb{R}^3 \) and pass it to \( H_\phi \). The hypernetwork returns weights \( \theta \) to the target network \( T_\theta \). Next, the input point cloud \( X_{[0:3]} \) is compared with the output from the target network \( T_\theta \). We sample the correct number of points from the prior distribution and transfer them by the target network.

E. Extending Pipeline With Coloring Module

In the second stage, we color code to the previously produced 3-D point clouds. In turn, this model architecture uses objects generated in the first stage. We use a two-stage strategy since we have to solve the tradeoffs between reconstruction and colorization. In the experimental section, we compare our model with a single-stage strategy which we consider our baseline, see Table III.

In our second stage, we model function \( C_\eta : \mathbb{R}^3 \rightarrow \mathbb{R}^3 \) (neural network with weights \( \eta \)), which takes an element from a uniform distribution and transfers it into the RGB color-space of reconstructed points \( T_\theta(X_{[0:3]}) \). Here, we use reconstruction in LAB format [46] and render images in RGB.

We use one sample from the 3-D uniform distribution and two functions: \( T_\theta \) and \( C_\eta \). The first one produces points on the surface, and the second one produces colors for such points. Since we use one sample from the 3-D uniform distribution, we share information between shapes and colors.

Analogically to the first stage, we do not train target network \( C_\eta \) directly. We use a hypernetwork \( H_\psi : \mathbb{R}^6 \supset X \rightarrow \eta \), which takes a point cloud with colors \( X \subset \mathbb{R}^6 \) and returns weights \( \theta \).
Fig. 6. Graphs present the behaviors cost function during training. As we can see in the first phase, we have the best score for airplanes. Likewise, we can observe the best score for the sofa dataset in the second stage of training.

TABLE I
EVALUATION OF OUR APPROACH AND THE PCGAN METHODS IN THE GENERATIVE TASK ONLY ON THREE FIRST COORDINATES (POSITIONS)

| Measure | MMD-CD ↓ | COV-CD ↑ | NN-CD ↓ | JSD-P ↓ |
|---------|----------|----------|---------|---------|
| PCGAN   | 0.00236  | 0.961    | 0.732   | 0.173   |
| HyperColor | 0.00050  | 0.555    | 0.794   | 0.235   |
| PCGAN   | 0.00337  | 0.245    | 0.982   | 0.235   |
| HyperColor | 0.00113  | 0.525    | 0.854   | 0.231   |
| PCGAN   | 0.00992  | 0.321    | 0.874   | 0.265   |
| HyperColor | 0.00396  | 0.562    | 0.613   | 0.091   |
| PCGAN   | 0.01225  | 0.292    | 0.959   | 0.360   |
| HyperColor | 0.00299  | 0.606    | 0.650   | 0.358   |
| PCGAN   | 0.00990  | 0.491    | 0.882   | 0.308   |
| HyperColor | 0.00946  | 0.503    | 0.637   | 0.327   |
| PCGAN   | 0.01704  | 0.304    | 0.965   | 0.294   |
| HyperColor | 0.0165   | 0.295    | 0.905   | 0.469   |

As we can see, our HyperColor gives better outcomes in each category (19/24). The best scores are in bold.

Then, we take the point cloud \( X \subset \mathbb{R}^6 \) and pass it to \( H_\psi \). The hypernetwork returns weights \( \eta \) to target network \( C_\eta \) which, in turn, produces colors for the reconstructed point cloud. Since we have a position from the first stage, we can use the mean squared error (mse)

\[
\text{MSE}(A, B) = \frac{1}{n} \sum_{i=1}^{n} (a_i - b_i)^2,
\]

to train the second stage (i.e., the coloring stage). The second stage does not have to be invariant to permutations. We simply color each point separately.

Our full procedure of reconstruction consists of four high-level steps that further split into substeps where appropriate as follows.

1) Take 3-D point cloud \( X \subset \mathbb{R}^6 \).
As we can see, our two-stage strategy gives essentially better results (6/6). The best scores are in bold.

2) Sample \( S \) from a uniform distribution on sphere: we draw as many points as we want to reconstruct.

3) Reconstruct shape:
   a) pass \( X_{[0,3]} \) to hypernetwork \( H_\phi \);
   b) \( H_\phi \) returns weights to the target network \( T_\theta \);
   c) transfer \( S \) by target network \( T_\theta \) to produce object reconstruction.

4) Reconstruct colors:
   a) pass \( X \) to hypernetwork \( H_\psi \);
   b) \( H_\psi \) returns weights to the target network \( C_\eta \);
   c) transfer \( S \) by target network \( C_\eta \) to produce colors for object reconstruction.

Once we have two point clouds, i.e., the original one and the reconstructed one produced in our two-stage process, we need to find the alignment between individual points. To that end, we find the closest element from the original object for each point from the reconstructed point cloud. We use a k-nearest neighbor [47] algorithm for such a task.

To produce colored meshes from the colored 3-D point cloud, we use the triangulation trick (see Fig. 5). First, we train a neural network, which transfers samples from a uniform sphere distribution into objects. In consequence, the unit sphere is transformed into the border of our dataset (mesh). The second stage produces colors for vertex from the mesh. We obtain the final coloring scheme by interpolation of colors from vertex \( k \).

Finally, we use Blender’s Vertex Paint module to visualize a colored mesh, see Fig. 8. It allows setting the color of vertices directly. Such an approach results in faces having a gradient calculated using their assigned vertex colors. Then we can obtain a textured object through the process of baking vertex color into an image.
IV. EXPERIMENTS

In this section, we describe the experimental results of the proposed generative models in various tasks, including 3-D points mesh generation and interpolation. First, we show that our model inherited reconstruction and generative capabilities from HyperCloud model. Then we show that we are able to produce continuous mesh representation with colors.

A. 3-D Point Cloud Generation Capabilities

Following the methodology for evaluating generative fidelity and diversification among samples provided in [48] and [27], we utilize the following criteria for evaluation: Jensen–Shannon divergence, coverage, minimum matching distance, and 1-nearest neighbor accuracy.

Jensen–Shannon Divergence (JSD): A measure of the distance between two empirical distributions $P$ and $Q$, defined as

$$
JSD(P\|Q) = \frac{KL(P\|M) + KL(Q\|M)}{2},
$$

where $M = \frac{P + Q}{2}$.

Coverage (COV): A measure of generative capabilities in terms of richness of generated samples from the model. For two point cloud sets $X_1, X_2 \subset \mathbb{R}$ coverage is defined as a fraction of points in $X_2$ that are in the given metric the nearest neighbor to some points in $X_1$.

Minimum Matching Distance (MMD): Since COV only takes the closest point clouds into account and does not depend on the distance between the matching, in literature additional metric was introduced. For point cloud sets $X_1, X_2$ MMD is a measure of similarity between point clouds in $X_1$ to those in $X_2$.

1-Nearest Neighbor Accuracy (1-NNA): It is a testing procedure characteristic for evaluating GANs. We consider two sets: set $S_g$ composed of generated point clouds and set of test, reference point clouds, $S_r$. We pick some generated point cloud $X$ from $S_g$ and find the corresponding nearest neighbor in $S_{-X} = S_r \cup S_g - \{X\}$, the set that aggregates both training and sampled shapes excluding the considered point cloud $X$. The 1-NNA is the leave-one-out accuracy of the 1-NN classifier

$$
1 - \text{NNA} = \frac{\sum_{X \in S_g} 1[N_X \in S_g] + \sum_{Y \in S_r} 1[N_Y \in S_r]}{|S_g| + |S_r|}.
$$

For each sample, the 1-NN classifier classifies it as coming from $S_r$ or $S_g$ according to the label of its nearest sample. The perfect situation occurs when the classifier cannot distinguish between real and generated point clouds, which means that the value of the criterion is close to 50%.

We compare our HyperColor with progressive conditional generative adversarial network (PCGAN) [4]. First, we use only three first components to verify only generative capability in the case of shapes. As we can see in Table I, HyperColor has better score in most of classes.
In the second experiment, we verify generative properties in color space. We first sorted samples from objects according to the three first components, i.e., Cartesian coordinates. After that, we use classical mse distance. As we can see in Table II our model obtains better results in the case of Chair, Car, Sofa, and Table classes and worse outcomes in the Airplane class. In Fig. 6 we present training curves. We selected these classes of objects because of an analogy with [20].

B. 3-D Point Cloud Reconstruction Capabilities

We now evaluate how well our model encodes a point cloud. We conducted an autoencoding task for 3-D point clouds from three categories from ShapeNet dataset (i.e., Airplane, Car, and Chair). Next, we evaluated the Chamfer distance between original shapes and reconstruction obtained in the first stage. We also gave the mse measure between original colors and the reconstructed ones.

In Table III we compare our double-stage strategy with the baseline strategy. As can be seen in Table III, our double-stage strategy yields better results in both shape reconstruction as well as in color space. Moreover, as we can see, the two-stage approach allows us to produce color coding without any loss of reconstruction quality.

C. 3-D Meshes Generation

The main advantage of our method is the ability to generate both autocolored 3-D point clouds and autocolored 3-D meshes. We achieve this without any need for postprocessing. In Fig. 7, we present autocolored meshes. Thanks to using a uniform distribution on the 3-D ball, we can easily construct a mesh of an object. All elements from the ball are transformed into a 3-D object.

In consequence, the unit sphere is transformed into the surface of the object. Thus, we can produce meshes without a secondary meshing procedure. It is obtained by propagating the triangulation of the 3-D sphere through the target network, as shown in Fig 5. When obtained a 3-D mesh, we can color its vertices in the second stage using a similar procedure as in the first stage. Next, we can produce colored meshes by simple interpolation. Our model is able to produce many different colors for single object (see Fig. 8). An example of the variety of coloring schemes can be seen in Fig. 9.

In our model, we can construct interpolation between objects. We have two prior distributions: Gaussian in the first and second stages. Therefore, we can produce smooth transition in shape and color space, see Fig. 10.

V. CONCLUSION

In this work, we have presented a hypernetwork [20] approach to synthesizing 3-D models of selected classes of real-life objects. Using our approach, we can swiftly generate any quantity of diversified 3-D models used for populating a given game scene. Furthermore, the conducted experiments suggest that our two-stage method gives better results in terms of shape reconstruction and coloring as compared to traditional single-stage techniques (see Table III).

Furthermore, the unique attribute in our method is the automatic and coloring of the 3-D models generated this way. As Figs. 1 and 2 show, such models can be easily embedded within the game scene background to compose plausible final effect.

Our work is only a first step toward developing ML-based methods for autogeneration of colored 3-D models. Hence, in the future, we plan to advance our method to produce 3-D models with finer details, i.e., with more complex meshes, including clear coloring borderlines.

Another exciting avenue of research is the automatic placement of the models in the game scene. Such an algorithm would require considering the correct orientation of the model concerning the scene’s topology and the placement of other models.
REFERENCES

[1] R. Hujsjer, J. Dobbe, W. F. Bronsvooort, and R. Bidarra, “Procedural natural systems for game level design,” in Proc. Braz. Symp. Games Digit. Entertainment, 2010, pp. 189–198.

[2] R. van der Linden, R. Lopes, and R. Bidarra, “Procedural generation of dungeons,” IEEE Trans. Comput. Intell. AI Games, vol. 6, no. 1, pp. 78–89, Mar. 2014.

[3] B. von R. Lipinski, S. Seibt, J. Roth, and D. Abé, “Level graph–incremental procedural generation of indoor levels using minimum spanning trees,” in Proc. IEEE Conf. Games, 2019, pp. 1–7.

[4] M. Arshad and W. J. Beksi, “A progressive conditional generative adversarial network for generating dense and colored 3-D point clouds,” in Proc. Int. Conf. 3-D Vis., Los Alamitos, CA, USA, 2020, pp. 712–722.

[5] A. Summerville et al., “Procedural content generation via machine learning (PCGML),” IEEE Trans. Games, vol. 10, no. 3, pp. 257–270, Sep. 2018.

[6] T. J. Rose and A. G. Bakouakous, “Algorithms and approaches for procedural terrain generation: A brief review of current techniques,” in Proc. 8th Int. Conf.+ of Games Virtual Worlds Serious Appl., 2016, pp. 1–2.

[7] K. Park, B. W. Mott, W. Min, K. E. Boyer, E. N. Wiebe, and J. C. Lester, “Generating educational game levels with multistep deep convolutional generative adversarial networks,” in Proc. IEEE Conf. Games, 2019, pp. 1–6.

[8] Z. Wu et al., “3-D shapenets: A deep representation for volumetric shapes,” Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2015, pp. 1912–1920.

[9] S. R. Richter and S. Roth, “Discriminative shape from shading in uncalibrated illumination,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2015, pp. 28–31.

[10] M. Tataruchenko, A. Dosovitskiy, and T. Brox, “Octree generating networks: Efficient convolutional architectures for high-resolution 3-D output,” in Proc. IEEE Int. Conf. Comput. Vis., 2017, pp. 2107–2115.

[11] G. Riegler, A. O. Ulusoy, and A. Geiger, “Octnet: Learning deep 3-D representations at high resolutions,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 484–499.

[12] J. Wu, C. Zhang, T. Xue, W. T. Freeman, and J. B. Tenenbaum, “Learning a probabilistic latent space of object shapes via 3-D generative-adversarial modeling,” in Proc. Adv. Neural Inf. Process. Syst., 2016, pp. 82–90.

[13] A. Kar, S. Tulsiani, J. Carreira, and J. Malik, “Category-specific object reconstruction from a single image,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2015, pp. 1966–1974.

[14] C. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese, “3D-R2N2: A unified approach for single and multi-view 3-D object reconstruction,” in Proc. Eur. Conf. Comput. Vis., 2016, pp. 628–644.

[15] S. R. Richter and S. Roth, “Matryoshka networks: Predicting 3-D geometry via nested shape layers,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2018, pp. 1936–1944.

[16] P. Spurek, M. Zieba, J. Tabor, and T. Trzcinski, “General hypernetwork framework for creating 3D point clouds,” IEEE Trans. Pattern Anal. Mach. Intell., to be published, doi: 10.1109/TPAMI.2021.3131131.

[17] A. Fadaeddini, B. Majidi, and M. Eshghi, “A case study of generative adversarial networks for procedural synthesis of original textures in video games,” in Proc. 2nd Nat. 1st Int. Digit. Games Res. Conf.: Trends, Technol., Appl., 2018, pp. 118–122.

[18] A. Maggiordomo, F. Ponchio, P. Cignoni, and M. Tarini, “Real-world textured things: Arepository of textured models generated with modern photo-reconstruction tools,” Comput. Aided Geometric Des., vol. 53, 2020, Art. no. 101943.

[19] P. Spurek, S. Winczowski, J. Tabor, M. Zamorski, M. Zieba, and T. Trzcinski, “Hypernetwork approach to generating point clouds,” in Proc. 37th Int. Conf. Mach. Learn., 2020, pp. 9099–9108.

[20] K. Mo et al., “PartNet: A large-scale benchmark for fine-grained and hierarchical part-level 3-D object understanding,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2019, pp. 909–918.

[21] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas, “Learning representations and generative models for 3-D point clouds,” in Proc. Int. Conf. Mach. Learn., 2018, pp. 40–49.

[22] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas, “Learning representations and generative models for 3-D point clouds,” in Proc. Int. Conf. Mach. Learn., 2018, pp. 40–49.